東海大學

資訊工程研究所

碩士論文

指導教授: 楊朝棟博士

基於 OpenStack 實作一個擁有虛擬機動態資源調配方法之雲端節能系統

Implementation of a Cloud Energy Saving System with Virtual Machine Dynamic Resource Allocation Method base on OpenStack

研究生: 陳建智

中華民國一零四年六月

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 陳 建 智 所提之論文

基於 OpenStack 實作一個擁有虛擬機動態資源調配方法之雲端節能系統

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 林迪彥 簽章

委 員 朱正忠

賴冠州

時文中

指 導 教 授 楊朝棟 簽章

中華民國 104 年 6 月 30 日

摘　要

美國國家標準與技術研究院（NIST）將雲端定義為：「雲端運算是一種模式，能方便且隨需求應變地透過連網存取廣大的共享運算資源（如網路、伺服器、儲存、應用程式、服務等），並可透過最少的管理工作及服務供應者互動，快速提供各項服務。」根據 Gartner 諮詢公司的分析，雲端運算是 2015 年對企業組織而言最重要的策略科技趨勢的前十大之一。所謂的策略科技 (strategic technology)，根據 Gartner 定義，指的是可能在未來三年對企業組織帶來重大影響的技術。各個企業、組織與學校也都跟隨著雲端的潮流，建立大規模的雲端運算叢集取代一人一電腦的情形。雖然虛擬化可以減少添購硬體設備的，但是也衍生出了兩個問題-能源的消耗與閒置資源的浪費。所以我們提出了兩個方法:1. 動態調配資源方法 2. 節電方法。如何有效節省並利用虛擬機於低負載時的閒置資源，與如何節省伺服器的能源消耗，是我們在本篇論文裡必須面對與解決的兩大問題。為了達到我們的目標，我們實現一個以雲端軟體 OpenStack 為基礎設施的平台並使用動態調配資源方法與節電方法來達到節省能源的目的。我們也將會利用 PDU 紀錄的耗電量來證明我們提出的方法是有用而且真的可以節省能源。

關鍵字: OpenStack，動態資源調配，能源節省，Live Migration，狀態監控

Abstract

The US National Institute of Standards and Technology (NIST) defines cloud computing as "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." According to analysis by Gartner, Inc., cloud computing is one of the top 10 strategic technology for most organizations in 2015. Gartner defines a strategic technology as one with the potential for significant impact on the organization in the next three years. Companies, organizations and academic institutions are following the cloud computing trend; the establishment of large-scale cloud computing clusters avoids the need to provide one person with one computer. Even though virtualization can reduce the cost of hardware equipment, but it still faces with two problems: energy consumption and the waste of the idle resources. To solve these two problems, we propose two algorithms, i.e., dynamic resource allocation and energy saving. In order to implement these two algorithms with live migration of virtual machines, we first build an infrastructure platform based on cloud software – OpenStack. Next, dynamic resource allocation and energy saving algorithms are designed and implemented. Finally, we use the Power Distribution Unit (PDU) to monitor system status and record power consumption; the real time status monitoring data verify that the proposed algorithms are efficient in energy saving and idle resource planning.

Keywords: OpenStack, Dynamic Resource Allocation, Energy Saving, Live Migration, Status Monitoring

致謝詞

一轉眼，兩年的研究所就結束了，也代表學生生活正式的結束了。研究所兩年，學到的東西多，但還是遠遠的不足，經過這兩年，感覺到自己的改變，改變得更成熟，對事情的看法也不再單一，能從更多角度去思考了。

我要感謝我的研究所指導老師楊朝棟教授，在我實驗卡關時，給了我能繼續往前的建議，還有提供能讓我上戰場的”武器”，沒有了老師提供的設備，我的實驗肯定會困難重重。也感謝老師讓我在畢業前去韓國參加 AINA 會議，學習新事務，拓展國際視野。

感謝抽空前來參加論文口試的委員們，謝謝系上劉榮春老師對我的研究提供了很多的意見、指導和鼓勵。謝謝林迺衛老師、朱正忠老師，賴冠州老師及時文中老師，給了我很多專業上的想法與建議，因為有您們的意見讓本來不完整的英文及鬆散的架構，在重整之後，讓我的論文能更加完整及嚴謹。

還要感謝實驗室的同學們，謝謝尹臻、getter、宇權、韻婷，在我遇到問題，總能幫助我解決，也謝謝昭為的 NAS，陪我度過這兩年。

謝謝我的家人、國小同學、國中同學、高中同學、女朋友在我背後支持我，不時地為我加油打氣。謝謝你們。

東海大學資訊工程學系 高效能實驗室 陳建智 104 年 07 月

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Cloud computing brings a huge change for industry evolved with the use of the Internet. Not only the IT industry which provides cloud computing technology, but also the general usage of it in the government, enterprise and individuals are changed with the born of cloud computing. In the IT industry, cloud computing has undoubtedly caused a comprehensive impact. Nearly all most basic computer components – processors, servers, storage devices, network equipment, information security equipment, software, data centers, information services, smart phone, tablet computer and other emerging mobile devices are unable to break off relations from cloud computing. In recent years, cloud computing has become one of the hottest topics. Cloud computing mainly combines virtualization, service management automation and standardized technology to provide flexible computing ability and data analysis method with high performance. Companies can run many kinds of service on the cloud platform without the need to construct data centers. This innovative computing and business model has attracted widespread attention in industry and academia.

1.1 Motivation

According to analysis by Gartner [1], Inc., cloud computing is one of the top 10 strategic technology trends for most organizations in 2015. Gartner defines a strategic technology as one with the potential for significant impact on the organization in the next three years. Factors that denote significant impact include a high potential for disruption to the business, end users or IT, the need for a major investment, or the risk of being late to adopt. These technologies impact the organization's long-term plans, programs and initiatives. As shown in Figure 1.1, during 2011 to 2015, cloud computing is always listed as one of the top 10 strategic technology trends.

Year	2011	2012	2013	2014	2015
1	Cloud Computing	Media Tablets and Beyond	Mobile Device Battles	Mobile Device Diversity and Management	Computing Everywhere
2	Mobile Applications and Media Tablets	Mobile-Centric Applications and Interfaces	Mobile Apps and HTML5	Mobile Apps and Applications	The Internet of Things
3	Social Communications and Collaboration	Contextual and Social User Experience	Personal Cloud	The Internet of Everything	3D Printing
4	Video	Internet of Things	Enterprise App Stores	Hybrid Cloud and IT as Service Broker	Advanced, Pervasive and Invisible Analytics
5	Next Generation Analytics	App Stores and Marketplaces	The Internet of Things	Cloud/Client Architecture	Context-Rich Systems
6	Social Analytics	Next-Generation Analytics	Hybrid IT and Cloud Computing	The Era of Personal Cloud	Smart Machines
7	Context-Aware Computing	Big Data	Strategic Big Data	Software Defined Anything	Cloud/Client Computing
8	Storage Class Memory	In-Memory Computing	Actionable Analytics	Web-Scale IT	Software-Defined Applications and Infrastructure
9	Ubiquitous Computing	Extreme Low-Energy Servers	In Memory Computing	Smart Machines	Web-Scale IT
10	Fabric-Based Infrastructure and Computers	Cloud Computing	Integrated Ecosystems	3-D Printing	Risk-Based Security and Self-Protection

FIGURE 1.1: The Top 10 Strategic Technology Trends during 2011 to 2015

Cloud computing is the trend of today's IT industry. Companies, organizations and academic institutions are following the cloud computing; the establishment of large-scale cloud computing clusters replaces the old case of one person with one computer. Even though virtualization can reduce the cost of hardware equipment, but it still spawns an issue – energy consumption. Virtual machines (VMs) with many kinds of service can run on the cloud cluster, but these services may not

be always accessed and remain high loading all the time. Therefore, we propose two methods: Dynamic Resource Allocation (DRA) [2–5] method and Energy Saving [6–8] method. How to effectively save and use the idled resources on low loading VMs, and save energy consumption on servers are the two issues we have to face and solve.

1.2 Thesis Goal and Contributions

The goal of this work is to implement a cloud system consisting of the effective DRA method, energy saving method, user interface, and status monitoring unit. The proposed system is built based on the software OpenStack, an infrastructure platform for the cloud. Next, we use two algorithms to improve traditional DRA and energy saving methods. By the improved DRA algorithm, we can reduce idled resource on the VMs. The improved energy saving algorithm reduces the energy consumption of the entire cloud cluster. All servers will be connected to a PDU and the power consumption of them are recorded and monitored. The results of the PDU power consumption records verify that the two proposed algorithms are effective.

1.3 Thesis Organization

In Chapter 2, we review background information including cloud computing, virtualization, hypervisor, OpenStack, live migration, NFS, PDU and related work. Chapter 3 introduces the architecture of our system and its implementation. Chapter 4 shows the experimental environment, results and analysis. Chapter 5 gives some conclusions and future work.

Chapter 2

Background Review and Related Work

In order to improve conventional dynamic allocation resource and energy saving methods, this section reviews some background information, including cloud computing, virtualization, hypervisor, OpenStack, live migration, NFS, PDU and related work.

2.1 Background Review

2.1.1 Cloud Computing

The NIST definition of cloud computing is: "a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction." This model of cloud computing consists of five essential characteristics, three service models, and four deployment models, as shown in Figure 2.1:

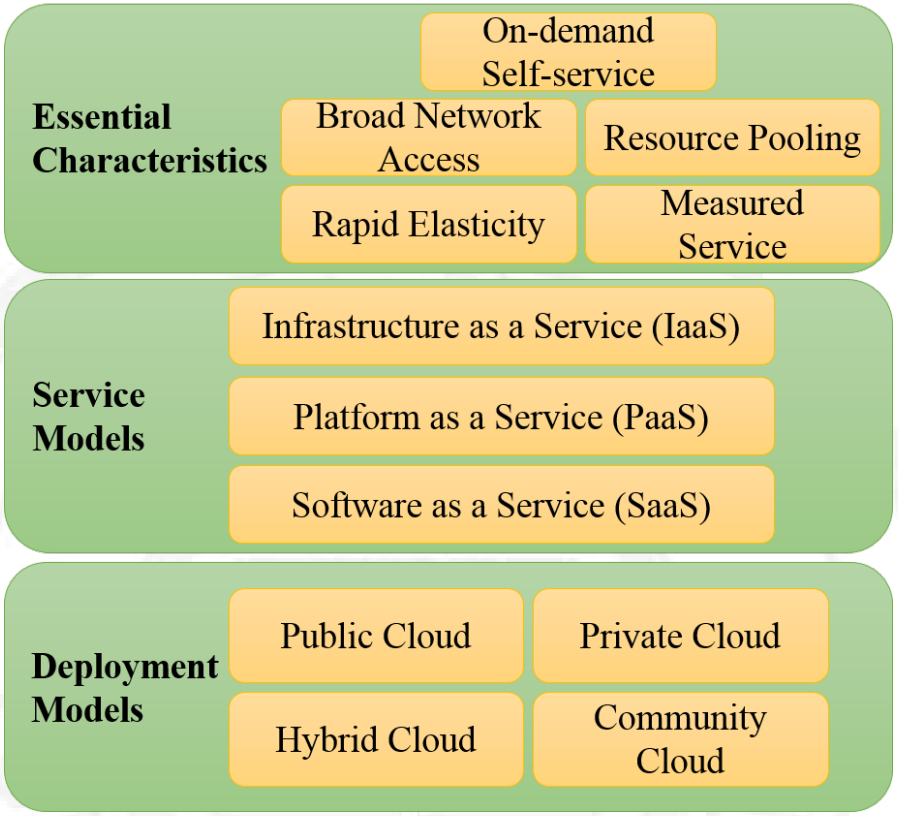


FIGURE 2.1: The overall cloud model

According to NIST definition of cloud computing, it identifies "five essential characteristics":

- On-demand Self-service
- Broad Network Access
- Resource Pooling
- Rapid Elasticity
- Measured Service

The so-called cloud computing service type is able to provide the service to users, and allows users to obtain resources through such a service. According to the NIST definition, cloud service architecture follows the service type divided into three layers, namely, infrastructure as a service (IaaS), Platform as a Service (PaaS) and software as a service (SaaS). And are introduced as follows:

- Infrastructure as a Service (IaaS): The capability provided to the consumer is to offer processing, storage, networks, and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, and which can include operating systems and applications. The consumer does not manage or control the underlying cloud infrastructure but has control over operating systems, storage, and deployed applications; and possibly limited control of select networking components (e.g., host firewalls).

- Platform as a Service (PaaS): The capability provided to the consumer is to deploy onto the cloud infrastructure consumer-created or acquired applications created using programming languages, libraries, services, and tools supported by the provider. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, or storage, but has control over the deployed applications and possibly configuration settings for the application-host environment.

- Software as a Service (SaaS): The capability provided to the consumer is to use the provider's applications running on a cloud infrastructure. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. The consumer does not manage or control the underlying cloud infrastructure including network, servers, operating systems, storage, or even individual application capabilities, with the possible exception of limited user-specific application configuration settings.

Cloud computing, according to their ownership of providers and users, can be divided into four categories, namely the public, private, community, and hybrid cloud.

- Public Cloud: The cloud infrastructure is provided for open use by the general public. It may be owned, managed, and operated by a business, academic, or government organization, or some combination of them. It exists on the premises of the cloud provider.

- Private Cloud: The cloud infrastructure is provided for exclusive use by a single organization comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organization, a third party, or some combination of them, and it may exist on or off premises.

- Hybrid Cloud: The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, community, or public) that remain unique entities, but are bound together by standardized or proprietary technology that enables data and application portability (e.g., cloud bursting for load balancing between clouds).

- Community Cloud: The cloud infrastructure is provided for exclusive use by a specific community of consumers from organizations that have shared concerns (e.g., mission, security requirements, policy, and compliance considerations). It may be owned, managed, and operated by one or more of the organizations in the community, a third party, or some combination of them, and it may exist on or off premises.

2.1.2 Virtualization

In computing, virtualization [9–13] refers to the act of creating a virtual (rather than actual) version of something, including (but not limited to) a virtual computer hardware platform, operating system, storage device, or computer network resources. With virtualization, the computer's physical resources, such as servers, network, memory, and storage, are abstractly presented after conversion, so that users can apply those resources in a better way than the original configuration. Simply put, virtualization is a technology that allows the user to transform hardware into software, and it allows the user to run multiple operating systems simultaneously on a single computer.

Virtual architecture is different from traditional architecture, as shown in Figure 2.2. Traditional architecture can run single operating system on a single computer, but the virtual architecture can run multiple operating systems on a single computer.

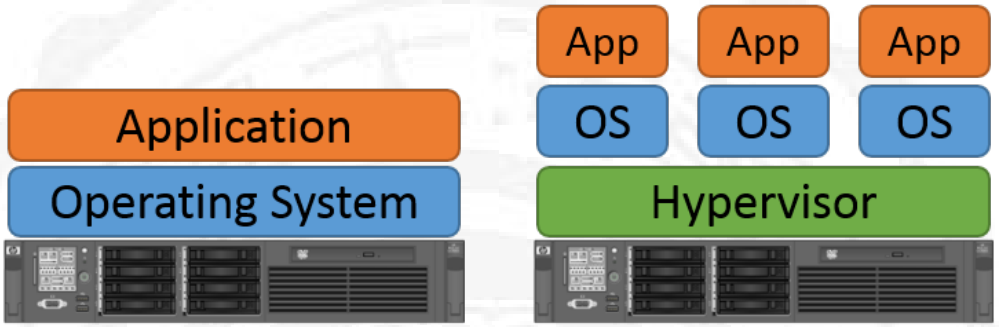


FIGURE 2.2: The different between traditional architecture and virtual architecture

There are many benefits of virtualization, such as:

- Encapsulation - VMs can be described in a file
  - Possible to ”snapshot”
  - Easy to move
- Enables running multiple operating systems
- Consolidation and use of unused computation power
- Resource management
- High availability and disaster recovery
- Create ”Base Environment”
- Safe testing of new software
- Easy Management

2.1.3 Hypervisor

A hypervisor [14] or virtual machine monitor (VMM) is a piece of computer software, firmware or hardware that creates and runs VMs. A computer on which a hypervisor is running one or more VMs is defined as a host machine. Each VM is called a guest machine. The hypervisor presents the guest operating systems with a virtual operating platform and manages the execution of the guest operating systems. Multiple instances of a variety of operating systems may share the virtualized hardware resources.

Simply stated, a hypervisor creates a layer of abstraction that isolates an OS and its associated applications from the underlying computing hardware. The isolation effectively mitigates software from its traditional reliance on hardware devices and their drivers. The implications of this behavior are profound. A hypervisor allows OSes and their application workloads to run on a broader array of hardware. Similarly, multiple OSes and workloads, each a unique VM or VM instance, can reside on the same system to simultaneously share computing resources. Each VM can be migrated between computing platforms on demand with little (if any) processing disruption. The result is better use of computing platforms with seamless workload migration and backup capabilities. Hypervisors generally fall into two categories: hosted and bare-metal. Both offer distinct benefits and drawbacks.

A hosted hypervisor, as shown in Figure 2.3, runs within the OS and allows additional OS and application instances to run on top of it. Examples of the hosted hypervisors include VMware Server and Microsoft Virtual Server, as well as numerous endpoint-based virtualization platforms like VMware Workstation, Microsoft Virtual PC and Parallels Workstation.

There are advantages of the hosted hypervisor:

- Virtualization installs like application rather than like OS.
- Can run alongside conventional applications.

- Avoid code duplication – OS already has process scheduler, memory management, device support etc.
- More suitable for personal users.

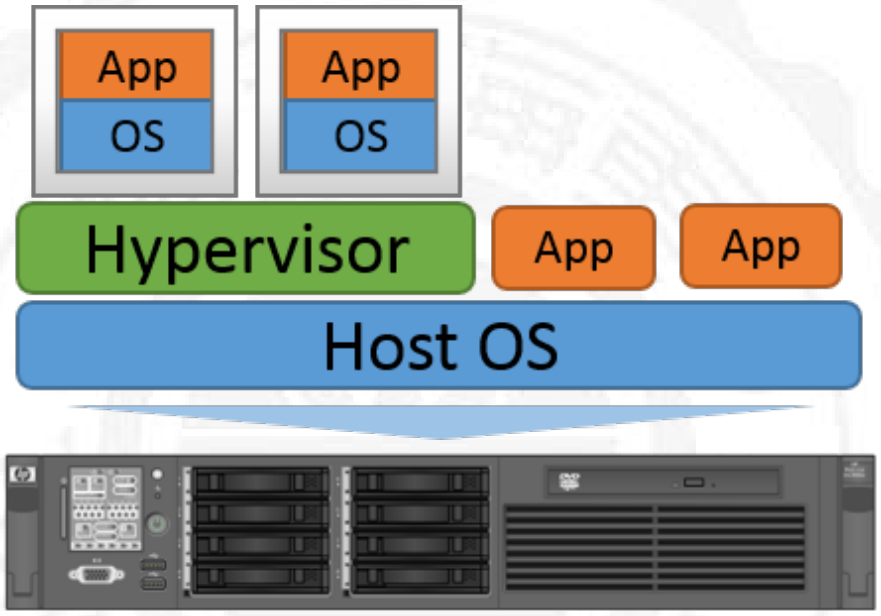


FIGURE 2.3: The hosted hypervisor architecture

The bare-metal hypervisor, as shown in Figure 2.4, is the most commonly deployed type, and it can be installed directly onto the computing hardware. Its OS installs and runs above the hypervisor. Major virtualization products that can be termed as bare-metal hypervisors include Oracle VM, VMware ESXi, Microsoft Hyper-V and Citrix XenServer.

There are advantages of Bare-Metal Hypervisor:

- Better performance with lower overhead.
- Highly efficient direct I/O pass-through architecture for network and disk.
- Complete control over hardware.
- Advanced features like live migration available.
- Suitable for production environments.

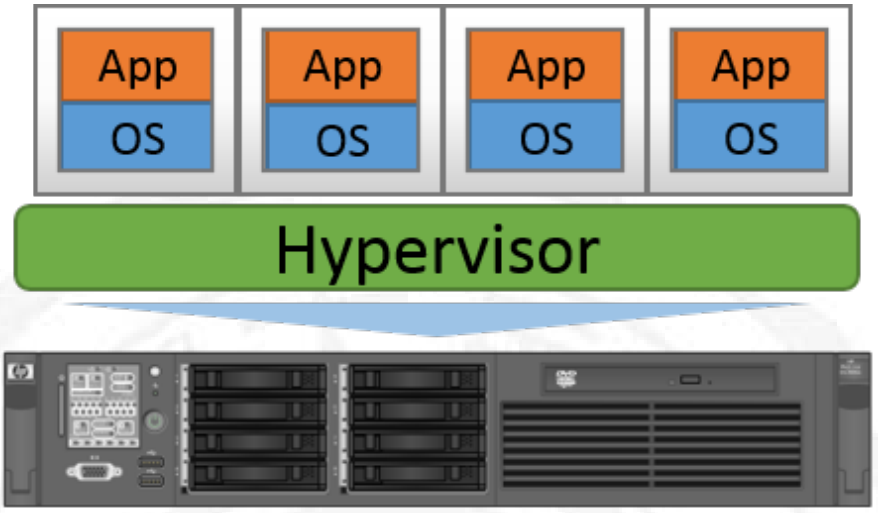


FIGURE 2.4: The bare-metal hypervisor architecture

2.1.4 OpenStack

OpenStack [15–18] is a free and open-source cloud computing software platform. It began in 2010 as a joint project of Rackspace Hosting and NASA. Currently, it is managed by the OpenStack Foundation, a non-profit which oversees both development and community-building around the project. And OpenStack.org released it under the terms of the Apache License. Users primarily deploy it as an IaaS solution. The technology consists of a series of interrelated projects that control pools of processing, storage, and networking resources throughout a data center which users manage through a web-based dashboard, command-line tools, or a RESTful API.

OpenStack has a modular architecture with various code names for its components.

- Compute (Nova)

  OpenStack Compute (Nova) is a cloud computing fabric controller, which is the main part of an IaaS system. It is designed to manage and automate pools of computer resources and can work with widely available virtualization technologies, as well as bare metal and high-performance computing (HPC) configurations. KVM, VMware, and Xen are available choices for hypervisor

technology, together with Hyper-V and Linux container technology such as LXC.

- Object Storage (Swift)

  OpenStack Object Storage (Swift) is a scalable redundant storage system. Objects and files written to multiple disk drives spread throughout servers in the data center, with the OpenStack software responsible for ensuring data replication and integrity across the cluster. Storage clusters scale horizontally simply by adding new servers. Should a server or hard drive fail, OpenStack replicates its content from other active nodes to new locations in the cluster. Because OpenStack uses software logic to ensure data replication and distribution across different devices, inexpensive commodity hard drives and servers can be used. The Total Cost of Ownership (TCO) can be higher than using enterprise-class storage because many copies require high availability.

- Block Storage (Cinder)

  Cinder is a block storage service for OpenStack. It is designed to allow the use of either a reference implementation (LVM) to present storage resources to end users that can be consumed by the OpenStack Compute Project (Nova). The short description of Cinder is that it virtualizes pools of block storage devices and provides end users with a self service API to request and consume those resources without requiring any knowledge of where their storage is actually deployed or on what type of device.

- Networking (Neutron)

  OpenStack Networking (Neutron, formerly Quantum) is a system for managing networks and IP addresses. OpenStack Networking ensures the network is not a bottleneck or limiting factor in a cloud deployment, and gives users self-service ability, even over network configurations.

- Dashboard (Horizon)

OpenStack Dashboard (Horizon) provides administrators and users a graphical interface to access, provision, and automate cloud-based resources. The design accommodates third party products and services, such as billing, monitoring, and additional management tools. The dashboard can also be branded for service providers and other commercial vendors who want to make use of it. The dashboard is one of several ways users can interact with OpenStack resources. Developers can automate access or build tools to manage resources using the native OpenStack API or the EC2 compatibility API.

- Identity Service (Keystone)

  OpenStack Identity (Keystone) provides a central directory of users mapped to the OpenStack services they can access. It acts as a common authentication system across the cloud operating system and can integrate with existing backend directory services like LDAP. It supports multiple forms of authentication including standard username and password credentials, token-based systems and AWS-style (i.e. Amazon Web Services) logins. Additionally, the catalog provides a list of all of the services deployed in an OpenStack cloud in a single registry. Users and third-party tools can determine which resources they can access by programs.

- Image Service (Glance)

  OpenStack Image Service (Glance) provides discovery, registration, and delivery services for disk and server images. Stored images can be used as a template. It can also be used to store and catalog an unlimited number of backups. The Image Service can store disk and server images in a variety of back-ends, including OpenStack Object Storage. The Image Service API provides a standard REST interface for querying information about disk images and lets clients stream the images to new servers.

- Telemetry (Ceilometer)

  OpenStack Telemetry Service (Ceilometer) provides a single point of contact for billing systems, providing all the counters they need to establish

customer billing, across all current and future OpenStack components. The delivery of counters is traceable and auditable, the counters must be easily extensible to support new projects, and agents doing data collections should be independent of the overall system.

- Orchestration (Heat)

  Heat is the main project in the OpenStack Orchestration program. It implements an orchestration engine to launch multiple composite cloud applications based on templates in the form of text files that can be treated like code. A native Heat template format is evolving, but Heat also attempts to provide compatibility with the AWS CloudFormation template format, so that many existing CloudFormation templates can be launched on OpenStack. Heat provides both an OpenStack-native ReST API and a CloudFormation-compatible Query API.

- Database (Trove)

  Trove is Database as a Service for OpenStack. It is designed to run entirely on OpenStack, with the goal of letting users to quickly and easily utilize the features of a relational or non-relational database without the burden of handling complex administrative tasks. Cloud users and database administrators can offer and manage multiple database instances as needed. Initially, the service will focus on providing resource isolation at high performance while automating complex administrative tasks such as deployment, configuration, patching, backups, restores, and monitoring.

2.1.5 OpenStack Conceptual Architecture

Launching a VM or instance involves many interactions among several services. Figure 2.5 provides the conceptual architecture of a typical OpenStack environment.

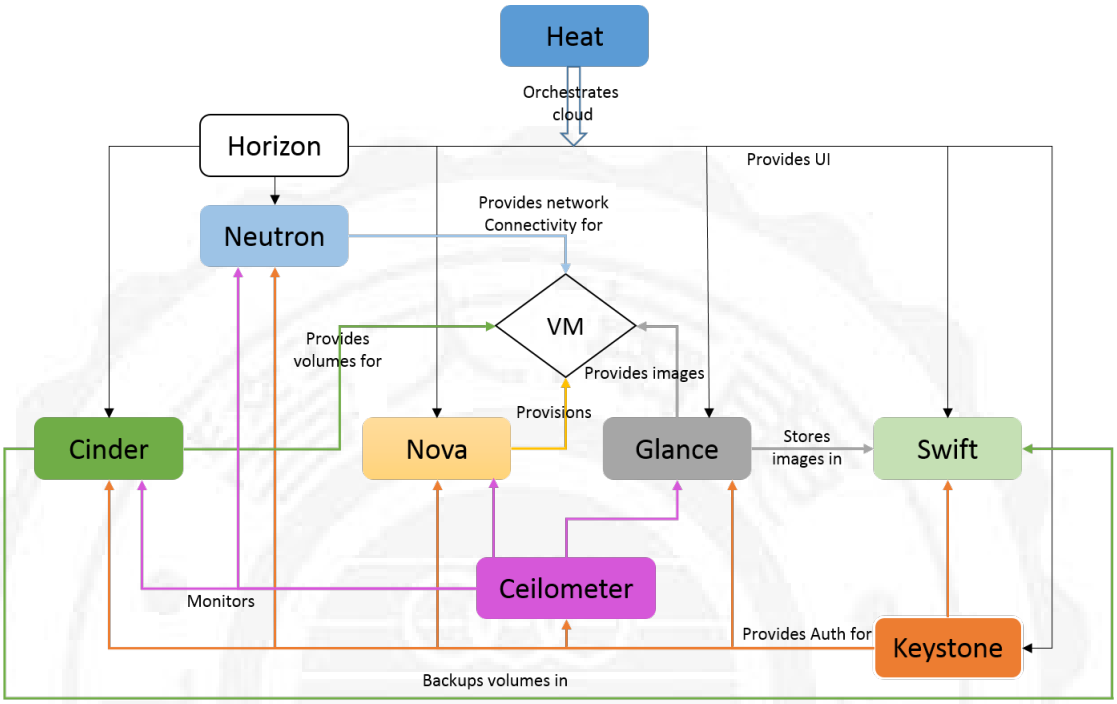


FIGURE 2.5: The conceptual architecture of OpenStack

In this work, we use version IceHouse. We just use Nova, Glance, Keystone and Horizon in our model.

2.1.6 Live Migration

Live migration [20–24],as shown in Figure 2.6, refers to the process of moving a running VM or application between different physical machines without disconnecting the client or application. Memory, storage, and network connectivity of the VM are transferred from the original guest machine to the destination.

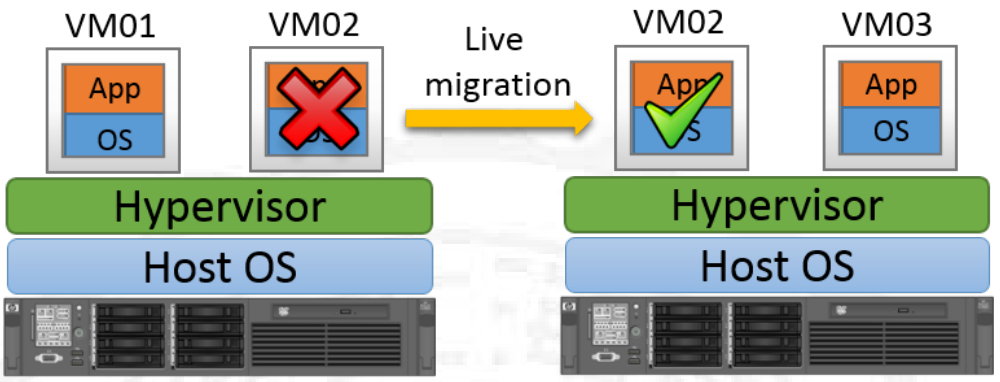


FIGURE 2.6: The concept of Live Migration

Two techniques for moving the VM's memory state from the source to the destination are pre-copy memory migration and post-copy memory migration.

- Pre-copy memory migration, as shown in Figure 2.7.
  - Warm-up phase

    In pre-copy memory migration, the hypervisor typically copies all the memory pages from source to destination while the VM is still running on the source. If some memory pages change (become 'dirty') during this process, they will be re-copied until the rate of re-copied pages is not less than the page dirty rate.
  - Stop-and-copy phase

    After the warm-up phase, the VM will be stopped on the original host, the remaining dirty pages will be copied to the destination, and the VM will be resumed on the destination host. The time between stopping the VM on the original host and resuming it on destination is called "down-time", and it ranges from a few milliseconds to seconds according to the size of memory and applications running on the VM. There are some techniques to reduce live migration down-time, such as using probability density function of memory change.

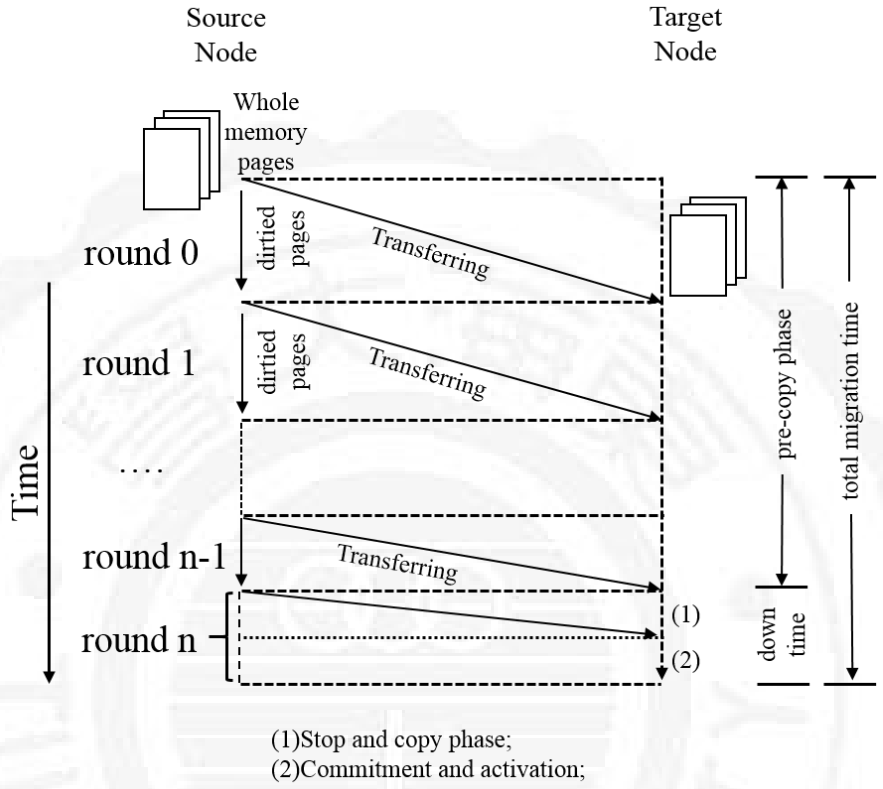


FIGURE 2.7: The phase of Pre-copy memory migration

- Post-copy memory migration

  Post-copy VM migration is initiated by suspending the VM at the source. With the VM suspended, a minimal subset of the execution state of the VM (CPU state, registers and, optionally non-pageable memory) is transferred to the target. The VM is then resumed at the target. Concurrently, the source actively pushes the remaining memory pages of the VM to the target - an activity known as pre-paging. At the target, if the VM tries to access a page that has not yet been transferred, it generates a page-fault. These faults, known as network faults, are trapped at the target and redirected to the source, which responds with the faulted page. Too many network faults can degrade performance of applications running inside the VM. Hence pre-paging can dynamically adapt the page transmission order to network faults by actively pushing pages in the vicinity of the last fault. An ideal pre-paging scheme would mask large majority of network faults, although its

performance depends upon the memory access pattern of the VM's workload. Post-copy sends each page exactly once over the network. In contrast, pre-copy can transfer the same page multiple times if the page is dirtied repeatedly at the source during migration. On the other hand, pre-copy retains an up-to-date state of the VM at the source during migration, whereas with post-copy, the VM's state is distributed over both source and destination. If the destination fails during migration, pre-copy can recover the VM, whereas post-copy cannot.

2.1.7 NFS (Network File System)

Network File System (NFS) [25–27], as shown in Figure 2.8, is a distributed file system protocol originally developed by Sun Microsystems in 1984, allowing a user on a client computer to access files over a network much like the local storage. NFS, like many other protocols, builds on the Open Network Computing Remote Procedure Call (ONC RPC) system. The Network File System is an open standard defined in Request for Comments (RFCs), allowing anyone to implement the protocol. Even though different universities and laboratories have developed a variety of distributed file systems, NFS is the first product is applicable for both academic and commercial use.

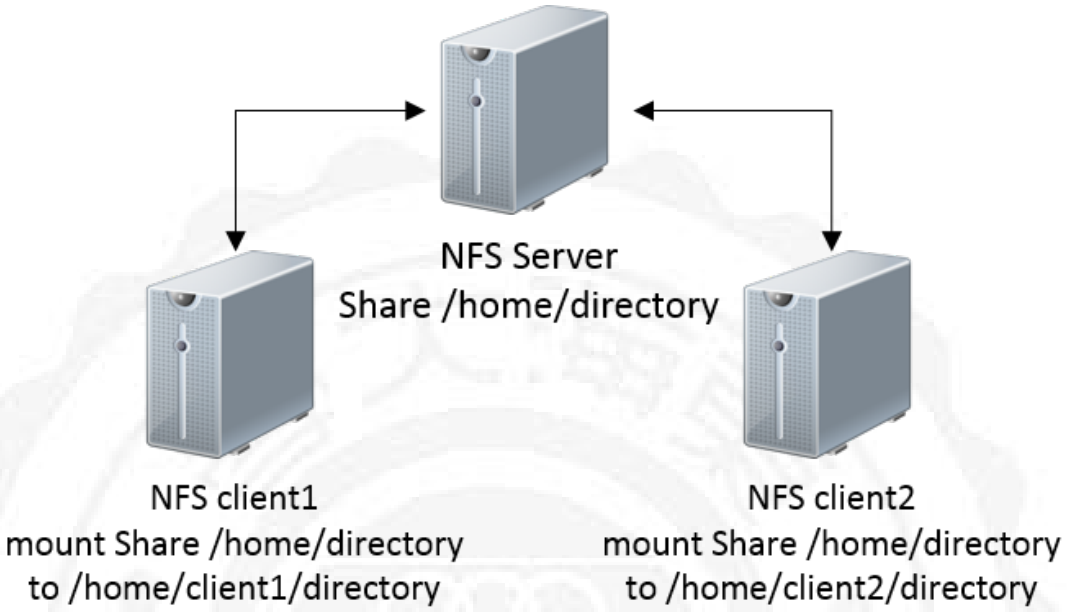


FIGURE 2.8: The Network File System (NFS)

NFS's basic principle is "to allow different clients and server nodes share the same file system through a set of RPC", thus, independent of the operating system, NFS allows different hardware and operating systems to share a common file system. NFS provides the following services:

- Search file in the directory.
- List the files in the directory.
- Manage Directory.
- Obtain attribute of all files.
- The file read / write

NFS is often used with Unix operating systems (such as Solaris, AIX and HP-UX) and Unix-like operating systems (such as Linux and FreeBSD). It is also available to operating systems such as the classic Mac OS, OpenVMS, IBM i, certain editions of Microsoft Windows, and Novell NetWare, and alternative remote file access protocols including the Server Message Block (SMB, also known as CIFS), Apple

Filing Protocol (AFP), NetWare Core Protocol (NCP), and OS/400 File Server file system (QFileSvr.400).

2.1.8 PDU (Power Distribution Units)

A Power Distribution Unit (PDU) [28, 29],as shown in Figure 2.9, is a device used in data centers to distribute AC power to multiple servers and other equipment. The PDUs range from simple 120 volts power strips to units that break out 120 volts from 240 volts and three-phase power. Advanced units can be managed remotely via the SNMP management protocol or from a Web browser or other management console, enabling outlets to be turned on and off at prescribed times and in a proper sequence to shut down and power up equipment.

FIGURE 2.9: Raritan's PDU

The growing complexity of IT environments, from wiring closets and server rooms to data centers of all sizes, has increased the need for reliable power distribution to the rack level. Eliminating power management problems is essential for IT and facilities managers to maintain system availability of increasing higher density equipment. The PDU is an essential element in managing power capacity and functionality for critical networks, servers and data center equipment.

- Basic PDU

  The most basic PDU is a large power strip without surge protection. It is designed to provide standard electrical outlets for data center equipment and has no monitoring or remote access capabilities. The floor-mounted and rack-mounted PDUs can be more sophisticated, providing data that can be used for power usage effectiveness (PUE) calculations.

- Floor-mounted PDU

  A floor-mounted PDU, sometimes called a main distribution unit (MDU), provides an important management bridge between a building's primary power and various equipment racks within a data center or network operations center (NOC). Each PDU can handle larger amounts of energy than an ordinary power strip (300 kilovolt-amps or higher depending on the manufacturer and model) and typically provides power to multiple equipment racks.

- Rack-mountable PDU

  A rack-mountable PDU mounts directly to an equipment rack so it can control and monitor power to specific servers, switches and other data center devices and assist in balancing power loads. Rack-mountable PDAs are known by several different names, including smart PDUs and intelligent PDUs. Such PDUs include three-phase displays for devices sharing power and remote management tools that use the Simple Network Management Protocol (SNMP) to provide administrators with the ability to adjust and monitor power demands from offsite locations.

2.2 Related Work

In the recent years, there are many power-saving and live migration research. We choose some related research about power-saving, live migration, and dynamic resource allocation for discussion.

As shown in [6], after a number of relevant research and experiments, the authors obtained some conclusions from analysis of the experimental results. Live migration of VMs will not cause significant additional electricity costs. The proposed power-saving method is proven to be effective. And about 7% to 14% saving of power consumption is achieved in their design. The percentage of power saving depends on not only the power-saving method but also the real operation situations of VMs and hosts. Their main function of the power-saving method is to obtain and analyze resource usage information of servers and VMs through the resource status monitoring program. If the required resources of VMs are less than those supported by currently running servers, then via Libvirt live migration instructions the VMs are centralized and some server is shut down to achieve the goal of energy saving. And as the resource requirement of the whole system increases, some of standby servers might be awakened to join the computing cluster, and appropriate live migrations of VMs are performed among the operating servers. Their power-saving method is shown in Figure 2.10.

Algorithm III.3: POWER-SAVING METHOD(c)

$sumvm \leftarrow all\ VM's\ computation\ sum$
$sumhost \leftarrow all\ host's\ computation\ sum$
$maxvmn \leftarrow vm\ with\ maximum\ computation\ on\ host\ n$
$maxhost \leftarrow host\ with\ largest\ VM's\ computation\ sum$
$minhost \leftarrow host\ with\ smallest\ VM's\ computation\ sum$
$minsumhost \leftarrow host\ with\ smallest\ computation\ sum$
if $sumvm > sumhost$
then $\begin{cases} start\ an\ host\ in\ shutdown\ status \\ migrate\ maxvmmaxhost\ to\ the\ host\ just\ start \end{cases}$
else if $sumvm < sumhost - minsumhost$
then $\begin{cases} migrate\ all\ VM\ on\ minhost\ to\ other\ host \\ shutdown\ minhost \end{cases}$

FIGURE 2.10: The Power-Saving method of first paper

The authors in [30] proposed a power saving algorithm as shown in Figure 2.11, including the power saving method program and its work flow, where $hostR$ is resource usage of host i, $hostN$ is the host network usage, $sumVM$ is sum of VM resource usage in the host, and $VMmax$ is the VMs used the most resources in

the host. Their proposed method first checks the system status data; if *hostR* or *hostN* is more than the hard cap the host needs to perform load balance; if the system has no hosts that can be loaded with more VMs, the method will turn on a host and has the $VMmax$ to migrate to other hosts; if the host is still in high loading, other VMs on it will be migrated until the host is with loading lower than the hard cap; if the *hostR* is lower than the soft cap, the host may be merged, and the method will find a host to turn it off if the sum of two host VMs loading is lower than the soft cap. Because over the soft cap the host performance is down, and if after merging the host loading is over the hard cap the host needs to perform load balance; hence, the merge causes wastes of power and time.

Algorithm 3.4.1: POWER-SAVING METHOD()

if (*hostR or hostN*) $>$ *hard cap*
then
- **if** (*need add one host*)
  - **then** { *turn on one host*
  - **do** *LoadBalance*
    - *find VMmax*
    - *migrate VMmax to other host*

else if (*hostR or hostN*) $<$ *soft cap*
then
- *find other host hostR* $<$ *soft cap*
- **if** (*sumVM* $<$ *soft cap*)
  - **then**
    - *merge VM to one host*
    - *shut down the idle host*

FIGURE 2.11: The Power-Saving method of second paper

In [5], the authors studied dynamic resource allocation. Their work advances the cloud computing field in two ways. First, it plays a significant role in the reduction of data center energy consumption costs, and thus helps to develop a strong and competitive cloud computing industry. Second, consumers are increasingly becoming conscious about the environment. A recent study shows that data centers represent a large and rapidly growing energy consumption sector of the economy and a significant source of CO2 emissions. Reducing greenhouse gas emissions is a

key energy policy focus of many countries around the world. The authors presented and evaluated energy-aware resource allocation algorithms utilizing the dynamic consolidation of VMs. Their experiment results show that their approach leads to a substantial reduction of energy consumption in cloud data centers in comparison to static resource allocation techniques. The authors aim at putting in a strong thrust on open challenges identified in their paper to enhance the energy-efficient management of cloud computing environments. They proposed Minimization of Migrations (MM) policy to select the minimum number of VMs needed to migrate from a host to lower the CPU utilization below the upper utilization threshold if the upper threshold is violated. The pseudo-code for the MM algorithm for the over-utilization case is presented in Figure 2.12. The algorithm sorts the list of VMs in the decreasing order of the CPU utilization. Then, it repeatedly looks through the list of VMs and finds a VM that is the best to migrate from the host. The best VM is the one that satisfies two conditions. First, the VM should have the utilization higher than the difference between the host's overall utilization and the upper utilization threshold. Second, if the VM is migrated from the host, the difference between the upper threshold and the new utilization is the minimum across the values provided by all the VMs. If there is no such a VM, the algorithm selects the VM with the highest utilization, removes it from the list of VMs, and proceeds to a new iteration. The algorithm stops when the new utilization of the host is below the upper utilization threshold. The complexity of the algorithm is proportional to the product of the number of over-utilized hosts and the number of VMs allocated to these hosts.

```
1  Input: hostList  Output: migrationList
2  foreach h in hostList do
3      vmList ← h.getVmList()
4      vmList.sortDecreasingUtilization()
5      hUtil ← h.getUtil()
6      bestFitUtil ← MAX
7      while hUtil > THRESH_UP do
8          foreach vm in vmList do
9              if vm.getUtil() > hUtil − THRESH_UP then
10                 t ← vm.getUtil() − hUtil + THRESH_UP
11                 if t < bestFitUtil then
12                     bestFitUtil ← t
13                     bestFitVm ← vm
14             else
15                 if bestFitUtil = MAX then
16                     bestFitVm ← vm
17                 break
18         hUtil ← hUtil − bestFitVm.getUtil()
19         migrationList.add(bestFitVm)
20         vmList.remove(bestFitVm)
21     if hUtil < THRESH_LOW then
22         migrationList.add(h.getVmList())
23         vmList.remove(h.getVmList())
24 return migrationList
```

FIGURE 2.12: The algorithm of Minimization of Migrations(MM)

In [20], the authors presented the design, implementation and evaluation of memory-compression-based VM migration approach (MECOM for short), which first introduces memory compression technique into live VM migration. Based on memory page characteristics, they designed a particular memory compression algorithm for live migration of VMs. Based on the analysis of memory data characteristics, they first classified pages into the following kinds: (1) pages composed of a great many of zero bytes and sporadic nonzero bytes; (2) pages with high word-similarity; (3) pages with low word-similarity. For the first kind of pages, we can scan the whole page and just record the information about the offset and value of nonzero bytes. For the second kind of pages, we can use methods that embody strong similarities, such as WKdm, which is a unique combination of dictionary and statistic techniques specifically designed to quickly and efficiently compress memory data. The last kind of pages has weak regularities, and then a universal approach with a high

compression ratio is appropriate. LZO, a modern implementation of Lempel–Ziv compression, is an option. Because the smaller amount of data is transferred and only very low compression overhead is introduced, the total migration time and downtime are both decreased significantly. Service degradation is also decreased greatly. Their experimental results show that their system can get better average performance than Xen: up to 27.1% on VM downtime, upto 32% on total migration time, and up to 68.8% data cut down that must be transferred. Their MECOM structure is shown in Figure 2.13.

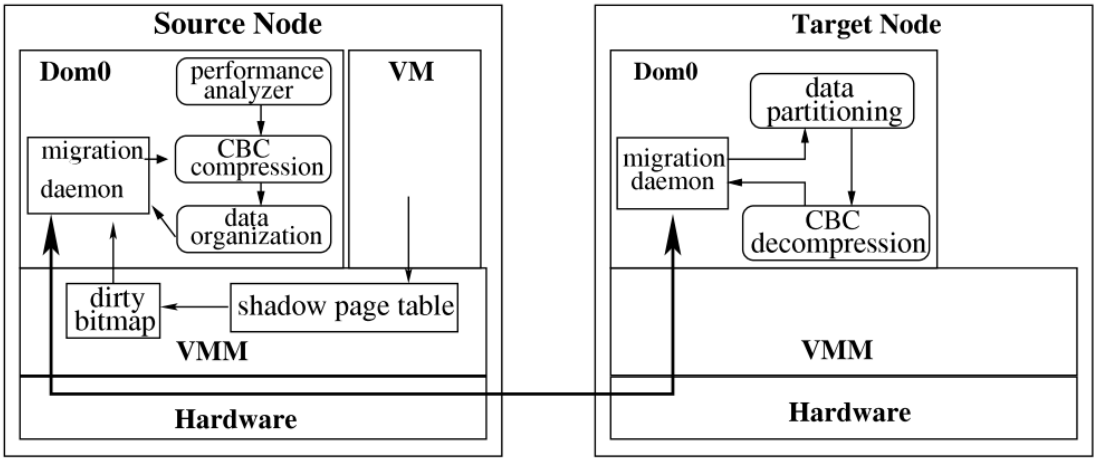


FIGURE 2.13: The MECOM structure

Chapter 3

System Design and Implementation

With the popularity of cloud computing, using the idle resources of VMs with low loading and saving energy consumption on servers are the two issues particularly worthy of study. In this section, we first build a cloud platform based on the infrastructure software OpenStack and then use the platform to implement a system with the proposed algorithms to solve these two issues. In addition, the system also provides a user interface.

3.1 System Architecture

This section introduces the architecture of the proposed cloud platform based on the infrastructure software OpenStack. Figure 3.1 shows the architecture. The architecture consists of a controller node and two computing nodes. Several major OpenStack services are running on the Controller node, such as Identity service, Image service, Networking service, Nova service, and Dashboard. To perform live migration on the VMs between the two computing nodes, we use Network File System (NFS) as shared storage and install the NFS server on the controller node.

The two computing nodes are only utilized to run the Nova service and NFS client and connected to the PDU for monitoring and recording their energy consumption.

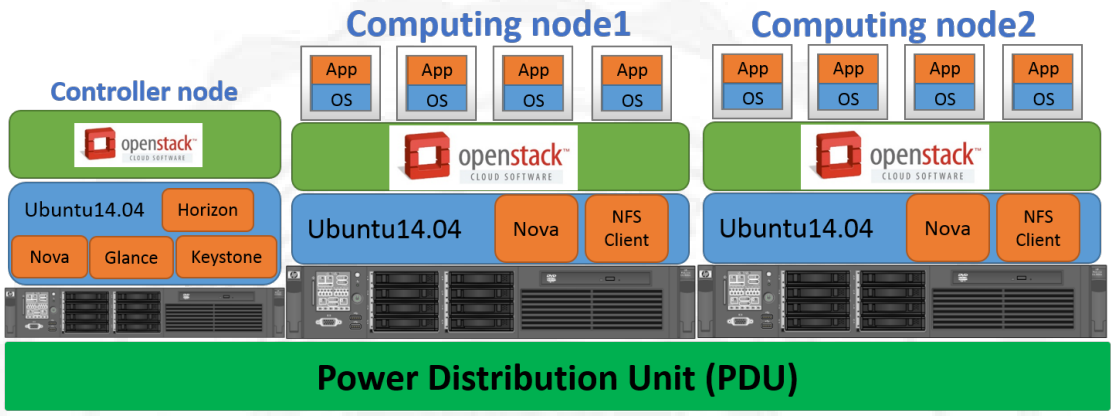


FIGURE 3.1: The overall system architecture

3.2 Design Flow

To achieve efficient dynamic allocation of resources and energy saving, two algorithms are designed based on the proposed cloud platform. The algorithmic design flow for DRA and the algorithmic design flow for energy saving are introduced in subsection 3.2.1 and subsection 3.2.2, respectively.

3.2.1 Design Flow of DRA method

First we calculate resource utilization of each VM. By calculating resource utilization of each VM, we can determine whether its allocated resources are excessive or inadequate. If there are excessive resources on a VM, we will reduce its resources. If there are inadequate resources on a VM, we will check whether there are enough resources on the host that the VM is located. If enough, we will increase the resources of the VM directly from the host. If not enough, we will check whether there is another available host; if not, we will turn on a shut-down host with enough resources to be allocated to the VM. We then perform live migration of the VM to the host with enough resources, and after that, increase the resources

of the VM. If the resources of a VM are not excessive or inadequate, we will not increase or decrease the resources of it, and then continue checking resources of the next VM. The flowchart of the proposed DRA method is shown in Figure 3.2.

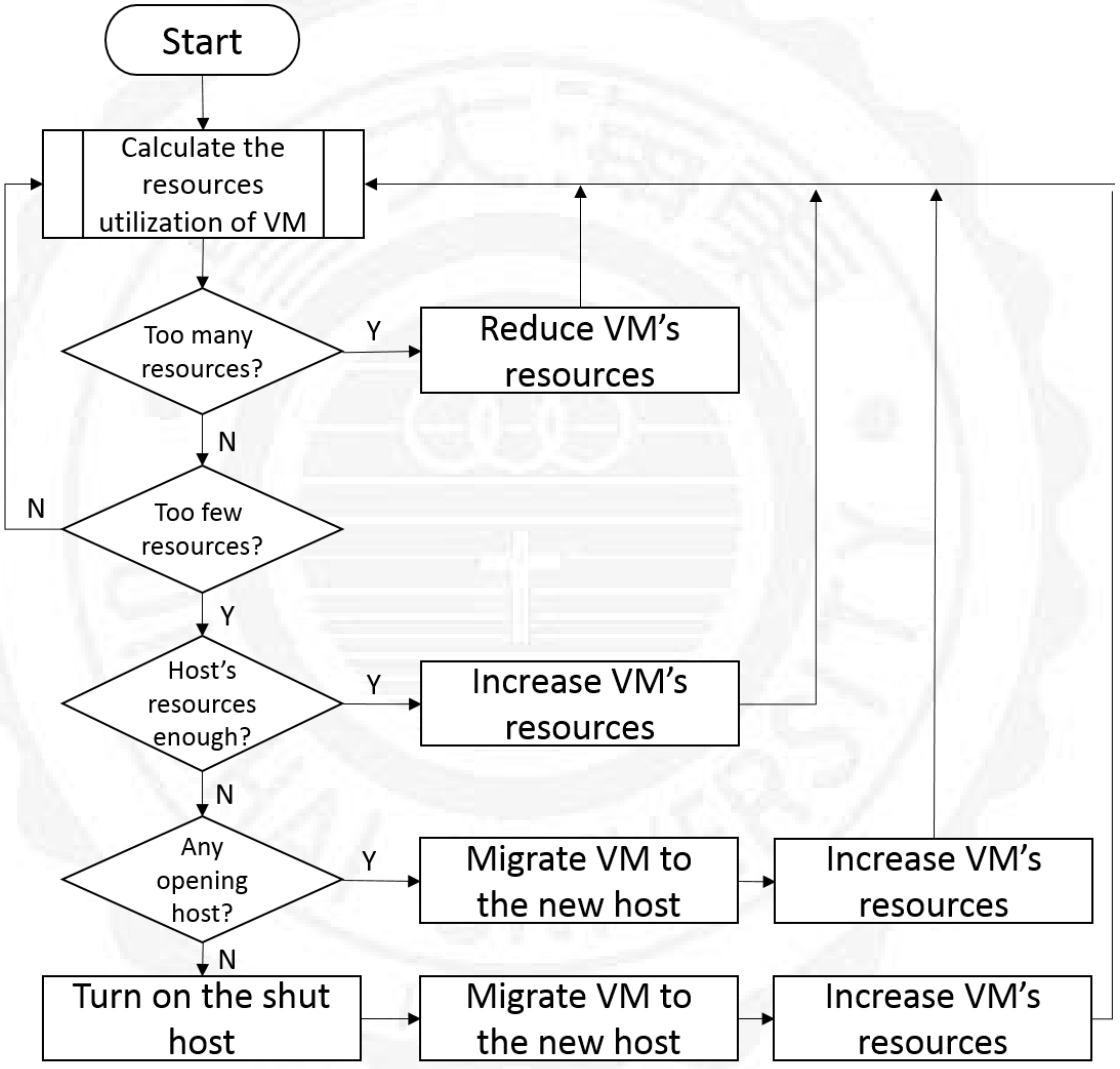


FIGURE 3.2: DRA method flow chart

3.2.2 Design Flow of Energy Saving method

For the proposed energy saving method shown in Figure 3.3, we first calculate idle resources of a host. If there are excessive idle resources, we will first check whether the idle resources are more than that of all VMs on the other host. If yes, we will move VMs from the host with fewer VMs to the other host with more VMs, and then shut down the host with VMs removed.

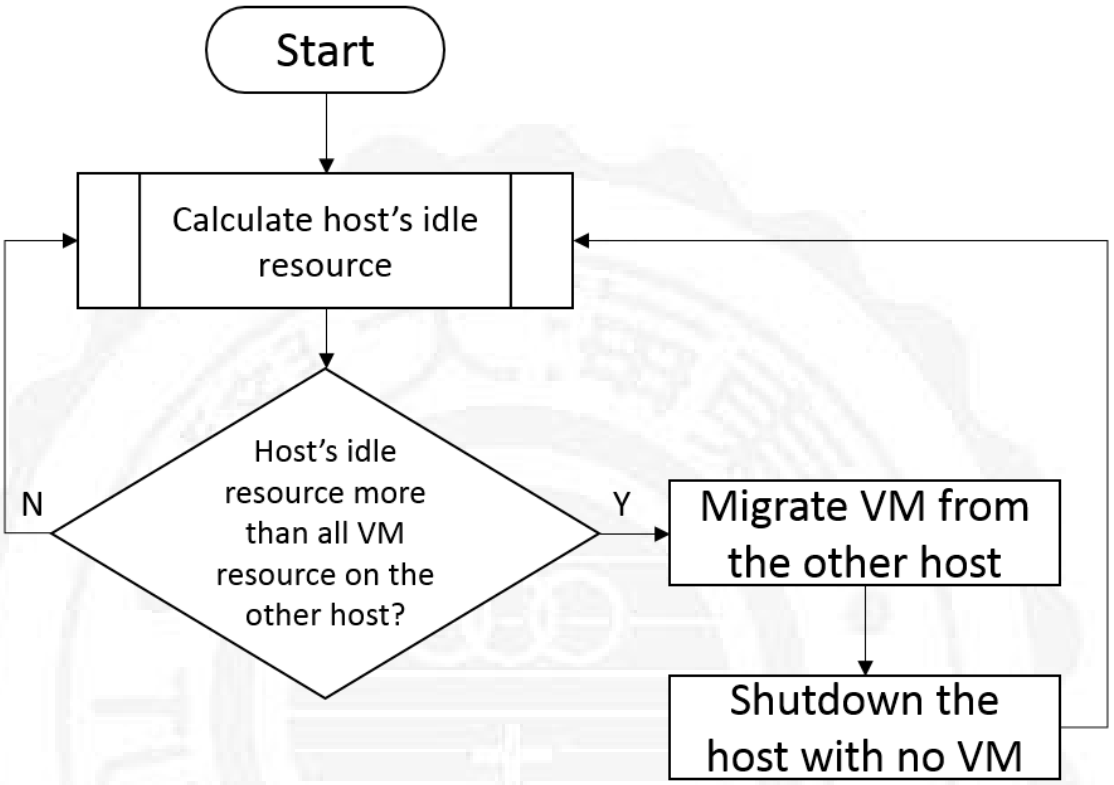


FIGURE 3.3: Energy Saving method flow chart

3.3 System Implementation

In this section, we describe how the proposed system including the improved DRA method, improved energy saving method, user interface, and status monitoring is implemented. Python language and PHP are adopted to develop programs, including the state monitoring program which monitors states of the VMs and the physical machines, the energy consumption record program which monitors the power consumption, the DRA method program which resizes VMs, and the energy saving method program which merges VMs to some host. The following provides a detailed description of the four programs.

3.3.1 Status Monitoring

We monitored the states, i.e., CPU utilization and memory utilization, of physical machines and VMs via status monitoring program. Based on monitoring data, we can observe whether the resources on each VM are excessive or inadequate. We used python system and process utilities (psutil). which is a cross-platform library for retrieving information on running processes and system utilization of CPU, memory, disks, and networking in Python. It is useful mainly for system monitoring, profiling and limiting process resources and management of running processes. The status monitoring function was developed with the Python programming language to capture status and post data to receiving program. We then used the receiving program which was developed with the PHP language to receive all monitoring data and insert these monitoring data into database. The flow of status monitoring is shown in Figure 3.4.

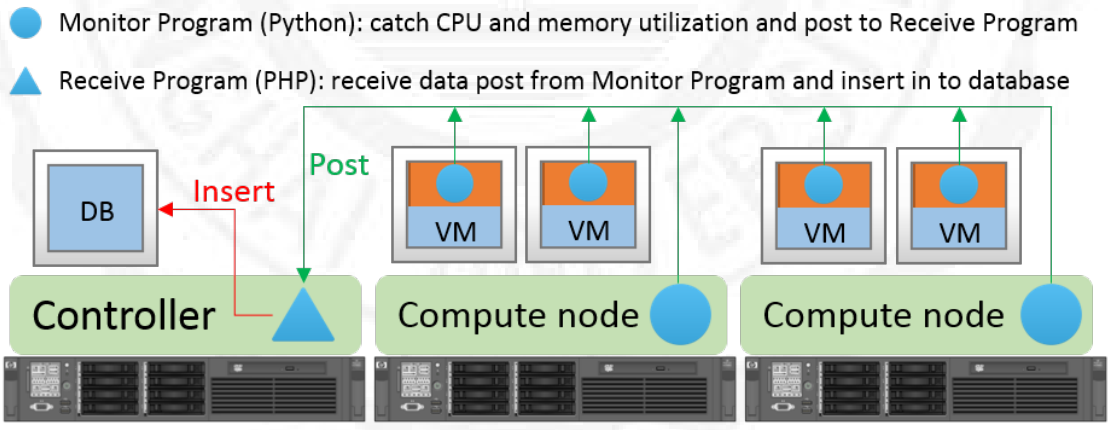


FIGURE 3.4: The Circuit of status monitoring

3.3.2 Energy Consumption Recording

We captured the energy consumption of compute nodes via a PDU, and used Simple Network Management Protocol (SNMP) to acquire energy consumption data of the PDU. The energy consumption recording function, developed by PHP language, is an automatic recording program. The flow of energy consumption recording is shown in Figure 3.5.

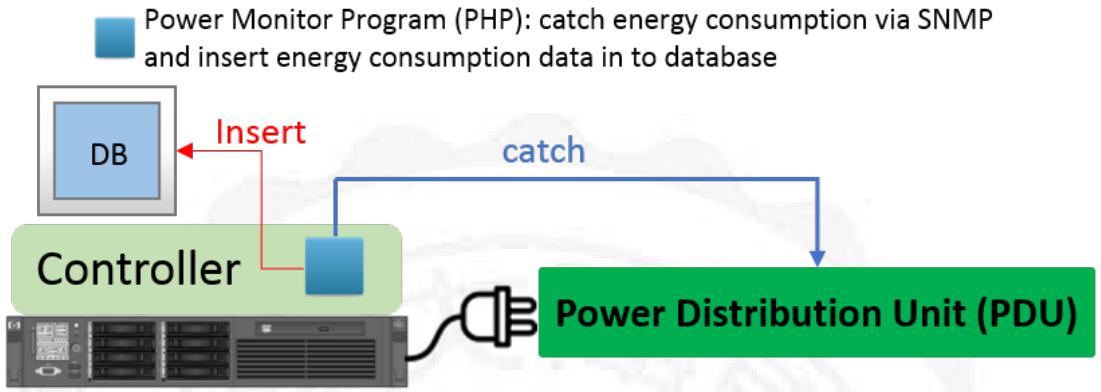


FIGURE 3.5: The Circuit of Energy Consumption Recording

Through the energy consumption recording program, we can automatically collect the energy consumption data of compute nodes from a PDU. Figure 3.6 is the web interface provided by the PDU used in this work. Figure 3.7 shows the detailed information from its outlet.

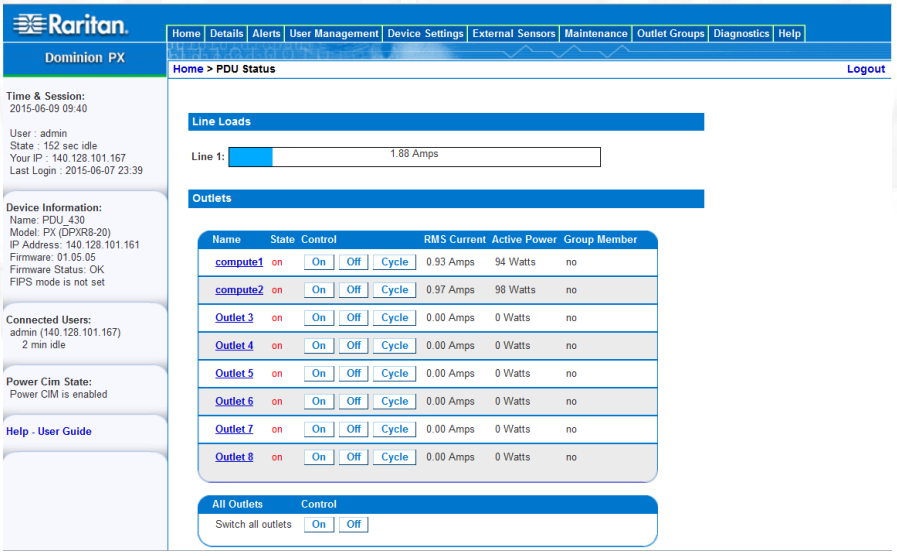


FIGURE 3.6: PDU's web interface

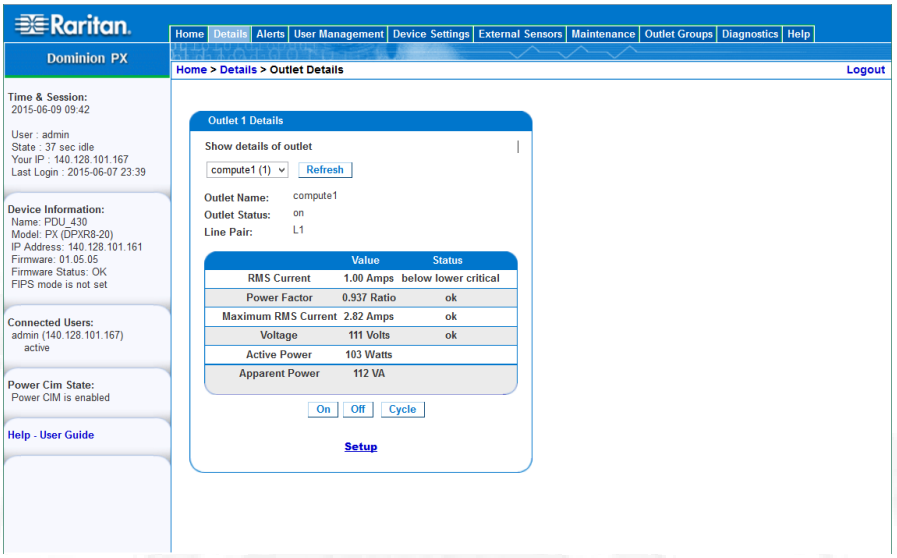


FIGURE 3.7: The detailed information of the outlet

3.3.3 DRA method

The main purpose of the DRA method, as shown in Algorithm 3.3.1, is to reduce the idle resources on VM. It can also increase the resources to the VM with inadequate resources. By the DRA method, we set the upper limit and lower limit of resource utilization for VMs. When the resource utilization of a VM exceeds the upper limit, we will increase the resources of the VM so that more resources can be used. When the resource utilization of a VM is less than the lower limit, we will reduce the resources of the VM to release its idle resources. If the resource utilization is between the upper limit and lower limit, we do not change the resources of

the VM.

Algorithm 3.3.1: DRA METHOD()

---

if (*VM resource utilization*) > *upper limit*

then
- *chech the resource of host*
- **if** (*host resource enough*)
  - **then** *resize VM bigger*
  - **else**
    - **then**
      - *turn on another host*
      - *Live migration VM to another host*
      - *then resize VM bigger*

else if (*VM resource utilization*) < *lower limit*

then
- *resize VM smaller*

else *undo*

When the resource utilization of a VM exceeds the upper limit, the performance of the VM will significantly decrease. We used CPU Limit and HPL to decide the percentage of CPU utilization will cause performance of VM obviously decrease. We respectively tested the limit of CPU utilization of 100%,90%,80%,70%,60%, 50%,40%,30%,20% and 10%, and then executed the HPL. The larger the score generated by HPL is, the better the performance is. The limit of CPU utilization can be seen as the remaining CPU utilization. For example, if the limit of CPU utilization is set to 100%, there is remaining 100% CPU utilization to execute HPL. If the limit CPU utilization is set to 60%, there is remaining 60% CPU utilization to execute HPL. HPL is used to quantify CPU performance. So, the score generated by HPL can be seen as the performance of a service. The difference between the two HPL scores indicates how the performance alters between two CPU utilizations. We will resize a VM's resource to prevent performance of the VM from dropping.

When a VM's average resource utilization is low, the VM's CPU resource is resized to a half. After resizing, the VM's average CPU utilization cannot exceed the upper limit, or the performance will be obviously dropped. We set the lower CPU utilization limit, of which the HPL score is smaller than the half of the HPL score of the upper CPU utilization limit. The average CPU utilization of a VM should not exceed the lower limit and cause performance dropped after resizing.

3.3.4 Energy Saving method

The energy saving method is mainly to save energy consumption by turning off physical machines. If the resources utilization is too high, we can merge the VMs to the other host via live migration and then shut down the host with no VMs to save energy. We define the parameters in the energy saving method as follows:

U_c is defined as number of used vCPU on all Host
U_m is defined as number of used memory on all Host
N_{Avm} is defined as number of instance on HostA
N_{Bvm} is defined as number of instance on HostB

In this method, we will first check whether the number of used vCPU and the number of used memory on all hosts exceed the limit resources of one compute node (32 vCPU and 62 GB). If the limit resources of one compute node have not been exceeded, we will check the number of instances on Host A and Host B. If the number of instances of Host B is smaller than that of Host A, we will perform live migration to migrate all VMs on Host B to Host A, and then shut down Host B. Likewise, if the number of instances of Host A is smaller than that of Host B, we will perform live migration to migrate all VMs on Host A to Host B, and then shut down Host A. If the number of used vCPU and the number of used memory on all Host exceed the limit resources of one compute node (32 vCPU and 62 GB), we cannot merge all VMs to the same compute node, since the resources of one

compute node is not enough to allocate to all VMs. Consequently, nothing will be done.

Algorithm 3.3.2: ENERGY SAVING METHOD()

---

if U_c $<=$ 32 *and* U_m $<=$ 62 GB

then
- **if** N_{Bvm} $<=$ N_{Avm} *and* N_{Bvm} $!= 0$
  - **then**
    - *Live migration all VM on HostB to HostA*
    - *shutdown HostB*
- **else** N_{Avm} $<=$ N_{Bvm} *and* N_{Avm} $!= 0$
  - **then**
    - *Live migration all VM on HostA to HostB*
    - *shutdown HostA*

else *undo*

3.4 User Interface

In the beginning of the cloud system login screen, as shown in Figure 3.8, it asks the user to enter the user name and password provided by the cloud infrastructure manager.

FIGURE 3.8: Login screen

After login, a window shows project and admin in the left column. Only the account with administrative privileges can use the admin function. We can see all instances in admin, as shown in Figure 3.9. There are many functions such as: instances' project name (the name of the host in which the instances are located), image name, IP address, the specification of instances, status, and uptime.

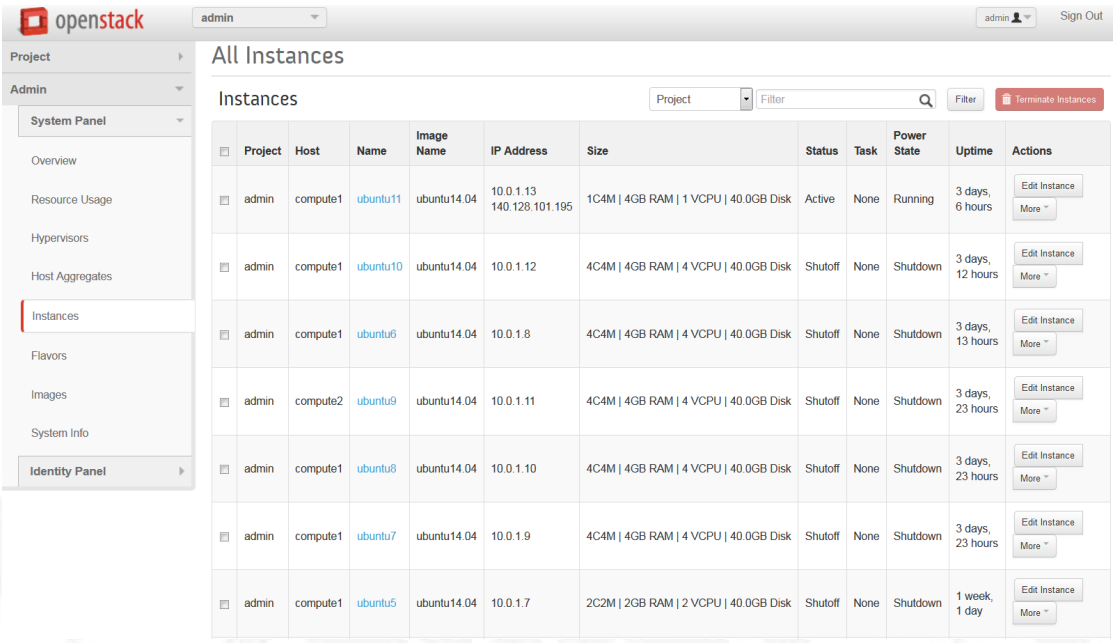


FIGURE 3.9: Detail of Instances

We can see various information of the host from the hypervisor in admin, as shown in Figure 3.10, including total vCPUs, used vCPUs, total memory, used memory, total storage, used storage, and the number of instances.

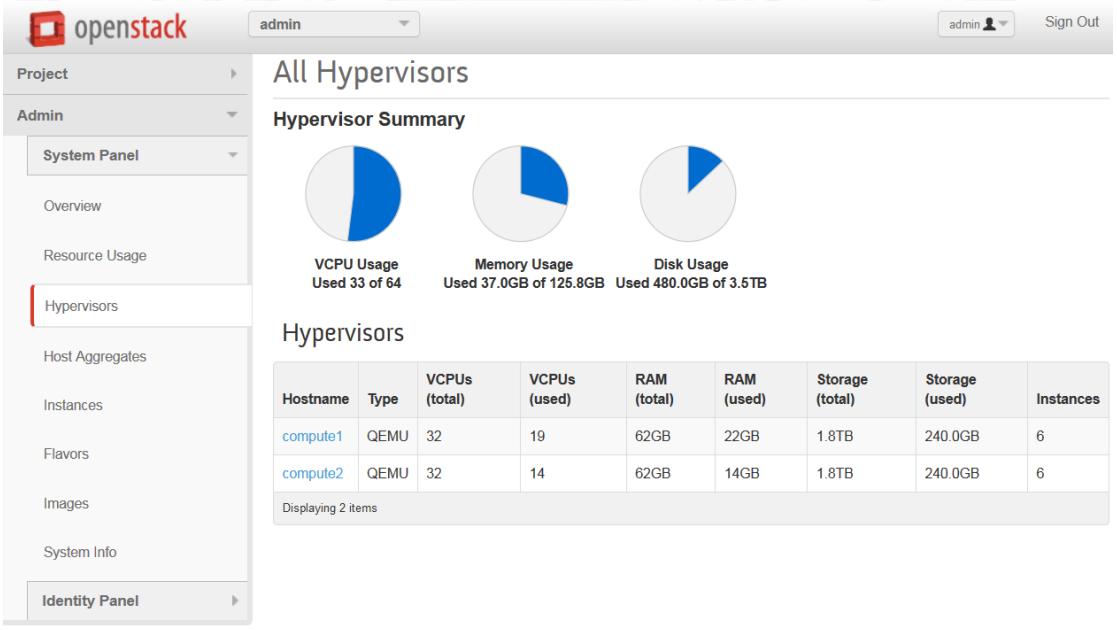


FIGURE 3.10: Detail of Hypervisors

Chapter 4

Experimental Results

In this chapter, we show the experimental environment and the experimental results. In section 4.1, we describe our experimental environment including hardware specification and software specification. And we will show our experimental results in section 4.2. We first test the performance of VM and then carry out the proposed algorithms for improving both present DRA and energy saving methods. Finally, we test the two algorithms implemented in the same system and show the experimental results.

4.1 Experimental Environment

The experimental environment consists of three computers and their hardware specifications are listed in Table 4.1. The state monitoring of the VM and the physical machine program, the energy consumption recording program, the DRA method program, the energy saving method program, the shared storage – NFS server and the user interface are built on the controller node, which consists of 12-core CPU, 30 GB memory, 2 TB disk and with Ubuntu 14.04 as the operating system. The hardware specification of each computing node is the same: 32-core CPU, 64 GB memory, 2 TB disk and with Ubuntu 14.04 as the operating system.

TABLE 4.1: Hardware specification

Host Name	CPU	RAM	HDD	OS
Controller node	Intel(R) Core(TM) i7 CPU X 990	30GB	2TB	Ubuntu14.04
Computing node1	AMD Opteron(TM) Processor 6274	64GB	2TB	Ubuntu14.04
Computing node2	AMD Opteron(TM) Processor 6274	64GB	2TB	Ubuntu14.04

Software specifications are listed in Table 4.2. The OpenStack version is Icehouse released on 17 April 2014. The Docker version is 1.3.2. The PHP version is 6.3.10. The SNMP version is 5.4.3.

TABLE 4.2: Software specification

Software	OpenStack	Python	PHP	SNMP	NFS
Version	Icehouse	2.7.6	5.5.9	5.7.2	4

4.2 Experimental Results and Discussion

4.2.1 Experiment of VM Performance

In the VM performance experiment, we tried to find at what setting of the VM' s vCPU utilization, the system performance is obviously lowest. In this experiment, we used the CPULimit kit to limit the utilization of CPU, and then executed the High-Performance Linpack (HPL). HPL has characteristic of a distribution system and use MPI to compute some data and finally it will produce a score. We can compare the performance of systems according to the score. HPL is used to quantify CPU performance. So, the score generated by HPL can be viewed as the performance of a service. The larger the score is generated by HPL, the better the performance is. We respectively tested the limit of CPU utilization of 100%、90%、80%、70%、60%、50%、40%、30%、20% and 10%, and then executed the HPL program.

In this experiment we tested the VM with 1 virtual CPU, 4 GB memory and 40 GB hard disk space. The experimental result of VM performance is shown in Figure 4.1. When we limited the utilization of CPU to 100%, i.e., no limit is set of the CPU utilization, and then executed HPL, we got a score of 15.56 Gflops, meaning 15.56 billion floating-point operations per second. When we limited the utilization of CPU to 90%, i.e., 90% of CPU utilization is used to execute HPL, We got a score of 13.74 Gflops, meaning 13.74 billion floating-point operations per second.

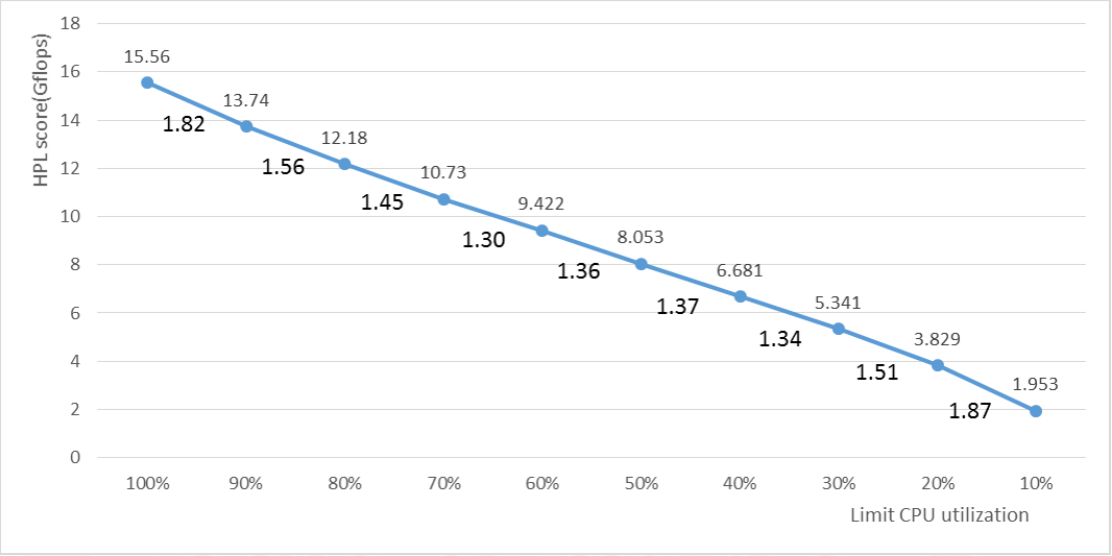


FIGURE 4.1: CPU utilization and HPL score plot

In the other words, the limit of CPU utilization can be seen as the remaining CPU utilization. For example, while the limit of CPU utilization is set to 100%, it means remaining 100% CPU utilization to execute HPL. If the limit of CPU utilization is set to 60%, it means remaining 60% CPU utilization to execute HPL.

The difference between two HPL scores increases, the performance increases faster. When the CPU utilization is 10%, the performance is 1.953 Gflops. The performance is obviously dropped. We set the upper limit for Algorithm 3.3.1 to 80%. When the VM's average CPU utilization reaches 80%, we will allocate more resources to the VM.

If a VM's average resource utilization is not high, and we will reduce resource allocated to the VM to release the idle resources. We observed that VM's performance will be optimal when the CPU utilization is about 80% with a HPL score of 12.18 Gflops. If we want to release the resource of a VM, we may reduce that VM's CPU resource to half. The half of HPL score of 80% CPU utilization is 6.09. When the limit of CPU utilization is 30%, its HPL score is 5.34, so we have to make sure that the average VM's CPU utilization is less than or equal to 30% before reducing the size. Then the CPU utilization will not exceed 80% by reducing resource of the VM. Based on results of this experiment, we will set the lower limit for Algorithm 3.3.1 to 30%. When the VM's average CPU utilization is less than or equal to 30%, we will resize VM's CPU resource to half.

4.2.2 Experiment of DRA method

The purpose of the DRA method is to release some idle resource of VMs. In this work, we created 7 VMs, and used different size of resources. VMs are created at computeA and computeB. The details of VMs are shown in Table 4.3.

Table 4.3: Details of VMs before the DRA method

Host	Name	IP Address	Size of Resource
ComputeA	Ubuntu1	10.0.1.2	8GB RAM | 8 VCPU | 40.0GB Disk
ComputeB	Ubuntu2	10.0.1.3	1GB RAM | 1 VCPU | 40.0GB Disk
ComputeA	Ubuntu3	10.0.1.4	8GB RAM | 8 VCPU | 40.0GB Disk
ComputeA	Ubuntu4	10.0.1.5	4GB RAM | 4 VCPU | 40.0GB Disk
ComputeB	Ubuntu5	10.0.1.6	1GB RAM | 1 VCPU | 40.0GB Disk
ComputeA	Ubuntu6	10.0.1.7	4GB RAM | 4 VCPU | 40.0GB Disk
ComputeA	Ubuntu7	10.0.1.8	8GB RAM | 8 VCPU | 40.0GB Disk

There are four resource sizes as listed in Table 4.4.

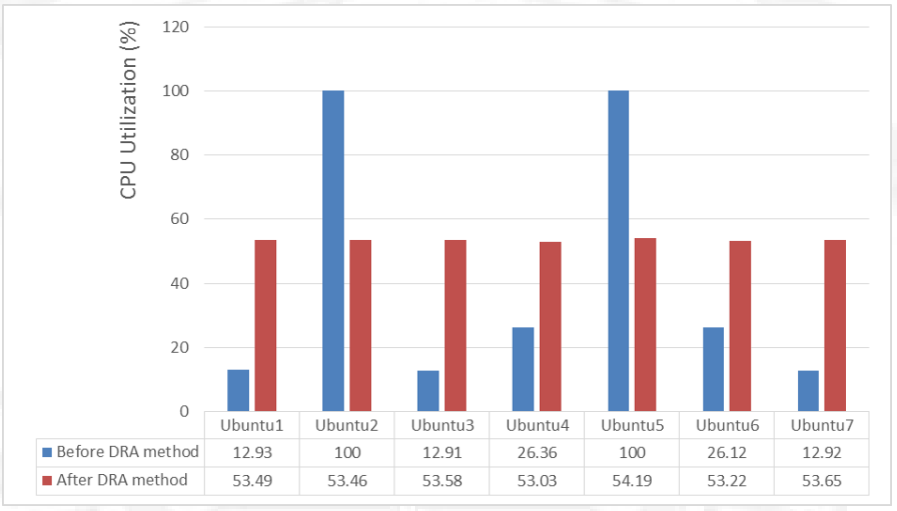
TABLE 4.4: Detail of resource size

Name	VCPUs	RAM	DISK
1C1M	1	1024MB	40GB
2C2M	2	2048MB	40GB
4C4M	4	4096MB	40GB
8C8M	8	8192MB	40GB

We monitored CPU utilization and power consumption of computeA and computeB, and also monitored CPU utilization of all VMs in an hour before applying the DRA method and an hour after applying it. Before executing the DRA method, there are 5 VMs running on computeA and 2 VMs on computeB. All VMs are running an infinite loop program to increase their CPU utilization. From the experimental results of VM performance, performance of a VM increases when the VM' s CPU utilization is close to 80%; whereas almost half of resources are idle when VM' s CPU utilization is smaller than 30%. By applying the DRA method, the average of monitoring data during an hour is computed to determine whether VM CPU utilization is exceed 80% or less than 30%. We will assign more resources to a VM if its CPU utilization exceeds 80% and reduce resources to a VM if its CPU utilization is less than 30%.

We calculate the one hour average CPU utilization of VMs after applying the DRA method and compare it with the one hour average CPU utilization of VMs before applying the DRA method, as shown in Figure 4.2. Before executing the DRA method, the CPU utilization of Ubuntu1, Ubuntu3, Ubuntu4, Ubuntu6 and Ubuntu7 are less than 30%, so we reduce their resources; and the CPU utilization of Ubuntu2 and Ubuntu5 are more than 80%, so we increase their resources. Before applying the DRA method, 34 VCPUs and 34 GB memory are used in the VMs. After executing the DRA method, Ubuntu1, Ubuntu3, Ubuntu4, Ubuntu6 and Ubuntu7 release the idle resources. We resize Ubuntu1, Ubuntu3 and Ubuntu7 from 8 VCPUs and 8 GB memory to 2 VCPUs and 2 GB memory; Ubuntu4 and

Ubuntu6 from 4 VCPUs and 4 GB memory to 2 VCPUs and 2 GB memory. And after executing the DRA method, Ubuntu2 and Ubuntu5 have more resource: we resize Ubuntu2 and Ubuntu5 from 1 VCPU and 1 GB memory to 2 VCPUs and 2 GB memory. After applying the DRA method, 14 VCPUs and 14 GB memory are used in the VMs. Thus, 20 VCPUs and 20 GB memory in total are released.



	Ubuntu1	Ubuntu2	Ubuntu3	Ubuntu4	Ubuntu5	Ubuntu6	Ubuntu7
Before DRA method	12.93	100	12.91	26.36	100	26.12	12.92
After DRA method	53.49	53.46	53.58	53.03	54.19	53.22	53.65

FIGURE 4.2: VM's CPU utilization before DRA method and after DRA method

The average monitoring data of power consumption and CPU utilization at compute nodes are shown in Figure 4.3 and Figure 4.4, respectively. Before applying the DRA method, the VMs need to resize smaller are all created on ComputeA, and VM need to resize bigger on ComputeB. After executing the DRA method, power consumption and CPU utilization of ComputeA and ComputeB slightly increase. However, ComputeA and ComputeB release 20 VCPUs and 20 GB memory in total.

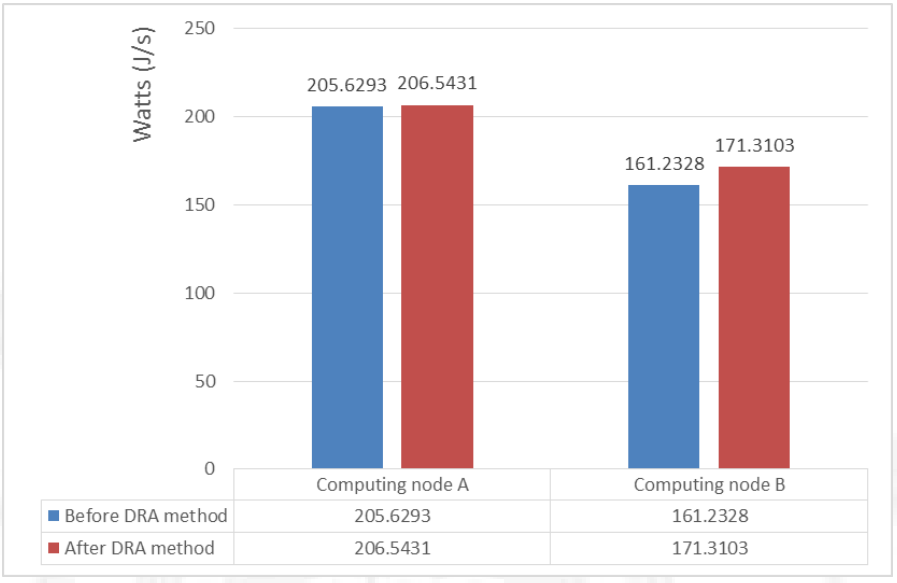


FIGURE 4.3: Compute node power consumption before DRA method and after DRA method

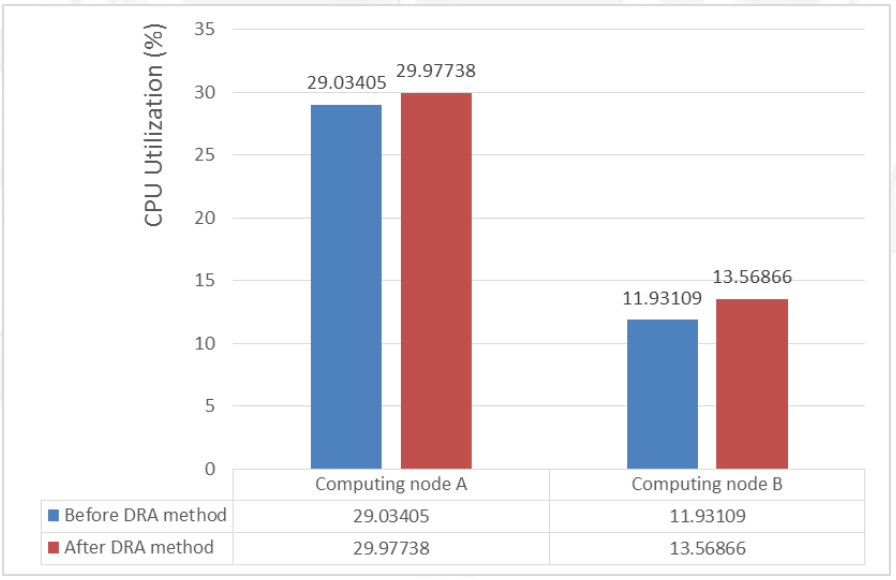


FIGURE 4.4: Compute node CPU utilization before DRA method and after DRA method

4.2.3 Experiment of Energy Saving method

The purpose of the energy saving method is to find which compute node has fewer VMs, and use live migration to merge VMs of this compute node to another compute node, and then shut down the compute node with VMs removed to save the power consumption. In the experiment of the energy saving method we

created 7 VMs, and used different resource sizes. VMs were created at computeA and computeB. The details of VM are shown in Table 4.3.

As shown in Figure 4.5, before executing the energy saving method, there were 5 VMs on computeA and 2 VMs on comptueB. The energy saving method works by first checking whether all used resource of VMs are located at more than one compute node. It found that the resources of VMs are located at more than one compute node, so it would not merge VMs to the same compute node. After the energy saving method, there were still 5 VMs on computeA and 2 VMs on comptueB.

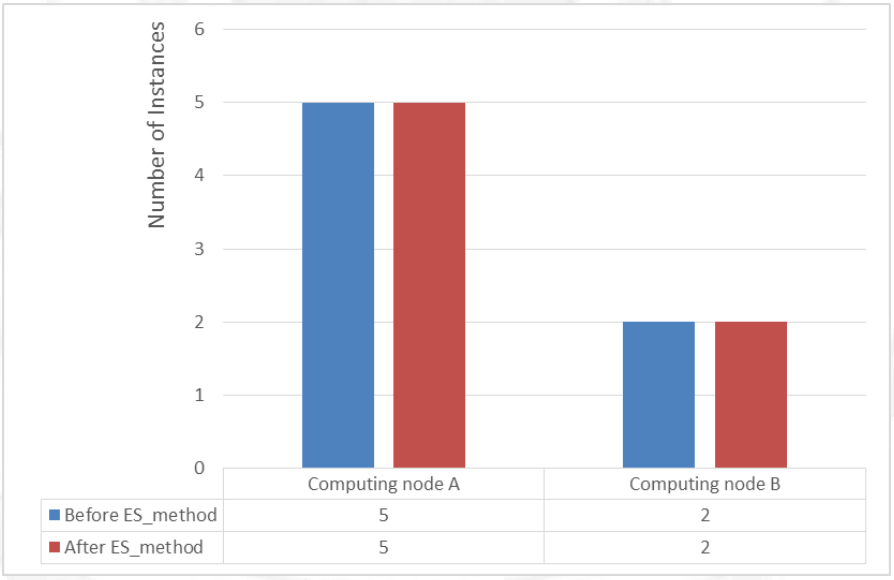


FIGURE 4.5: Number of instance before ES method and after ES method

We monitored CPU utilization and power consumption of computeA and computeB for an hour before applying the energy saving method and an hour after applying it. The average of monitoring data of power consumption and CPU utilization are shown in Figure 4.6 and Figure 4.7, respectively. The CPU utilization and power consumption of ComputeA and ComputeB are almost not changed. After executing the energy saving method, no live migration was performed on VMs, because the maximum resource of either compute node is fewer than all used resource of VMs.

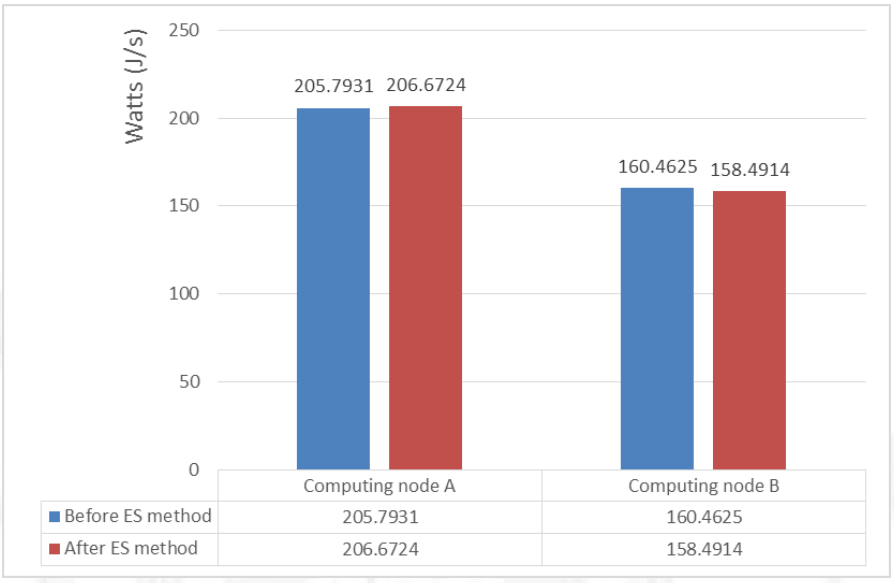


FIGURE 4.6: Compute node power consumption before ES method and after ES method

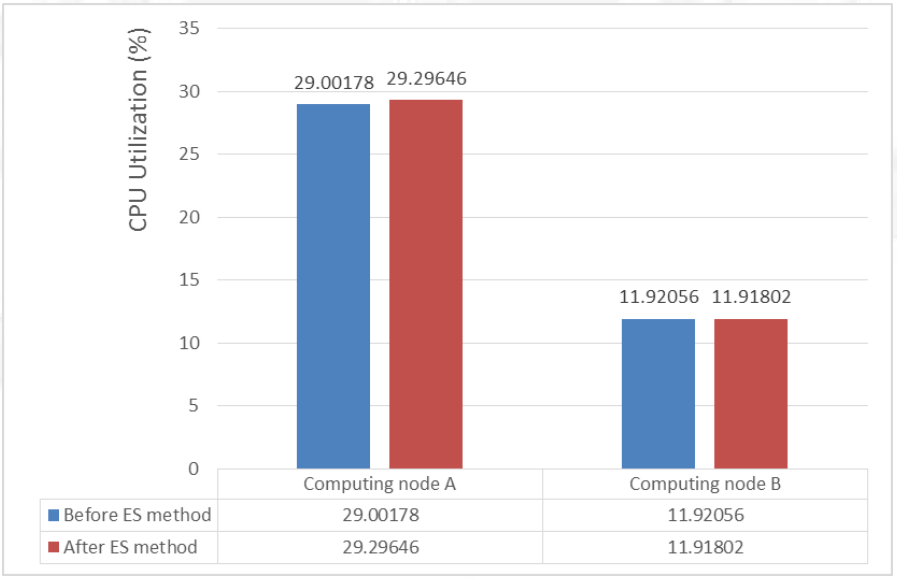


FIGURE 4.7: Compute node CPU utilization before ES method and after ES method

4.2.4 Experiment of DRA and Energy Saving method

In this experiment the DRA method is used to release idle resource of VMs; after that, the compute node will have more available resources to use the energy saving method to merge VMs to a compute node and shut down the compute node

without VMs. After applying the DRA and energy saving methods, the details of VMs are shown in Table 4.5.

TABLE 4.5: Details of VMs after applying the DRA and energy saving method

Host	Name	IP Address	Size of Resource
ComputeA	Ubuntu1	10.0.1.2	2GB RAM | 2 VCPU | 40.0GB Disk
ComputeA	Ubuntu2	10.0.1.3	2GB RAM | 2 VCPU | 40.0GB Disk
ComputeA	Ubuntu3	10.0.1.4	2GB RAM | 2 VCPU | 40.0GB Disk
ComputeA	Ubuntu4	10.0.1.5	2GB RAM | 2 VCPU | 40.0GB Disk
ComputeA	Ubuntu5	10.0.1.6	2GB RAM | 2 VCPU | 40.0GB Disk
ComputeA	Ubuntu6	10.0.1.7	2GB RAM | 2 VCPU | 40.0GB Disk
ComputeA	Ubuntu7	10.0.1.8	2GB RAM | 2 VCPU | 40.0GB Disk

By the DRA method, Ubuntu1, Ubuntu3, Ubuntu4, Ubuntu6 and Ubuntu 7 released the idle resources: Ubuntu1, Ubuntu3 and Ubuntu7 were resized from 8 VCPUs and 8 GB memory to 2VCPUs and 2GB memory; Ubuntu4 and Ubuntu6 were resized from 4 VCPUs and 4 GB memory to 2VCPUs and 2GB memory. Besides, Ubuntu2 and Ubuntu5 were allocated with more resources to use: Ubuntu2 and Ubuntu 5 were resized from 1VCPU and 1GB memory to 2VCPUs and 2GM memory. In total, 14 VCPUs and 14 GB memory were used in the VMs; in other words, 20 VCPUs and 20 GB memory were released. The CPU utilization of VMs are shown in Figure 4.8.

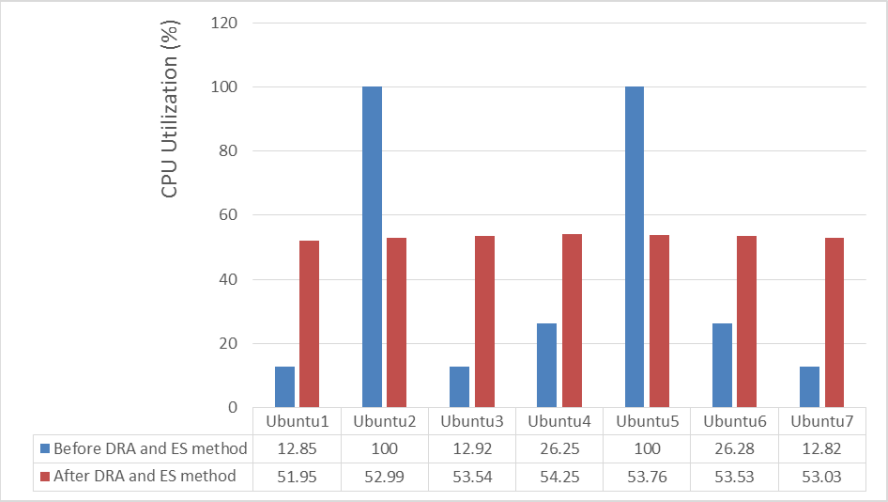


FIGURE 4.8: VM CPU utilization compare graph

As shown in Figure 4.9, after applying the DRA and energy saving methods, all VMs on compute were moved to computeA by live migration. After performing live migration, we turned off the comptueB with VMs removed.

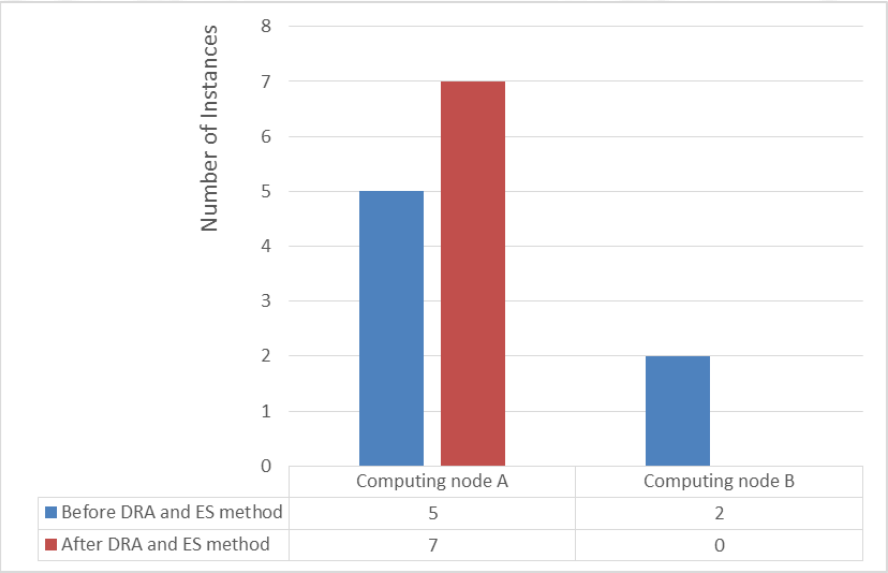


FIGURE 4.9: Number of instance compare graph

Before applying the DRA and energy saving methods, there were VCPUs and 32 GB memory used on conputeA, 2 VCPUs and 2 GB memory used on computeB. After executing the DRA and energy saving methods, only 14 VCPUs and 14 GB memory were used on computeA. The DRA and energy saving methods increased

the CPU utilization of compute, but it also decreased CPU utilization of computeB to zero, as shown in Figure 4.10.

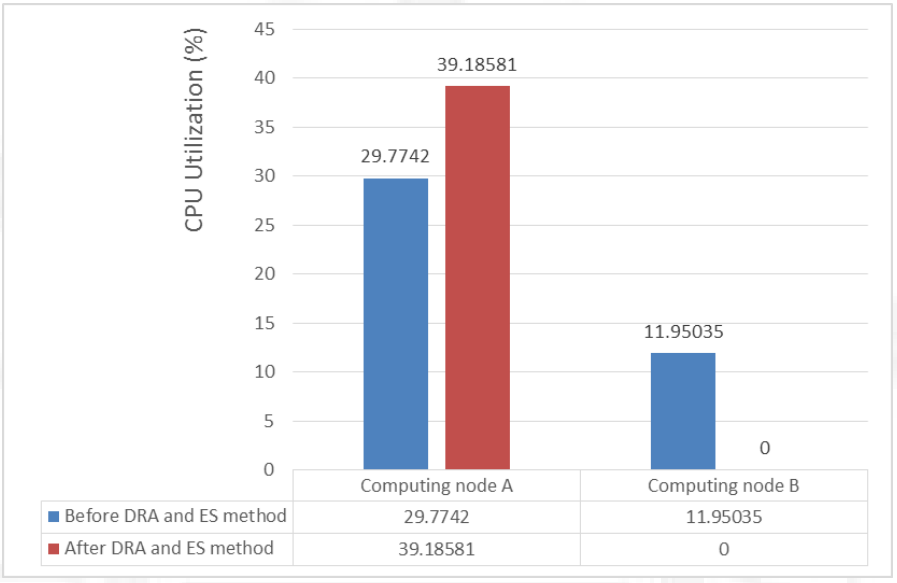


FIGURE 4.10: Compute node CPU utilization compare graph

As shown in Figure 4.11, before applying the DRA and energy saving methods, the total power consumption of the system was 363.5517 watts. The power consumption of computeA increased from 204.3103 watts to 218.5 watts, but the power consumption if comptueB decreased from 159.2414 watts to zero. Even though the power consumption of computeA increased, the system saved 145.0517 Watts overall. In other words, a total of 39.89% power consumption was saved.

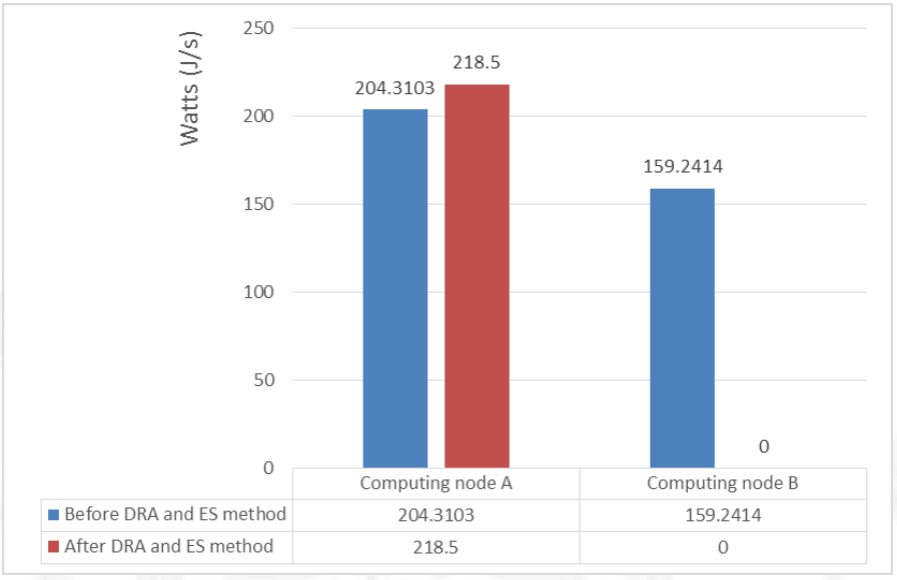


FIGURE 4.11: Compute node power consumption compare graph

Through executing the DRA and energy saving methods, it not only saved 39.89% power consumption in total, but also released 20 VCPUs and 20 GB memory.

4.2.5 Discussion

The authors in [6] showed that live migration of VMs does not cause significant additional electricity costs and their power-saving method is effective, achieving from 7% to 14% in their design. The authors in [30] showed how to use conditions to choose the migrating VMs: when the host meets with the high condition, it will start load-balance. Their method chooses the VMs conforming to the condition that less memory and more CPU resource are used, and then do the load balance on the selected VMs; less time will be spent and less host resources will be used. They showed that about 15% of accumulated power consumption are saved by using their power saving method. In both papers KVM and libvirt are used to build the cloud cluster.

Different from above two papers, we used the cloud software OpenStack to build a cloud system with the energy saving method and the DRA Method on VMs. Through our energy saving method, we first determined which PM has fewer

VMs, and then perform live migration to remove all VMs on this PM to other PM. Because the live migration will cause power consumption, so we choose the PM with fewer VMs to merge to other PM. In our system with the DRA method and the energy saving method, we not only saved about 39% power consumption, but also released the idle resources of VMs. Before executing the energy saving method, we must execute the DRA method to adjust resources on VMs by either releasing idle resources of VMs or allocating more resource to some VMs. The utilization of VMs fluctuates and will not always keep high or low. Therefore, we repeatedly execute the DRA and energy saving methods at a regular period of time to ensure that all VMs do not have too much idle resources or too few resource.

Chapter 5

Conclusions and Future Work

In this thesis, we built an energy saving cloud system which implements VM DRA to achieve our goal: reduce the idled resources on the VMs and reduce the energy consumption of the entire cloud cluster. We will describe our conclusion in section 5.1 and future work in section 5.2.

5.1 Concluding Remarks

In this work, we implement an infrastructure platform base on cloud software – OpenStack. We achieve our goal by using a DRA method to reduce the idled resources on the VM and an energy saving method to save the energy consumption of servers. The DRA method not only can reduce the waste of the idled resources on VMs, but also can increase allocation of resources to any VM with inadequate resources. The energy saving method works mainly through shutting idle physical machines to save energy consumption.

Through the DRA method we can find the most suitable resource capacity for the VMs. After running a period of time, we will use the average CPU utilization of this period of time to alter allocation of the resources of VMs to achieve the suitable resource capacity. After applying the DRA method, we execute the energy saving method. The energy saving method can merge the VMs to some compute

node and shut down the compute node with all VMs moved to achieve the energy saving purpose. In this thesis, we achieve our goal to release the idled resources on the VMs and allocate more resource to the VMs in need of resources. The experimental results show that 39.89% of power is saved and 20 VCPUs and 20 GB memory are released by the DRA combined with the energy saving method.

5.2 Future work

The DRA method and the energy saving method still have some work to do. For the DRA method, we plan to combine OpenStack with Docker to achieve less down time when resizing VMs. For the energy saving method, we will consider a few more factors, such as power consumption in resizing VMs. In the future work, we will not only continue studying the DRA method and the energy saving method to improve our system, but also apply our system to a larger environment to verify its performance.

References

[1] Gartner identifies the top 10 strategic technology trends for 2015, 2014. `http://www.gartner.com/newsroom/id/2867917`.

[2] Andreas Wolke, Boldbaatar Tsend-Ayush, Carl Pfeiffer, and Martin Bichler. More than bin packing: Dynamic resource allocation strategies in cloud data centers. *Information Systems*, pages 83 – 95, 2015.

[3] Lucas W. Krakow, Louis Rabiet, Yun Zou, Guillaume Iooss, and Sanjay Rajopadhye Edwin K.P. Chong. Optimizing dynamic resource allocation. *Procedia Computer Science*, pages 1277 – 1288, 2014.

[4] M.B. Nagpure, P. Dahiwale, and P Marbate. An efficient dynamic resource allocation strategy for vm environment in cloud. *Pervasive Computing (ICPC), 2015 International Conference on*, pages 1 – 5, 2015.

[5] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-aware resource allocation heuristics for efficient management of data centers for cloud computing. *Future Generation Computer Systems*, 28:755 – 768, 2012.

[6] Chao-Tung Yang, Kuan-Lung Huang, Jung-Chun Liu, Yi-Wei Su, and Chu W.C.-C. Implementation of a power saving method for virtual machine management in cloud. *Cloud Computing and Big Data (CloudCom-Asia), 2013 International Conference on*, pages 283 – 290, 2013.

[7] Ningling Wang, Peng Fu, Yongping Yang, Longfei Zhu, and Dianfa Wu. Spatial-temporal energy-saving effect for the diagnosis of energy-consumption

benchmark state of thermal power units. *Energy Procedia*, pages 1848 – 1851, 2014.

[8] G. Thomas, K. Chandrasekar, B. Akesson, and B. Juurlink. A predictor-based power-saving policy for dram memories. *Digital System Design (DSD), 2012 15th Euromicro Conference on*, pages 882 – 889, 2012.

[9] Rajkumar Buyya, Christian Vecchiola, and S. Thamarai Selvi. *Mastering Cloud Computing: Chapter 3 – Virtualization.* MORGAN KAUFMANN, 2013.

[10] Xiaofei Liao, Hai Jin, Shizhan Yu, and Yu Zhang. A novel memory allocation scheme for memory energy reduction in virtualization environment. *Journal of Computer and System Sciences*, 81:3 – 15, 2015.

[11] Yaozu Dong, Xiantao Zhang, Jinquan Dai, and Haibing Guan. Hyvi: A hybrid virtualization solution balancing performance and manageability. *Parallel and Distributed Systems*, 25:2332 – 2341, 2014.

[12] S.A. Babu, M.J. Hareesh, J.P. Martin, S. Cherian, and Y. Sastri. System performance evaluation of para virtualization, container virtualization, and full virtualization using xen, openvz, and xenserver. In *Advances in Computing and Communications (ICACC), 2014 Fourth International Conference on*, pages 247–250, Aug 2014.

[13] Inhyuk Kim, Taehyoung Kim, and Young Ik Eom. Nhvm: Design and implementation of linux server virtual machine using hybrid virtualization technology. In *Computational Science and Its Applications (ICCSA), 2010 International Conference on*, pages 171–175, March 2010.

[14] Hypervisor, 2015. `http://en.wikipedia.org/wiki/Hypervisor`.

[15] Openstack open source cloud computing software, 2015. `http://www.openstack.org/`.

[16] What is openstack?, 2015. `http://opensource.com/resources/what-is-openstack`.

[17] Openstack, 2015. http://en.wikipedia.org/wiki/OpenStack.

[18] Zhaojun Li, Haijiang Li, Xicheng Wang, and Keqiu Li. A generic cloud platform for engineering optimization based on openstack. *Advances in Engineering Software*, 75:42 – 57, 2014.

[19] Openstack architecture, 2014. http://docs.openstack.org/icehouse/install-guide/install/apt/content/ch_overview.html#architecture_overview.

[20] Hai Jin, Li Deng, Song Wua, Xuanhua Shia, Hanhua Chena, and Xiaodong Panc. Mecom: Live migration of virtual machines by adaptively compressing memory pages. *Future Generation Computer Systems*, 38:23 – 25, 2014.

[21] Mattias Forsman, Andreas Glad, Lars Lundberg, and Dragos Ilie. Algorithms for automated live migration of virtual machines. *Journal of Systems and Software*, 101:110 – 126, 2015.

[22] Muhammad Atif and Peter Strazdins. Adaptive parallel application resource remapping through the live migration of virtual machines. *Future Generation Computer Systems*, 37:148 – 161, 2014.

[23] Hai Jin, Wei Gao, Song Wu, Xuanhua Shi, Xiaoxin Wu, and Fan Zhou. Optimizing the live migration of virtual machine by cpu scheduling. *Journal of Network and Computer Applications*, 34:1088 – 1096, 2011.

[24] Kejiang Ye, Xiaohong Jiang, Ran Ma, and Fengxi Yan. Vc-migration: Live migration of virtual clusters in the cloud. In *Grid Computing (GRID), 2012 ACM/IEEE 13th International Conference on*, pages 209–218, Sept 2012.

[25] Shaoming Guo, Wang Yang, and Guojun Wang. Nfs protocol performance analysis and improvement for mobile transparent computing. *High Performance Computing and Communications 2013 IEEE International Conference on*, pages 1878 – 1883, 2013.

[26] Network file system, 2015. http://en.wikipedia.org/wiki/Network_File_System.

[27] Alex Osadzinski. *The Network File System (NFS).* Computer Standards and Interfaces, 1988–1989.

[28] Power distribution unit, 2015. `http://en.wikipedia.org/wiki/Power_distribution_unit`.

[29] What is power distribution unit?, 2013. `http://searchdatacenter.techtarget.com/definition/power-distribution-unit-PDU`.

[30] Chao-Tung Yang, Chih-Liang Chuang, Jung-Chung Liu, Chien-Chih Chen, and W.C. Chu. Implementation of cloud infrastructure monitor platform with power saving method. In *Advanced Information Networking and Applications Workshops (WAINA), 2015 IEEE 29th International Conference on*, pages 223–228, March 2015.

[31] Cloud computing, 2015. `http://en.wikipedia.org/wiki/Cloud_computing`.

[32] What is cloud computing?, 2015. `http://www.ibm.com/cloud-computing/us/en/what-is-cloud-computing.html`.

[33] Cloud open lab, 2012. `http://www.cloudopenlab.org.tw/ccipo_industryDefinition.do`.

[34] Rajkumar Buyya, Christian Vecchiola, and S. Thamarai Selvi. *Mastering Cloud Computing: Chapter 4 – Cloud Computing Architecture.* MORGAN KAUFMANN, 2013.

Appendix A

OpenStack Installation

I. Network Time Protocol (NTP)

```
$ apt-get install ntp
```

II. Database (Controller node setup)

```
$ apt-get install python-mysqldb mysql-server
#===== MySQL configure =====
[mysqld]
...
bind-address = Your_IP
[mysqld]
...
default-storage-engine = innodb
innodb_file_per_table
collation-server = utf8_general_ci
init-connect = 'SET NAMES utf8'
character-set-server = utf8

$ service mysql restart
$ mysql_secure_installation
Set root password? [Y/n] Y
Remove anonymous users? [Y/n] Y
Disallow root login remotely? [Y/n] Y
Remove test database and access to it? [Y/n] Y
Reload privilege tables now? [Y/n] Y
```

III. Database (Compute node setup)

```
$ apt-get install python-mysqldb
```

IV. MySQL Setting

```
$ mysql -u root -p
CREATE DATABASE keystone;
GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'localhost' \
  IDENTIFIED BY 'KEYSTONE_DBPASS';
GRANT ALL PRIVILEGES ON keystone.* TO 'keystone'@'%' \
  IDENTIFIED BY 'KEYSTONE_DBPASS';
exit
$ mysql -u root -p
CREATE DATABASE glance;
GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'localhost' \
  IDENTIFIED BY 'GLANCE_DBPASS';
GRANT ALL PRIVILEGES ON glance.* TO 'glance'@'%' \
  IDENTIFIED BY 'GLANCE_DBPASS';
exit
$ mysql -u root -p
CREATE DATABASE nova;
GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'localhost' \
  IDENTIFIED BY 'NOVA_DBPASS';
GRANT ALL PRIVILEGES ON nova.* TO 'nova'@'%' \
  IDENTIFIED BY 'NOVA_DBPASS';
exit
```

V. Messaging Server

```
$ apt-get install rabbitmq-server
$ rabbitmqctl change_password guest YOUR_RABBIT_PASS
```

VI. Identity Service Install and Configure

```
$ apt-get install keystone
$ openssl rand -hex 10
#===== KeyStone configure =====
#Edit /etc/keystone/keystone.conf
[DEFAULT]
...
```

```
admin_token = ADMIN_TOKEN
...
log_dir = /var/log/keystone
[database]
# The SQLAlchemy connection string used to connect to the database
connection = mysql://keystone:KEYSTONE_DBPASS@controller/keystone
...

$ rm /var/lib/keystone/keystone.db
$ su -s /bin/sh -c "keystone-manage db_sync" keystone
$ service keystone restart

$ export OS_SERVICE_TOKEN=ADMIN_TOKEN
$ export OS_SERVICE_ENDPOINT=http://controller:35357/v2.0
$ keystone user-create --name=admin --pass=ADMIN_PASS
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
| enabled  |               True               |
|    id    | 4d411f2291f34941b30eef9bd797505a |
|   name   |              admin               |
| username |              admin               |
+----------+----------------------------------+
$ keystone role-create --name=admin
+----------+----------------------------------+
| Property |              Value               |
+----------+----------------------------------+
|    id    | cd2cb9a39e874ea69e5d4b896eb16128 |
|   name   |              admin               |
+----------+----------------------------------+
$ keystone tenant-create --name=admin --description="Admin Tenant"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |           Admin Tenant           |
|   enabled   |               True               |
|      id     | cf12a15c5ea84b019aec3dc45580896b |
|     name    |              admin               |
+-------------+----------------------------------+
$ keystone user-role-add --user=admin --tenant=admin --role=admin
$ keystone user-role-add --user=admin --role=_member_ --tenant=admin
$ keystone tenant-create --name=service --description="Service Tenant"
+-------------+----------------------------------+
|   Property  |              Value               |
+-------------+----------------------------------+
| description |           service Tenant         |
|   enabled   |               True               |
|      id     | 55cbd79c0c014c8a95534ebd16213ca1 |
```

```
|     name    |              service              |
+-------------+-----------------------------------+
$ keystone service-create --name=keystone --type=identity \
  --description="OpenStack Identity"
+-------------+-----------------------------------+
|   Property  |              Value                |
+-------------+-----------------------------------+
| description |        OpenStack Identity         |
|   enabled   |               True                |
|      id     | 15c11a23667e427e91bc31335b45f4bd  |
|     name    |             keystone              |
|     type    |             identity              |
+-------------+-----------------------------------+
$ keystone endpoint-create \
  --service-id=$(keystone service-list | awk '/ identity / {print $2}') \
  --publicurl=http://IP:5000/v2.0 \
  --internalurl=http://IP:5000/v2.0 \
  --adminurl=http://IP:35357/v2.0
+-------------+-----------------------------------+
|   Property  |              Value                |
+-------------+-----------------------------------+
|   adminurl  |   http://IP:35357/v2.0    |
|      id     | 11f9c625a3b94a3f8e66bf4e5de2679f  |
| internalurl |   http://IP:5000/v2.0     |
|  publicurl  |   http://IP:5000/v2.0     |
|    region   |           regionOne               |
|  service_id | 15c11a23667e427e91bc31335b45f4bd  |
+-------------+-----------------------------------+
$ unset OS_SERVICE_TOKEN OS_SERVICE_ENDPOINT
$ keystone --os-tenant-name admin --os-username admin --os-password ADMIN_PASS \
  --os-auth-url http://controller:35357/v2.0 token-get
+-----------+-----------------------------------+
|  Property |              Value                |
+-----------+-----------------------------------+
|  expires  |       2014-10-10T12:50:12Z        |
|     id    | 8963eb5ccd864769a894ec316ef8f7d4  |
| tenant_id | cf12a15c5ea84b019aec3dc45580896b  |
|  user_id  | 4d411f2291f34941b30eef9bd797505a  |
+-----------+-----------------------------------+
$ keystone --os-tenant-name admin --os-username admin --os-password ADMIN_PASS \
  --os-auth-url http://controller:35357/v2.0 tenant-list
+----------------------------------+----------+---------+
|                id                |   name   | enabled |
+----------------------------------+----------+---------+
| 4d411f2291f34941b30eef9bd797505a |  admin   |   True  |
| 6b69202e1bf846a4ae50d65bc4789122 | service  |   True  |
+----------------------------------+----------+---------+
$ keystone user-list
```

```
+----------------------------------+-------+---------+
|                id                |  name | enabled |
+----------------------------------+-------+---------+
| 4d411f2291f34941b30eef9bd797505a | admin |   True  |
+----------------------------------+-------+---------+
$ keystone user-role-list
+----------------------------------+----------+
|                id                |   name   |
+----------------------------------+----------+
| 9fe2ff9ee4384b1894a90878d3e92bab | _member_ |
| 5d3b60b66f1f438b80eaae41a77b5951 |  admin   |
+----------------------------------+----------+
#===== admin-openrc.sh =====
# Create admin-openrc.sh file
export OS_USERNAME=admin
export OS_PASSWORD=ADMIN_PASS
export OS_TENANT_NAME=admin
export OS_AUTH_URL=http://IP:35357/v2.0

$ source admin-openrc.sh
$ keystone token-get
```

VII. Image Service Install and Configure

```
$ apt-get install glance python-glanceclient
#===== Glance configure =====
#Edit /etc/glance/glance-api.conf and /etc/glance/glance-registry.conf
[database]
connection = mysql://glance:GLANCE_DBPASS@controller/glance
[keystone_authtoken]
auth_uri = http://controller:5000
auth_host = controller
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = glance
admin_password = GLANCE_PASS
[paste_deploy]
...
flavor = keystone

$ rm /var/lib/glance/glance.sqlite
$ su -s /bin/sh -c "glance-manage db_sync" glance
$ keystone user-create --name=glance --pass=GLANCE_PASS \
   --email=glance@example.com
+----------+----------------------------------+
```

```
| Property |              Value               |
+----------+----------------------------------+
| enabled  |               True               |
|    id    | 4dserthdf33456set2342f9bd797505a |
|   name   |              glance              |
| username |              glance              |
+----------+----------------------------------+
$ keystone user-role-add --user=glance --tenant=service --role=admin
$ keystone service-create --name=glance --type=image \
  --description="OpenStack Image Service"
+-------------+----------------------------------+
| Property    | Value                            |
+-------------+----------------------------------+
| description | OpenStack Image service          |
| enabled     | True                             |
| id          | 178124d6081c441b80d79972614149c6 |
| name        | glance                           |
| type        | image                            |
+-------------+----------------------------------+
$ keystone endpoint-create \
  --service-id=$(keystone service-list | awk '/ image / {print $2}') \
  --publicurl=http://controller:9292 \
  --internalurl=http://controller:9292 \
  --adminurl=http://controller:9292
+--------------+----------------------------------+
| Property     | Value                            |
+--------------+----------------------------------+
| adminurl     | http://controller:9292           |
| id           | 805b1dbc90ab47479111102bc6423313 |
| internalurl  | http://controller:9292           |
| publicurl    | http://controller:9292           |
| region       | RegionOne                        |
| service_id   | 178124d6081c441b80d79972614149c6 |
| service_name | glance                           |
| service_type | image                            |
+--------------+----------------------------------+
$ service glance-registry restart
$ service glance-api restart

$ mkdir /tmp/images
$ cd /tmp/images/
$ wget http://download.cirros-cloud.net/0.3.2/cirros-0.3.2-x86_64-disk.img
$ source admin-openrc.sh
$ glance image-create --name "cirros-0.3.2-x86_64" --disk-format qcow2 \
  --container-format bare --is-public True --progress < cirros-0.3.2-x86_64-disk.img
+------------------+--------------------------------------+
| Property         | Value                                |
+------------------+--------------------------------------+
```

```
| checksum          | 64d7c1cd2b6f60c92c14662941cb7913     |
| container_format  | bare                                 |
| created_at        | 2014-04-08T18:59:18                  |
| deleted           | False                                |
| deleted_at        | None                                 |
| disk_format       | qcow2                                |
| id                | acafc7c0-40aa-4026-9673-b879898e1fc2 |
| is_public         | True                                 |
| min_disk          | 0                                    |
| min_ram           | 0                                    |
| name              | cirros-0.3.2-x86_64                  |
| owner             | efa984b0a914450e9a47788ad330699d     |
| protected         | False                                |
| size              | 13167616                             |
| status            | active                               |
| updated_at        | 2014-01-08T18:59:18                  |
+-------------------+--------------------------------------+
$ glance image-list
+---------+----------------------+-------------+------------------+----------+--------+
| ID      | Name                 | Disk Format | Container Format | Size     | Status |
+---------+----------------------+-------------+------------------+----------+--------+
| acaf... | cirros-0.3.2-x86_64 | qcow2       | bare             | 13167616 | active |
+---------+----------------------+-------------+------------------+----------+--------+
$ rm -r /tmp/images
```

VIII. Compute Service Install and Configure (Controller node setup)

```
$ apt-get install nova-api nova-cert nova-conductor nova-consoleauth \
  nova-novncproxy nova-scheduler python-novaclient
#===== Nova configure =====
#Edit /etc/nova/nova.conf file
[DEFAULT]
...
rpc_backend = rabbit
rabbit_host = controller
rabbit_password = RABBIT_PASS
...
my_ip = IP
vncserver_listen = IP
vncserver_proxyclient_address = IP
...
auth_strategy = keystone
[keystone_authtoken]
...
auth_uri = http://controller:5000
auth_host = controller
```

```
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = nova
admin_password = NOVA_PASS
[database]
connection = mysql://nova:NOVA_DBPASS@controller/nova

$ rm /var/lib/nova/nova.sqlite
$ su -s /bin/sh -c "nova-manage db sync" nova
$ keystone user-create --name=nova --pass=NOVA_PASS
+----------+----------------------------------+
| Property |  Value                           |
+----------+----------------------------------+
| email    | None                             |
| enabled  | True                             |
| id       | 8e0b71d732db4bfba04943a96230c8c0 |
| name     | nova                             |
| username | nova                             |
+----------+----------------------------------+
$ keystone user-role-add --user=nova --tenant=service --role=admin
+----------+----------------------------------+
| Property | Value                            |
+----------+----------------------------------+
| id       | cd2cb9a39e874ea69e5d4b896eb16128 |
| name     | admin                            |
+-------+-------------------------------------+
$ keystone service-create --name=nova --type=compute \
  --description="OpenStack Compute"
+-------------+----------------------------------+
| Property    |  Value                           |
+-------------+----------------------------------+
| description | OpenStack Compute                |
| enabled     | True                             |
| id          | 060d59eac51b4594815603d75a00aba2 |
| name        | nova                             |
| type        | compute                          |
+-------------+----------------------------------+
$ keystone endpoint-create \
  --service-id=$(keystone service-list | awk '/ compute / {print $2}') \
  --publicurl=http://IP:8774/v2/%\(tenant_id\)s \
  --internalurl=http://IP:8774/v2/%\(tenant_id\)s \
  --adminurl=http://IP:8774/v2/%\(tenant_id\)s
+--------------+-----------------------------------------+
| Property     |  Value                                  |
+--------------+-----------------------------------------+
| adminurl     | http://IP:8774/v2/%(tenant_id)s |
| id           | 4e885d4ad43f4c4fbf2287734bc58d6b        |
```

```
| internalurl  | http://IP:8774/v2/%(tenant_id)s |
| publicurl    | http://IP:8774/v2/%(tenant_id)s |
| region       | RegionOne                       |
| service_id   | 060d59eac51b4594815603d75a00aba2 |
| service_name | nova                            |
| service_type | compute                         |
+--------------+---------------------------------+
$ service nova-api restart
$ service nova-cert restart
$ service nova-consoleauth restart
$ service nova-scheduler restart
$ service nova-conductor restart
$ service nova-novncproxy restart
$ nova image-list
+---------+---------------------+--------+--------+
| ID      | Name                | Status | Server |
+---------+---------------------+--------+--------+
| acaf... | cirros-0.3.2-x86_64 | ACTIVE |        |
+---------+---------------------+--------+--------+
```

IX. Compute Service Install and Configure (Compute node setup)

```
# apt-get install nova-compute-kvm
#===== Nova configure =====
#Edit /etc/nova/nova.conf file
[DEFAULT]
...
auth_strategy = keystone
...
rpc_backend = rabbit
rabbit_host = controller
rabbit_password = RABBIT_PASS
...
my_ip = Compute node IP
vnc_enabled = True
vncserver_listen = 0.0.0.0
vncserver_proxyclient_address = Compute node IP
novncproxy_base_url = http://controller:6080/vnc_auto.html
...
glance_host = controller
[database]
# The SQLAlchemy connection string used to connect to the database
connection = mysql://nova:NOVA_DBPASS@controller/nova
 [keystone_authtoken]
auth_uri = http://IP:5000
auth_host = controller
```

```
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = nova
admin_password = NOVA_PASS

$ rm /var/lib/nova/nova.sqlite
$ service nova-compute restart
```

X. Legacy Networking (nova-network) (Controller node setup)

```
#===== Network configure (Controller node) =====
#Edit /etc/nova/nova.conf file
[DEFAULT]
...
network_api_class = nova.network.api.API
security_group_api = nova

$ service nova-api restart
$ service nova-scheduler restart
$ service nova-conductor restart

#===== Network configure (Compute node) =====
$ apt-get install nova-network nova-api-metadata
#Edit /etc/nova/nova.conf file
[DEFAULT]
...
network_api_class = nova.network.api.API
security_group_api = nova
firewall_driver = nova.virt.libvirt.firewall.IptablesFirewallDriver
network_manager = nova.network.manager.FlatDHCPManager
network_size = 254
allow_same_net_traffic = False
multi_host = True
send_arp_for_ha = True
share_dhcp_address = True
force_dhcp_release = True
flat_network_bridge = br100
flat_interface = INTERFACE_NAME
public_interface = INTERFACE_NAME

$ service nova-network restart
$ service nova-api-metadata restart

#===== Create initial network =====
$ source admin-openrc.sh
```

```
$ nova network-create demo-net --bridge br100 --multi-host T \
  --fixed-range-v4 NETWORK_CIDR
$ nova net-list
+---------+----------+--------------+
| ID      | Label    | CIDR         |
+---------+----------+--------------+
| 84b3... | demo-net | 10.0.1.0/24  |
+---------+----------+--------------+
```

XI. Dashboard Installation

```
$ apt-get install apache2 memcached libapache2-mod-wsgi openstack-dashboard
$ apt-get remove --purge openstack-dashboard-ubuntu-theme
```

XII. Launch an Instance

```
$ source demo-openrc.sh
$ ssh-keygen
$ nova keypair-add --pub-key ~/.ssh/id_rsa.pub demo-key
$ nova keypair-list
+----------+-------------------------------------------------+
| Name     | Fingerprint                                     |
+----------+-------------------------------------------------+
| demo-key | 6c:74:ec:3a:08:05:4e:9e:21:22:a6:dd:b2:62:b8:28 |
+----------+-------------------------------------------------+
$ nova flavor-list
+----+-----------+-----------+------+-----------+-------+-------------+-----------+
| ID | Name      | Memory_MB | Disk | Ephemeral | VCPUs | RXTX_Factor | Is_Public |
+----+-----------+-----------+------+-----------+-------+-------------+-----------+
| 1  | m1.tiny   | 512       | 1    | 0         | 1     | 1.0         | True      |
| 2  | m1.small  | 2048      | 20   | 0         | 1     | 1.0         | True      |
| 3  | m1.medium | 4096      | 40   | 0         | 2     | 1.0         | True      |
| 4  | m1.large  | 8192      | 80   | 0         | 4     | 1.0         | True      |
| 5  | m1.xlarge | 16384     | 160  | 0         | 8     | 1.0         | True      |
+----+-----------+-----------+------+-----------+-------+-------------+-----------+
$ nova image-list
+--------------------------------------+---------------------+--------+--------+
| ID                                   | Name                | Status | Server |
+--------------------------------------+---------------------+--------+--------+
| acafc7c0-40aa-4026-9673-b879898e1fc2 | cirros-0.3.2-x86_64 | ACTIVE |        |
+--------------------------------------+---------------------+--------+--------+
$ nova net-list
+--------------------------------------+----------+------------------+
| ID                                   | Label    | CIDR             |
```

```
+--------------------------------------+----------+------------------+
| 7f849be3-4494-495a-95a1-0f99ccb884c4 | demo-net | 203.0.113.24/29  |
+--------------------------------------+----------+------------------+
$ nova secgroup-list
+--------------------------------------+---------+-------------+
| Id                                   | Name    | Description |
+--------------------------------------+---------+-------------+
| ad8d4ea5-3cad-4f7d-b164-ada67ec59473 | default | default     |
+--------------------------------------+---------+-------------+
$ nova boot --flavor m1.tiny --image cirros-0.3.2-x86_64 --nic net-id=DEMO_NET_ID \
  --security-group default --key-name demo-key demo-instance1
+--------------------------------------+--------------------------------------+
| Property                             | Value                                |
+--------------------------------------+--------------------------------------+
| OS-DCF:diskConfig                    | MANUAL                               |
| OS-EXT-AZ:availability_zone          | nova                                 |
| OS-EXT-STS:power_state               | 0                                    |
| OS-EXT-STS:task_state                | scheduling                           |
| OS-EXT-STS:vm_state                  | building                             |
| OS-SRV-USG:launched_at               | -                                    |
| OS-SRV-USG:terminated_at             | -                                    |
| accessIPv4                           |                                      |
| accessIPv6                           |                                      |
| adminPass                            | ThZqrg7ach78                         |
| config_drive                         |                                      |
| created                              | 2014-04-10T00:09:16Z                 |
| flavor                               | m1.tiny (1)                          |
| hostId                               |                                      |
| id                                   | 45ea195c-c469-43eb-83db-1a663bbad2fc |
| image                                | cirros-0.3.2-x86_64 (acaf...)        |
| key_name                             | demo-key                             |
| metadata                             | {}                                   |
| name                                 | demo-instance1                       |
| os-extended-volumes:volumes_attached | []                                   |
| progress                             | 0                                    |
| security_groups                      | default                              |
| status                               | BUILD                                |
| tenant_id                            | 93849608fe3d462ca9fa0e5dbfd4d040     |
| updated                              | 2014-04-10T00:09:16Z                 |
| user_id                              | 8397567baf4746cca7a1e608677c3b23     |
+--------------------------------------+--------------------------------------+
$ nova list
+--------+----------------+--------+------------+-------------+----------------------+
| ID     | Name           | Status | Task State | Power State | Networks             |
+--------+----------------+--------+------------+-------------+----------------------+
| 45ea.. | demo-instance1 | ACTIVE | -          | Running     | demo-net=203.0.113.26 |
+--------+----------------+--------+------------+-------------+----------------------+
# Permit ICMP (ping):
```

```
$ nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
+-------------+-----------+---------+-----------+--------------+
| IP Protocol | From Port | To Port | IP Range  | Source Group |
+-------------+-----------+---------+-----------+--------------+
| icmp        | -1        | -1      | 0.0.0.0/0 |              |
+-------------+-----------+---------+-----------+--------------+
# Permit secure shell (SSH) access:
$ nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
+-------------+-----------+---------+-----------+--------------+
| IP Protocol | From Port | To Port | IP Range  | Source Group |
+-------------+-----------+---------+-----------+--------------+
| tcp         | 22        | 22      | 0.0.0.0/0 |              |
+-------------+-----------+---------+-----------+--------------+
```

Appendix B

NFS Installation

I. NFS Installation and Configuration (Controller node)

```
$ apt-get install nfs-kernel-server
$ mkdir -p /var/openstack/nfs-storage
#Edit /etc/exports file
/var/openstack/nfs-storage  *(insecure,rw,sync,no_root_squash)

$ service nfs-kernel-server restart
#Edit /etc/nova/nova.conf file
live_migration_flag=VIR_MIGRATE_UNDEFINE_SOURCE,VIR_MIGRATE_PEER2PEER,VIR_MIGRATE_LIVE

$ service nova-api restart
$ service nova-cert restart
$ service nova-consoleauth restart
$ service nova-scheduler restart
$ service nova-conductor restart
$ service nova-novncproxy restart
```

II. NFS Installation and Configuration (Compute node)

```
$ mkdir /var/lib/nova/instances
$ chown nova:nova instances
$ apt-get install nfs-common
$ mount Controller_IP:/var/openstack/nfs-storage /var/lib/nova/instances/
#Edit /etc/fstab file
Controller_IP:/var/openstack/nfs-storage /var/lib/nova/instances/   nfs   defaults   0    0
```

```
#Edit /etc/passwd file
nova:x:107:114::/var/lib/nova:/bin/bash

$ su nova
$ ssh-keygen
# do the free password to login between compute nodes

#Edit /etc/libvirt/libvirtd.conf file
listen_tls = 0
listen_tcp = 1
auth_tcp = "none"

#Edit /etc/init/libvirt-bin.conf file
env libvirtd_opts="-d -l"

#Edit /etc/default/libvirt-bin file
libvirtd_opts=" -d -l"

$ uuidgen
# through the uuidgen will get the code
#Edit /etc/libvirt/libvirtd.conf file
host_uuid=code

$ service libvirt-bin restart

#Edit /etc/nova/nova.conf file
[DEFAULT]
...
live_migration_bandwidth=0
live_migration_retry_count=30
live_migration_flag=VIR_MIGRATE_UNDEFINE_SOURCE,VIR_MIGRATE_PEER2PEER,VIR_MIGRATE_LIVE

$ service nova-compute restart
```

Appendix C

Programming Codes

I. Dynamic Resource Allocation method

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from novaclient import client
import time
import MySQLdb

nova = client.Client(2, "admin", "password", "admin", "http://IP:5000/v2.0")

def resizebigger(instance):
        rStart = time.time()
        if instance.flavor['id'] == "11":
                print "Need to resize",instance.name,"bigger"
                instance.resize("22");
                resizeconfirm(instance);
        elif instance.flavor['id'] == "22":
                print "Need to resize",instance.name,"bigger"
                instance.resize("44");
                resizeconfirm(instance);
        elif instance.flavor['id'] == "44":
                print "Need to resize",instance.name,"bigger"
                instance.resize("88");
                resizeconfirm(instance);
        else:
                print instance.name,"is using the biggest flavor, doesn't need to resize."
        rEnd = time.time()
        print "Resize",instance.name,"cost %f seconds." %(rEnd - rStart)
```

```
def resizesmaller(instance):
        rStart = time.time()
        if instance.flavor['id'] == "88":
                print "Need to resize",instance.name,"smaller"
                instance.resize("44");
                resizeconfirm(instance);
        elif instance.flavor['id'] == "44":
                print "Need to resize",instance.name,"smaller"
                instance.resize("22");
                resizeconfirm(instance);
        elif instance.flavor['id'] == "22":
                print "Need to resize",instance.name,"smaller"
                instance.resize("11");
                resizeconfirm(instance);
        else:
                print instance.name,"is using the smallest flavor, doesn't need to resize."
        rEnd = time.time()
        print "Resize",instance.name,"cost %f seconds." %(rEnd - rStart)

def resizeconfirm(instance):
        print "resizing.."
        name = instance.id
        ninstance = nova.servers.get(name)
        while ninstance.status != "VERIFY_RESIZE":
                ninstance = nova.servers.get(name)
                time.sleep(1)
        instance.confirm_resize();
        print instance.name,"resize done!"

def livemigration(instance):
        lStart = time.time()
        if instance.hostId == "00bfb7537d1ec2cb155d2a2bf46672fe520d03c5024176e7e6cf8ed9":
                print instance.name,"is on compute1"
                print "Live migration",instance.name,"to compute2"
                instance.live_migrate("compute2", block_migration=False, disk_over_commit=False)
        else:
                print instance.name,"is on compute2"
                print "Live migration",instance.name,"to compute1"
                instance.live_migrate("compute1", block_migration=False, disk_over_commit=False)
        name = instance.id
        ninstance = nova.servers.get(name)
        print "migrating.."
        while ninstance.status != "ACTIVE":
                ninstance = nova.servers.get(name)
                time.sleep(1)
        print instance.name,"Live migration done"
        lEnd = time.time()
        print "Live migration",instance.name,"cost %f seconds." %(lEnd - lStart)
```

```
def dra(AVGIutil,i):
        if AVGIutil >= 80:
                print "check the resource of host"
                if s[i].hostId == "00bfb7537d1ec2cb155d2a2bf46672fe520d03c5024176e7e6cf8ed9" \\
                and h[0].vcpus-h[0].vcpus_used>=8 and h[0].free_ram_mb>=8192:
                        print s[i].name,"is on compute1 and resource is enough"
                        resizebigger(s[i]);
                elif s[i].hostId == "80a926d1b3eca95cfaaac24ed1909cfc742d38e7a28cd77d8ed28086" \\
                and h[1].vcpus-h[1].vcpus_used>=8 and h[1].free_ram_mb>=8192:
                        print s[i].name,"is on compute2 and resource is enough"
                        resizebigger(s[i]);
                else:
                        print s[i].name,"is on the host which resource is not enough"
                        print "turn on another host"
                        livemigration(s[i]);
                        resizebigger(s[i]);
        elif AVGIutil <= 30:
                resizesmaller(s[i]);
        else:
                print s[i].name,"CPU utilization is good, doesn't need to resize."

tStart = time.time() #total DRA+ES start time
tdStart = time.time() #total DRA start time
while True:
        dStart = time.time()
s = nova.servers.list()
h = nova.hypervisors.list()
db = MySQLdb.connect(host="localhost", user="root", passwd="password", db="ServerMonitor")
cursor = db.cursor()
cursor.execute("SELECT `IID`,AVG(`USED`) as `CPUAVG` FROM `ICPU` \\
WHERE  `TIME`>'2015-06-23 11:02:00' AND `TIME`<'2015-06-23 12:02:00' GROUP BY `IID`")
result = cursor.fetchall()
for row in result:
        i = 7-row[0]
        AVGIutil = row[1]
        print s[i].name,"CPU utilization is",row[1],"%"; #row[0]=IID
        dra(AVGIutil,i);
        print "";
        dEnd = time.time()
        time.sleep(2)
        if dEnd - dStart < 2:
                break
tdEnd = time.time() #total DRA end time
print "DRA method total cost %f seconds." %(tdEnd - tdStart)
print "";
import es_method
tEnd = time.time() #total DRA+ES end time
```

```
print "DRA and ES method total cost %f seconds." %(tEnd - tStart)
print "program over!"
```

II. Energy Saving method

```
#!/usr/bin/python
# -*- coding: utf-8 -*-
from novaclient import client
import time
import subprocess
import sys

nova = client.Client(2, "admin", "password", "admin", "http://IP:5000/v2.0")

h = nova.hypervisors.list()
s = nova.servers.list()

def c2toc1(instance):
        if instance.hostId == "80a926d1b3eca95cfaaac24ed1909cfc742d38e7a28cd77d8ed28086":
                lStart = time.time()
                print instance.name,"is on compute2"
                print "Live migration",instance.name,"to compute1"
                instance.live_migrate("compute1", block_migration=False, disk_over_commit=False)
                name = instance.id
                ninstance = nova.servers.get(name)
                print "migrating.."
                while ninstance.status != "ACTIVE":
                        ninstance = nova.servers.get(name)
                        time.sleep(1)
                print instance.name,"Live migration done"
                lEnd = time.time()
                print "Live migration",instance.name,"cost %f seconds." %(lEnd - lStart)
        else:
                print instance.name,"is on compute1, doesn't need to Live migration."


def c1toc2(instance):
        if instance.hostId == "00bfb7537d1ec2cb155d2a2bf46672fe520d03c5024176e7e6cf8ed9":
                lStart = time.time()
                print instance.name,"is on compute1"
                print "Live migration",instance.name,"to compute2"
                instance.live_migrate("compute2", block_migration=False, disk_over_commit=False)
                name = instance.id
                ninstance = nova.servers.get(name)
                print "migrating.."
                while ninstance.status != "ACTIVE":
```

```
                        ninstance = nova.servers.get(name)
                        time.sleep(1)
                print instance.name,"Live migration done"
                lEnd = time.time()
                print "Live migration",instance.name,"cost %f seconds." %(lEnd - lStart)
        else:
                print instance.name,"is on compute2, doesn't need to Live migration."

def es():
        compute1 = h[0]
        compute2 = h[1]
        eStart = time.time()
        if compute1.vcpus_used+compute2.vcpus_used <= 32 \\
        and compute1.memory_mb_used+compute2.memory_mb_used <= 63488:
                if compute2.running_vms <= compute1.running_vms and compute2.running_vms != 0:
                        print "migrate",compute2.hypervisor_hostname,"VM to",\\
                                and compute1.hypervisor_hostname
                        for i in range(0,len(s)): #0<=i<len(s)
                                c2toc1(s[i]);
                        print "All VM are on compute1 and then shutdown compute2"
                        ssh = subprocess.Popen(["ssh", "%s" % "compute2", "init 0"]) #shutdown compute
                        time.sleep(5)
                        print "";
                elif compute1.running_vms <= compute2.running_vms and compute1.running_vms != 0:
                        print "migrate",compute1.hypervisor_hostname,"VM to",\\
                                and compute2.hypervisor_hostname
                        for i in range(0,len(s)): #0<=i<len(s)
                                c1toc2(s[i]);
                        print "All VM are on compute2 and then shutdown compute1"
                        time.sleep(5)
                        ssh = subprocess.Popen(["ssh", "%s" % "compute1", "init 0"]) #shutdown compute
                        print "";
                else:
                        if compute1.running_vms != 0 and compute2.running_vms == 0:
                                print "All VM are on compute1."
                        elif compute2.running_vms != 0 and compute1.running_vms == 0:
                                print "All VM are on compute2."
                        else:
                                print "There are no VM."
                        print "";
        else:
                print "undo"
                print "";
        eEnd = time.time()
        print "ES method total cost %f seconds." %(eEnd - eStart)
```

Appendix D

Monitor Codes

I. Server and Virtual Machines Information Monitor program

```
# -*- coding: utf-8 -*-
import sys
import os
import atexit
import time
import psutil

print "Loading..."
time.sleep(3)

line_num = 1
ShowInfo="On"
def print_line(str):
 if ShowInfo=='On':
  print str

#function of Get CPU State
def getCPUstate(interval=1):
    return (str(psutil.cpu_percent(interval)))

#function of Get Memory
def getMemorystate():
    phymem = psutil.phymem_usage()
    buffers = getattr(psutil, 'phymem_buffers', lambda: 0)()
    cached = getattr(psutil, 'cached_phymem', lambda: 0)()
    used = phymem.total - (phymem.free + buffers + cached)
    line = " Memory: %5s%% %6s/%s" % (
```

```
        phymem.percent,
        str(int(used / 1024 / 1024)) + "M",
        str(int(phymem.total / 1024 / 1024)) + "M"
    )
    return line

def bytes2human(n):
    """
    >>> bytes2human(10000)
    '9.8 K'
    >>> bytes2human(100001221)
    '95.4 M'
    """
    symbols = ('K', 'M', 'G', 'T', 'P', 'E', 'Z', 'Y')
    prefix = {}
    for i, s in enumerate(symbols):
        prefix[s] = 1 << (i+1)*10
    for s in reversed(symbols):
        if n >= prefix[s]:
            value = float(n) / prefix[s]
            return '%.2f %s' % (value, s)
    return '%.2f B' % (n)

def poll(interval):
    """Retrieve raw stats within an interval window."""
    tot_before = psutil.network_io_counters()
    pnic_before = psutil.network_io_counters(pernic=True)
    diskio_before=psutil.disk_io_counters()
    # sleep some time
    time.sleep(interval)
    tot_after = psutil.network_io_counters()
    pnic_after = psutil.network_io_counters(pernic=True)
    diskio_after=psutil.disk_io_counters()
    # get cpu state
    cpu_state = getCPUstate(interval)
    # get memory
    memory_state = getMemorystate()
    return (tot_before, tot_after, pnic_before, pnic_after,\\
        cpu_state,memory_state,diskio_before,diskio_after)

def refresh_window(tot_before, tot_after, pnic_before, pnic_after,\\
        cpu_state,memory_state,diskio_before,diskio_after):
    os.system("cls")
    """Print stats on screen."""
    print_line(time.asctime()+" | "+cpu_state+" | "+memory_state)

    #CPU USED INFO
    CPU_USED=cpu_state
```

```
    #RAM USED INFO
    phymem = psutil.phymem_usage()
    buffers = getattr(psutil, 'phymem_buffers', lambda: 0)()
    cached = getattr(psutil, 'cached_phymem', lambda: 0)()
    used = phymem.total - (phymem.free + buffers + cached)
    RAM_PA=phymem.percent
    RAM_USED=str(int(used / 1024 / 1024))
    RAM_ALL=str(int(phymem.total / 1024 / 1024))
    #DISK IO INFO
    DISK_READ=str(diskio_after.read_bytes - diskio_before.read_bytes)
    DISK_WRITE=str(diskio_after.write_bytes - diskio_before.write_bytes)

    #SEND-TO-DB SERVER
    import httplib,urllib
    httpClient = None
    try:
        ServerID="1"
        params = urllib.urlencode({'CPU_USED': CPU_USED, 'RAM_PA': RAM_PA,\\
         'RAM_ALL': RAM_ALL,'RAM_USED':RAM_USED,'SID':ServerID})
        headers = {"Content-type": "application/x-www-form-urlencoded", "Accept": "text/plain"}
        httpClient = httplib.HTTPConnection('IP', 80, timeout=3)
        httpClient.request('POST', '/power/API/ServerInfo.php',params,headers)

        #response HTTPResponse
        response = httpClient.getresponse()
        print response.status
        print response.reason
        print response.read()

    except Exception, e:
        print e
    finally:
        if httpClient:
            httpClient.close()
try:
    interval = 0
    while 1:
        args = poll(interval)
        refresh_window(*args)
        interval = 1
except (KeyboardInterrupt, SystemExit):
    pass
```

II. PDU Information program

```
<?php
```

```
function get_server_info($host, $community, $objectid) {
$a = snmpget($host, $community, $objectid);

$tmp = explode(":", $a);
if (count($tmp) > 1) {
$a = trim($tmp[1]);
}
return $a;
}
$host="IP";
$community="public";

for($i=1;$i<=8;$i++){
$Power = get_server_info($host,$community,".1.3.6.1.4.1.13742.4.1.2.2.1.7.".$i);
echo $i."-Power:".$Power."<br>";

$I = get_server_info($host,$community,".1.3.6.1.4.1.13742.4.1.2.2.1.4.".$i);
echo $i."-I:".$I."<br>";

$V = get_server_info($host,$community,".1.3.6.1.4.1.13742.4.1.2.2.1.6.".$i);
echo $i."-V:".$V."<br>";

$PF = get_server_info($host,$community,".1.3.6.1.4.1.13742.4.1.2.2.1.9.".$i);
echo $i."-PF:".$PF."<br>";
$sql="INSERT INTO `ServerMonitor`.`Power` (`ID`, `TIME`, `SID`, `V`, `C`, `P`, `PF`) \\
 VALUES (NULL, CURRENT_TIMESTAMP, '".$i."', '".$V."', '".$I."', '".$Power."','".$PF."');";
//echo $sql;
mysql_query($sql) or die('MySQL query error');
$sql="UPDATE `ServerMonitor`.`PowerRealTime` SET `TIME` = CURRENT_TIMESTAMP(), \\
`V` = '".$V."', `C` = '".$I."', `P` = '".$Power."',\\
 `PF` = '".$PF."' WHERE `PowerRealTime`.`ID` = ".$i.";";
mysql_query($sql) or die('MySQL query error');
}
?>
```