東海大學

資訊工程研究所

碩士論文

指導教授: 楊朝棟博士

利用異質物件儲存技術實作軟體定義儲存服務

Implementation of Software-Defined Storage Service with Heterogeneous Object Storage Technologies

研究生: 沈宇權

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 沈 宇 權 所提之論文

利用異質物件儲存技術實作軟體定義儲存服務

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 林迪清 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

在資訊時代的迅速發展下，不只是個人電腦的普及化，現在更讓這些資訊的使用方式轉變成雲端運算以及雲端服務。雲端服務是一種概念，使用者以手邊的端點設備將需求透過網路丟到雲端環境上，並透過雲端環境上的運算處理後給予使用者一個回覆。其中一個最為方便的例子就是雲端儲存。軟體定義儲存是一種虛擬化的技術，透過軟體做資源的整合將提高易用性和可用性，SDS 是一項逐漸流行的技術，在網路上有著許多不同的開放原始碼專案。希望能夠使用這樣的技術來幫助我們更有效的利用硬體和軟體上資源。本文將實現一個結合不同儲存技術並符合 SDS 概念的雲端服務。在系統架構中，我們採用 OpenStack Swift 和 Ceph 這兩項開源軟體，使系統有更好的兼容性設計，因為包含了這兩項技術，系統的架構設計上也繼承了該儲存系統的擴充性。實現一套自動分配機制是本文的重要目標以及貢獻，這讓使用者上傳檔案之後能自動分配至較適合此檔案的軟體儲存技術，而在管理者方面透過參數設定此分配機制，能夠讓此系統有更靈活的操作也使得本系統更為智能。最後提供一個有善的使用者介面，更符合雲端服務何時何地任何設備都可以使用的概念。

關鍵字: 雲端服務，儲存服務，軟體定義儲存，物件儲存，自動分配機制，整合儲存

Abstract

With the rapid development of information, personal computers are not only popular but also provide cloud services. Cloud Service is a concept that users can upload their requirement via internet to cloud environment and then receive a response by post-processing of the cloud environment, for example cloud storage. Software-defined storage (SDS) is a kind of virtualization technology for cloud storage. It uses the software to integrate the resources so as to improve accessibility and usability. There are many different open source projects for SDS in the Internet. This work aims to utilize these open source projects of SDS to improve the integration of the hardware and software resources effectively. In other words, we integrate various SDS open source projects or technologies to implement a cloud system. In the system architecture, we use some open-source softwares to make the proposed system more compatible and automatically assign a file to an appropriate storage location after users upload files. In addition, a manager can set some parameters to make this system more flexible. We also provide a high usability user interface. The user interface is designed as a web application. According to the concept of cloud services, this interface can be used anywhere and anytime.

Keywords: Cloud service, Storage service, Software-Defined Storage, Object storage, Automatic distribution, Integrated storage

致謝詞

首先誠摯的感謝指導教授楊朝棟博士，老師悉心的教導使我得以一窺資訊工程領域的深奧，不時的討論並指點我正確的方向，使我在這些年中獲益匪淺。老師對學問的嚴謹更是我輩學習的典範。

本論文的完成另外亦得感謝每一個口試委員的熱心指導，感謝林迺衛老師、朱正忠老師、賴冠州老師、時文忠老師，不辭辛勞撥空擔任我的口試委員，提供給了我許多的建議。因為有你們的體諒及幫忙，使得本論文能夠更完整而嚴謹。

兩年裡的日子，實驗室裡共同的生活點滴，學術上的討論、言不及義的閒扯、讓人又愛又怕的宵夜、趕作業的革命情感、因為睡太晚而遲到的實驗室會議.....，感謝眾位學長姐、同學、學弟妹的共同砥礪，你/妳們的陪伴讓兩年的研究生活變得絢麗多彩。

感謝實驗室的夥伴們在計畫上互相幫忙，大家共同做事一起努力的感覺真的很棒。不僅是計畫上，當然課程、研究還有論文，這些都不得不好好的跟大家說聲謝謝，我在大家身上學習到很多事物。

那些在背後默默支持我的家人以及女朋友更是我前進的動力，沒有你們的體諒、包容，相信這兩年的生活將是很不一樣的光景。

最後，謹以此文獻給我摯愛的雙親。

Table of Contents

摘要 **I**

Abstract **II**

致謝詞 **III**

Table of Contents **IV**

List of Figures **VI**

List of Tables **VII**

1 Introduction **1**
- 1.1 Motivation . . . 1
- 1.2 Thesis Goal and Contributions . . . 2
- 1.3 Thesis Organization . . . 2

2 Background Review and Related Work **3**
- 2.1 Background Review . . . 3
  - 2.1.1 Software-Defined Storage . . . 3
  - 2.1.2 Swift . . . 4
  - 2.1.3 Ceph . . . 6
  - 2.1.4 OpenStack . . . 8
  - 2.1.5 COSBench . . . 10
  - 2.1.6 EMC ViPR . . . 11
  - 2.1.7 Cubic Spline . . . 14
- 2.2 Related Works . . . 15

3 System Design and Implementation **18**
- 3.1 System Architecture . . . 19
- 3.2 Design Flow . . . 19
- 3.3 System Implementation . . . 20
  - 3.3.1 Storage Service Deployment . . . 20
  - 3.3.2 User Service . . . 23
  - 3.3.3 File Distribution Mechanism . . . 26

4 Experiments and Results **30**
4.1 Experimental Environment 30
4.2 Performance 31
4.3 User Interface 36

5 Conclusions and Future Work **42**
5.1 Concluding Remark 42
5.2 Future Works 43

References **44**

Appendix **48**

A OpenStack Installation **48**

B Swift Installation **57**

C Ceph Installation **64**

List of Figures

2.1 Swift architecture 5
2.2 Ceph architecture 6
2.3 Ceph object storage architecture 7
2.4 Openstack Juno arch 10
2.5 COSBench Architecture 11
2.6 ViPR System Architecture 12
2.7 ViPR Cluster 13
2.8 Cubic Spline Schematic Diagram 15

3.1 System Architecture 19
3.2 Controller Architecture 20
3.3 OpenStack Overview 21
3.4 OpenStack instances 21
3.5 Swift enviroment 22
3.6 Ceph enviroment 22
3.7 Responsive Web Design 24
3.8 Asynchronous JavaScript and XML 26
3.9 File transfer speeds 27
3.10 File transfer speeds using Cubic Spline 27
3.11 All file transfer speeds using Cubic Spline 28

4.1 Network infrastructure speed 32
4.2 Disk infrastructure write speed 33
4.3 Disk infrastructure read speed 33
4.4 Measure the upload speed 34
4.5 Measure the download speed 35
4.6 User interface overview 36
4.7 Overviwe page 37
4.8 My storage page view 37
4.9 Upload the files 38
4.10 File details 39
4.11 Account page view 40
4.12 Responsive Web Design 41

List of Tables

3.1 Software & language Specification 23

4.1 Hardware Specification 31
4.2 Storage Environment Specification 31
4.3 Software Specification 31

Chapter 1

Introduction

In the rapid development of the information age, personal computers have been very popular, and nowadays cloud computing [1–4] is becoming the trend. Many large enterprises such as Google, Amazon, and Yahoo provide a variety of cloud services. Not only these enterprises but also government and academic institutions have entered this area to build their own cloud. Building cloud servers are very relatively expensive, including the equipment purchase, technical maintenance, and equipment maintenance costs.

1.1 Motivation

Cloud service is a concept. The users upload their requirements via the Internet to the cloud environment, and obtain replies of processing results from the cloud environment. One of the most convenient examples is cloud storage [5–8]. More and more users use this convenient way to upload and store data, and enterprises are also attracted to join the cloud service delivery. If a company wants to build its own cloud storage, how to choose the hardware device or technical resources are important issues. The software-defined storage (SDS) is a kind of virtualization technology. SDS uses the software to integrate the resource and improve accessibility and usability, and it has become an increasingly popular technology.

Besides, in the Internet there are many different open source projects; these technologies can be adopted to help us more effectively use the hardware and software resources.

1.2 Thesis Goal and Contributions

This thesis will implement an integration of the cloud service with SDS technology. In the system architecture, we will use some open-source software, making the system more compatible. And implement a mechanism to automatically assign files, uploaded by users, to an appropriate storage system. In addition, system managers can set parameters to customize the mechanism, making this service has a flexible configuration and makes it more intelligent. We also provide a high usability user interface. The user interface is designed as a web application. According to the concept of cloud services, this interface can be used anywhere and anytime.

1.3 Thesis Organization

Chapter 2 will describe some background information, including cloud service, SDS, OpenStack, Swift, Ceph and COSBench. Chapter 3 will introduce our experimental environment and methods, and the overall architecture. Chapter 4 presents and analyses experimental results. Finally, Chapter 5 summarizes this thesis by pointing out its major contributions and directions for the future work.

Chapter 2

Background Review and Related Work

2.1 Background Review

2.1.1 Software-Defined Storage

Software-Defined Storage (SDS) [9–11] is a concept for the computer data storage technology which separates storage hardware from the software that manages the storage infrastructure. The software enabling SDS environments can provide policy management for feature options such as deduplication, replication, thin provisioning, snapshots and backup. Characteristics of SDS could include any or all of the following features:

- Abstraction of logical storage services and capabilities from the underlying physical storage systems, and in some cases, pooling across multiple different implementations. Since data movement is relatively expensive and slow compared to compute and services (the "data gravity" problem in infonomics), pooling approaches sometimes suggest leaving it in place and creating a mapping layer to it that spans arrays. Examples include

- Automation with policy-driven storage provisioning with service-level agreements replacing technology details. This requires management interfaces that span traditional storage array products, as a particular definition of separating the ”control plane” from ”data plane,” in the spirit of OpenFlow. Prior industry standards efforts include the Storage Management Initiative – Specification (SMI-S) which began in 2000.
- Commodity hardware with storage logic abstracted into a software layer. This is also described as a clustered file system for converged storage.
- Scale-out storage architecture.

2.1.2 Swift

OpenStack Object Storage (Swift) [12] is a scalable redundant storage system. Objects and files are 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.

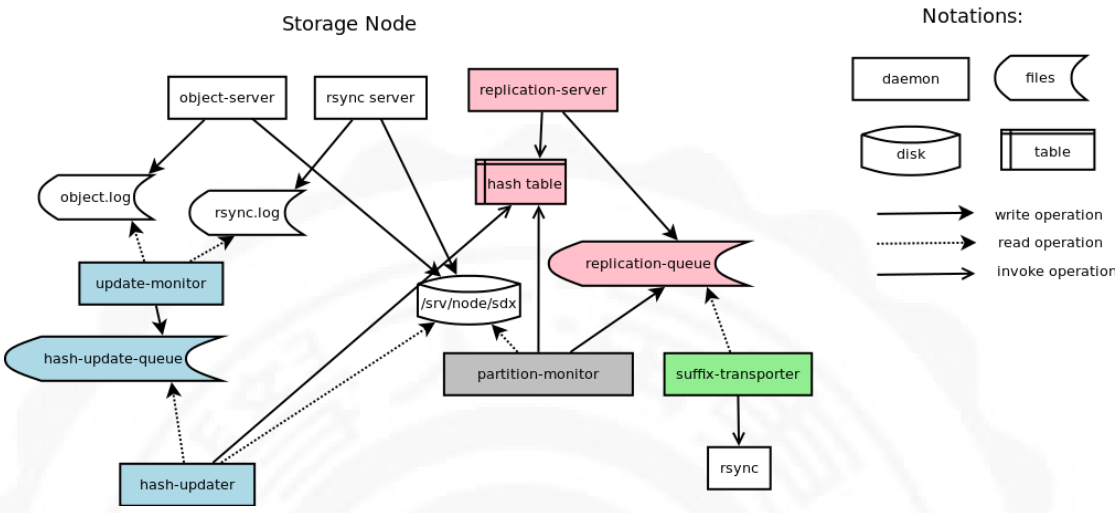


FIGURE 2.1: Swift architecture

As the figure 2.1 shows that the icons with colors are the main components of this design. All of them is independent with the current object-replicator. The origin logic of object-replicator was split into four parts with different colors. the components with the color of cyan are in charge of calculating hash in real-time; the components with the color of pink are in charge of indexing the hash of suffix and partition directories, receiving and sending requests to compare the hash of partition or suffix, generating jobs of replicating suffix directories to the replication-queue; The partition-monitor is in charge of checking the partition whether to move at interval; The suffix-transporter is in charge of monitoring the replication-queue and invoking the rsync to sync suffix directories.

- Proxy Server: It is responsible for tying together the rest of the Swift architecture. For each request, it will look up the location of the account, container, or object in the ring and route the request accordingly. For Erasure Code type policies, the Proxy Server is also responsible for encoding and decoding object data.

- Object Server: It is a very simple storage server that can store, retrieve and delete objects stored on local devices. Objects are stored as binary files on the filesystem with metadata stored in the file's extended attributes (xattrs).

This requires that the underlying filesystem choice for object servers support xattrs on files. Some filesystems, like ext3, have xattrs turned off by default.

- Container Server: The Server's primary job is to handle listings of objects. It doesn't know where those object's are, just what objects are in a specific container. The listings are stored as sqlite database files, and replicated across the cluster similar to how objects are. Statistics are also tracked that include the total number of objects, and total storage usage for that container.

- Account Server: It is very similar to the Container Server, excepting that it is responsible for listings of containers rather than objects.

2.1.3 Ceph

Ceph [13] is a software storage platform designed to present object, block, and file storage from a single distributed computer cluster. Ceph is a distributed storage designed to provide excellent performance, reliability and scalability.The Ceph file system has three main components: the client, each instance of which exposes a near-POSIX file system interface to a host or process; a cluster of OSDs, which collectively stores all data and metadata; and a metadata server cluster, which manages the namespace (file names and directories) while coordinating security, consistency and coherence. The system is designed to be both self-healing and self-managing and strives to cut both administrator and budget costs.

FIGURE 2.2: Ceph architecture

- Object Storage: Ceph is a distributed object storage and file system designed to provide excellent performance, reliability and scalability. Its software

libraries offer client applications with direct access to the reliable autonomic distributed object store (RADOS) object-based storage system, and also provide a basis for some of Ceph's advanced features, including RADOS Block Device (RBD), RADOS Gateway, and the Ceph File System.

The librados software libraries enable applications written in C, C++, Java, Python and PHP. The RADOS Gateway also exposes the object store as a RESTful interface which can present as both native Amazon S3 and OpenStack Swift APIs. The librados libraries provide advanced features, including:

- Partial or complete reads and writes
- Snapshots
- Atomic transactions with features like append, truncate and clone range
- Object level key-value mappings

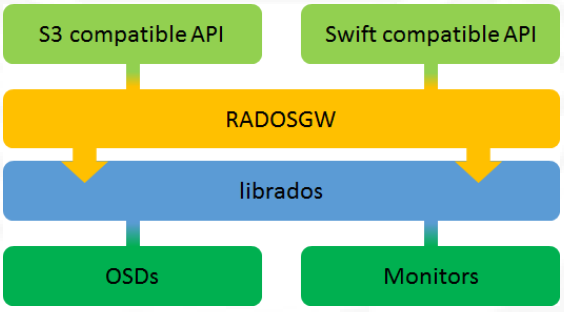


FIGURE 2.3: Ceph object storage architecture

- Block Storage: Ceph's object storage system allows users to mount Ceph as a thinly provisioned block device. Ceph's RADOS Block Device (RBD) provides access to block device images that are striped and replicated across the entire storage cluster. When an application writes data to Ceph using a block device, Ceph automatically stripes and replicates the data across the cluster. Ceph's RADOS Block Device (RBD) also integrates with KVMs, bringing Ceph's virtually unconstrained storage to KVMs running on user's Ceph clients.

Ceph RBD interfaces with the same Ceph object storage system that provides the librados interface and the CephFS file system, and it stores block device images as objects. Since RBD is built on top of librados, RBD inherits librados's capabilities, including read-only snapshots and revert to snapshot. Ceph's object storage system is not bounded to native binding or RESTful APIs. User can mount Ceph as a thinly provisioned block device. When write data to Ceph using a block device, Ceph automatically stripes and replicates the data across the cluster. By striping images across the cluster, Ceph increases read access performance for large block device images.

- File System: Ceph's file system (CephFS) runs on top of the same object storage system that provides object storage and block device interfaces. Ceph provides a POSIX-compliant network file system that aims for high performance, large data storage, and maximum compatibility with legacy applications. Compared to many object storage systems available today Ceph's object storage system offers a significant feature: a traditional file system interface with POSIX semantics. Object storage systems are a significant innovation, but they supplement rather than replace traditional file systems. The Ceph metadata server cluster provides a service that maps the directories and file names of the file system to objects stored within RADOS clusters. The metadata server cluster can expand, contract, and dynamically rebalance the file system to distribute data evenly among cluster hosts. As storage requirements grow for legacy applications, organizations can configure their legacy applications to use the Ceph file system. This means user can run one storage cluster for object, block and file-based data storage. This ensures high performance and prevents heavy loads on specific hosts within the cluster.

2.1.4 OpenStack

OpenStack [14] is an IaaS cloud computing project for public and private clouds. It is free open source software released under the terms of the Apache License.

The project aims to deliver solutions for all types of clouds by being simple to implement, massively scalable, and features rich. The technology consists of a series of interrelated projects delivering various components for a cloud infrastructure solution. Founded by Rackspace Hosting and NASA, OpenStack has grown to be a global software community of developers collaborating on a standard and massively scalable open source cloud operating system. Its mission is to enable any organization to create and offer cloud computing services running on standard hardware.

The project is managed by the OpenStack Foundation, a non-profit corporate entity established in September 2012 to promote, protect and empower OpenStack software and its community.

OpenStack offers flexibility and choice through a highly engaged community of over 6,000 individuals and over 190 companies including Rackspace, such as Intel, AMD, Canonical, SUSE Linux, Inktank, Red Hat, Groupe Bull, Cisco, Dell, HP, IBM, NEC, VMware and Yahoo. It is portable software, but is mostly developed and used on operating systems running Linux.

The technology consists of a series of interrelated projects that control large pools of processing, storage, and networking resources through-out a datacenter, all managed through a dashboard that gives administrators control while empowering its users to provision resources through a web interface.

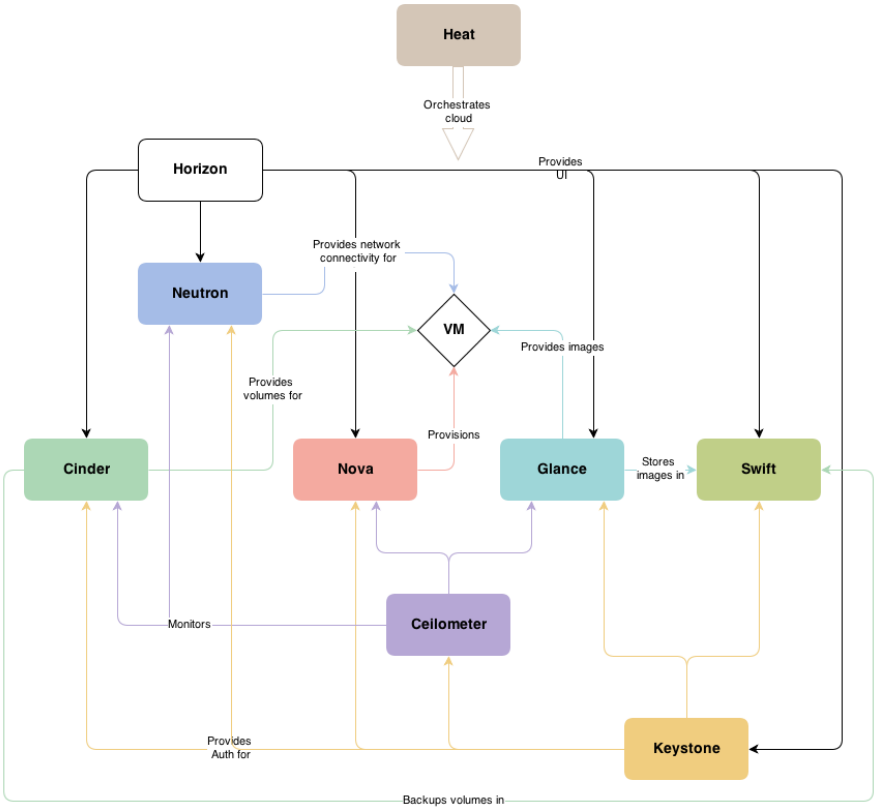


FIGURE 2.4: Openstack Juno arch

In this work, we use version Juno. The architecture is as shown in Figure 2.4, in which Horizon provides GUI, Neutron provides networking, Nova is virtual machine provisions, Glance provides virtual machine images, Keystone controls all authentication in OpenStack, Swift backups virtual machine images, Cinder provides volumes for virtual machines, and Ceilometer monitors Cinder, Neutron, Nova and Glance. We just use Horizon, Neutron, Nova, Glance and Keystone in our model.

2.1.5 COSBench

Cloud Object Storage Benchmark (COSBench) [15] is a benchmarking tool to measure the performance of Cloud Object Storage services. COSBench has two components, namely controller and driver, and can operate in two different modes, either independent or managed.The architecture is as shown in Figure 2.5. In independent mode, only driver is used. At runtime, it loads configurations and

spawns agent threads which stress the target service in a way consistent with the user-defined usage pattern. Under managed mode, on the other hand, both components are required in that the controller is added to supervise multiple drivers so that they can work collaboratively in a distributed environment. In this case, each driver will spawn an additional daemon thread for receiving and responding controller commands.

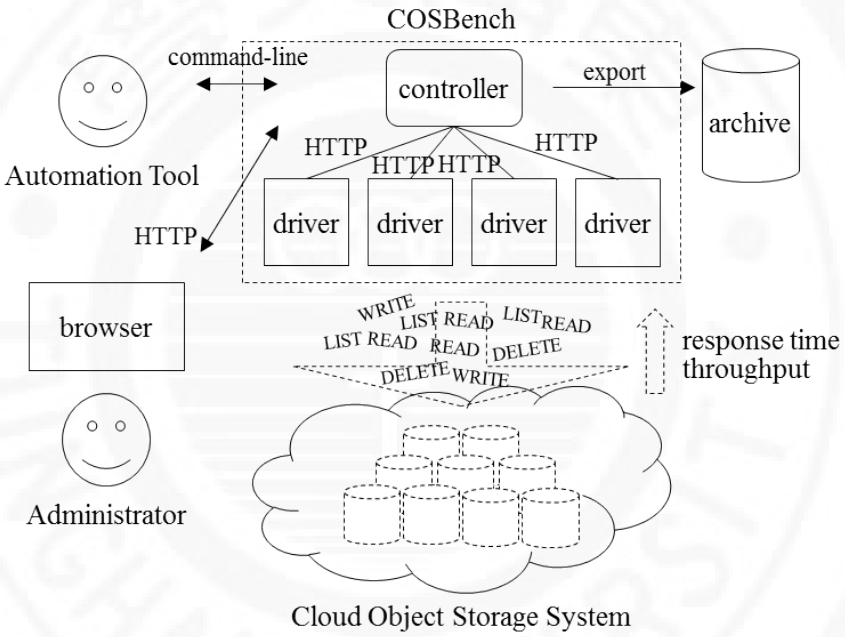


FIGURE 2.5: COSBench Architecture

2.1.6 EMC ViPR

EMC Virtualization Platform Reinvented (ViPR) is a logical storage system, not a physical storage. It can integrate EMC storage and third-party storage in a storage pool, and manage it as a single system, but still save the worth of original storage. ViPR can replicate data across several different place and data center with different store product, and it provides a unified block store, object store and file system and other services. At the same time, ViPR provides a unified metadata service and self-service deployment, measurement and monitoring services.

ViPR run on three to five virtual server machines. It consists of the control plane and data plane; the first part realizes automatic store, and the second part

provides data service by building on the first part. ViPR provides some APIs to let users to use services and manage storage, as shown in Figure 2.6.

One part of ViPR is the control plane. It can manage all applications of storage array, such as data mining, storage resource searching, and capacity counting and reporting. The controller controls the application of virtual storage array, and manages the storage resource. In order to achieve the goal, ViPR virtualizes the control path of physical storage, and runs the function with it. By the virtual plane, people can manage the storage pool and split it to different virtual storage arrays with policies, just like virtualization of the server.

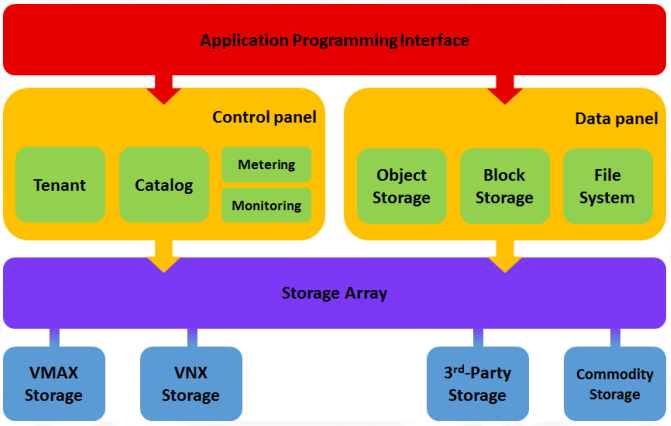


FIGURE 2.6: ViPR System Architecture

In addition to the controller and data service, ViPR provides the open RESTful API. The developers can develop new services without the limit of the hardware. Through these APIs, people can access data, and add, delete, modify, monitor and measure the logical storage resources.

ViPR is built by the scale-out architecture; if it is built with at least three clusters, the architecture will provide the ability of high availability, load balance and system upgrade and so on. It provides RESTfulAPI, GUI(console), CLI and SDK to let user control it with high flexibility as shown in Figure 2.7. And ViPR can provide the automatic of disaster recovery. Through the continuous remote replication technology, it can provide the ability of data replication continuously to successfully achieve the goal of disaster recovery.

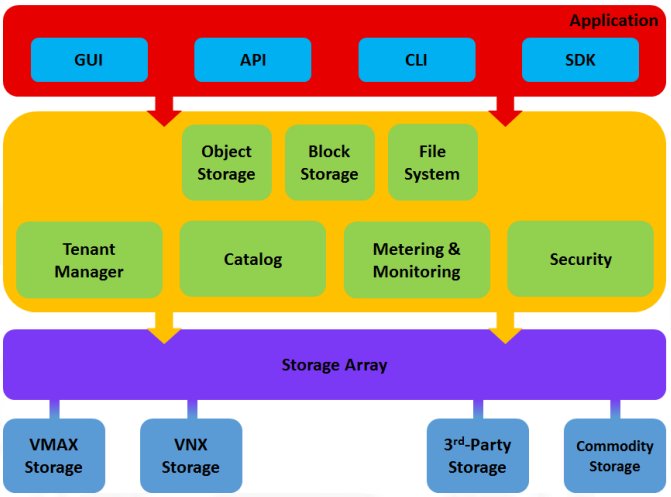


FIGURE 2.7: ViPR Cluster

The data service of ViPR has some features:

- Unified Storage: ViPR offers the traditional layer on top of cloud storage data type, with the following advantages: tradition by allowing local access to the underlying storage, improved compatibility with existing applications; while allowing manipulation of data at the right time through different methods to improve the efficiency of the work; and it can convert existing data to the cloud easily.

- Heterogeneous Storage: ViPR provides an engine for a variety of storage devices, for a given environment or use case can select the appropriate storage capacity and hardware, and the user can reuse the existing storage hardware investments, but also applies to mixed ViPR additional scenarios, such as across different devices tiered storage and replication.

- Enterprise-grade Storage: cloud storage platforms often lack some of the features needed in enterprise scenarios, such as snapshots and compliance. The ability to leverage and extend ViPR data services to the underlying storage devices to provide enterprise-class cloud storage, such as ViPR can provide incremental real-time snapshot of the object store.

- Flexible Storage: ViPR data services are implemented by software, which makes ViPR data service can be deployed on any server in the data center,

in simple, reliable, lightweight and scalable design; And while all of these functions underlying storage devices, IP is not locked in any storage array.

- Extensible Storage: ViPR data services provides a rich API, third-party services can use it to develop their own systems. In addition, ViPR will also expose the raw building blocks, these modules can provide the core IP cloud-scale services, including distributed B + tree, metadata; third party can use these to develop services that modules can be placed inside or outside ViPR to extend a platform.

- Object Storage: object storage services like Amazon S3 ViPR provided the model for the image. ViPR APIs support AmazonS3, OpenStackSwift and EMC's Atmos, vNextAPI support EMCCentera CAS. The ViPR also provides some extensions, including Byte range updates, atomic increase, rich ACL snapshots, metering and billing, and so on. In addition, ViPR provides such functions as file can be accessed directly on the same underlying file storage device object storage, and the performance of the local file system consistency.

2.1.7 Cubic Spline

In mathematics, a spline is a numeric function that is piecewise-defined by polynomial functions, and which possesses a sufficiently high degree of smoothness at the places where the polynomial pieces connect.

In interpolating problems, spline interpolation is often preferred to polynomial interpolation because it yields similar results to interpolating with higher degree polynomials while avoiding instability due to Runge's phenomenon. In computer graphics, parametric curves whose coordinates are given by splines are popular because of the simplicity of their construction, their ease and accuracy of evaluation, and their capacity to approximate complex shapes through curve fitting and interactive curve design.

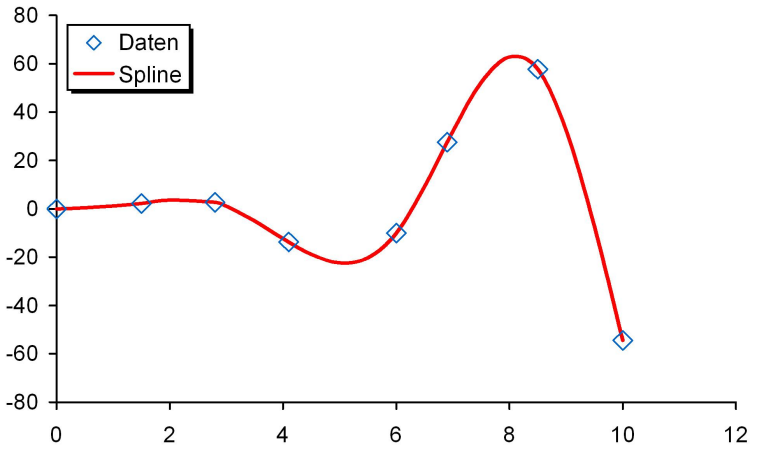


FIGURE 2.8: Cubic Spline Schematic Diagram

A cubic spline is a spline constructed of piecewise third-order polynomials which pass through a set of m control points. The second derivative of each polynomial is commonly set to zero at the endpoints, since this provides a boundary condition that completes the system of m-2 equations. This produces a so-called "natural" cubic spline and leads to a simple tridiagonal system which can be solved easily to give the coefficients of the polynomials. However, this choice is not the only one possible, and other boundary conditions can be used instead.

2.2 Related Works

In the early development of cloud services, the exact meaning of Software-Defined is still inconclusive. The concept of "Software-defined data center" is first proposed by VMware due to the fact that software becomes more important. With the concept of virtualization in building all the hardware resources as a resource pool, users apply software to control the arrangement of hardware resources.

When using programmable software to control the arrangement of hardware resources, there is no need to think about how to manipulate server, security guard and allocate resources. In other words, all the resources are functioning perfectly.

Cloud computing brought more possibilities so that software-Defined may be different concepts in hardware and software architecture. These concepts become the custom functions and automation of operations. Many software-defined storage research papers and products are proposed.

Yang et al. [16] proposed an integrated storage service. They use the Open Stack to build and manage the cloud services, and use software to integrate storage resources including Hadoop HDFS, Ceph and Swift on Open Stack to achieve the concept of SDS. The software used can integrate different storage devices to provide an integrated storage array and build a virtual storage pool, so that users do not feel restrained by the storage devices.

My thesis is based on Yang' s paper. We refer the infrastructure environment and improve the architecture. Then, we propose the mechanism to store data. In addition, we design the new use interface. These are the main difference between two papers.

EMC Virtualization Platform Reinvented (ViPR) [17] is a logical storage system, not a physical storage. It can integrate EMC storage and third-party storage in a storage pool, and manage it as a single system, but still save the worth of original storage. ViPR can replicate data across several different place and data center with different store product, and it provides a unified block store, object store and file system and other services. At the same time, ViPR provides a unified metadata service and self-service deployment, measurement and monitoring services.

A file system architecture for organization of data and metadata that will efficiently organize and enable sharing besides exploiting the power of storage virtualization and maintaining simplicity in such a highly complex and virtualized environment proposed by Ankur Agrrawal et al. [18].

Dejun Wang proposed an efficient cloud storage mode for heterogeneous cloud infrastructures [19]. And by extensive tests he verified the model with numerical examples. He thought that cloud storage system with traditional storage type

has some differences; such as the demand from the performance point of view, data security, reliability, efficiency and other indicators need to be considered for cloud storage services, which are services in a wide range of complex network environment for the demands of large-scale users.

here

Chapter 3

System Design and Implementation

Building a cloud system to integrate multiple storage technologies is the main goal of this work. This section introduces the system architecture and implementation which adopt the open-source software to have better development and maintenance in the future. The integrated heterogeneous storage technologies in the system are useful and complete object storage. Moreover, a graphics user interface is provided so that a manager can set some parameters to make the system more flexible.

3.1 System Architecture

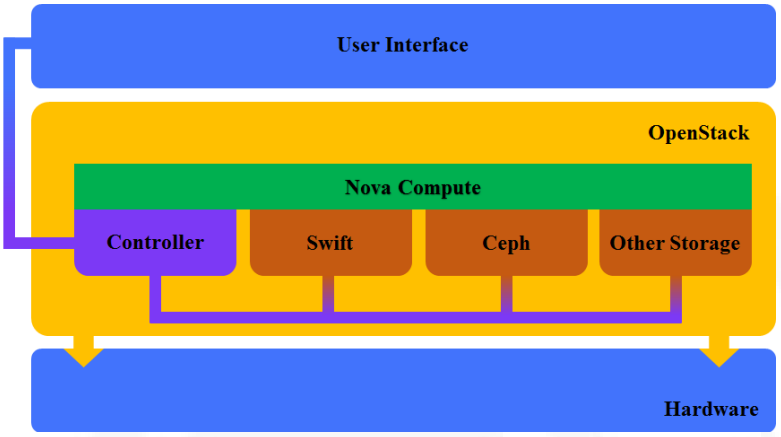


FIGURE 3.1: System Architecture

The proposed system architecture is shown in Figure 3.1. We use OpenStack as our infrastructure management, which is more efficient than hardware resources. We create virtual machines using the OpenStack to deploy our services including storage service and control service. The storage service is the basis of storage function in our system, such as Swift, Ceph and other storage functions. The control service is built on the Controller to manage the storage services. In other words, the Controller control the storage devices and resources indirectly. In addition, the Controller has its own distribution mechanism. The mechanism can automatically assign files to an appropriate storage functions after users upload files. The Controller also provide graphical user interface on web browser so that users can enjoy the proposed cloud system by web browser anytime and anywhere.

3.2 Design Flow

Our design consists of the storage service and user service. As shown in Figure 3.2. These service build on Controller. The basic of system is built by OpenStack, and OpenStack is combined with several components which are computing side, network side and storage side. We will integrate some different storage to be the

storage side in the OpenStack cloud. And the storage controller was integrated them by the API which these storage have. The concept is shown in Figure 3.2.

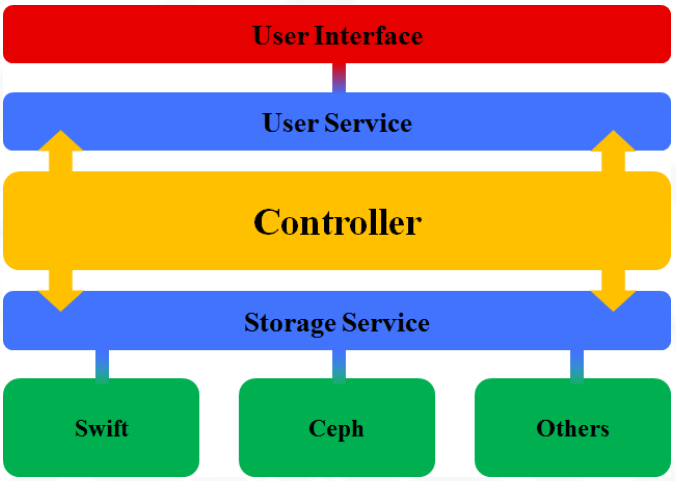


FIGURE 3.2: Controller Architecture

3.3 System Implementation

The implementation of the proposed system consists of three parts, the storage service deployment, user services and file distribution mechanism.

3.3.1 Storage Service Deployment

By using Ubuntu OS to create virtual machines, open source software OpenStack is applied to build and manage the proposed cloud system. The OpenStack architecture overview is shown in Figure. Some virtual machines are created to form a storage cluster. The overview of the system is shown in Figure 3.3 and Figure 3.4.

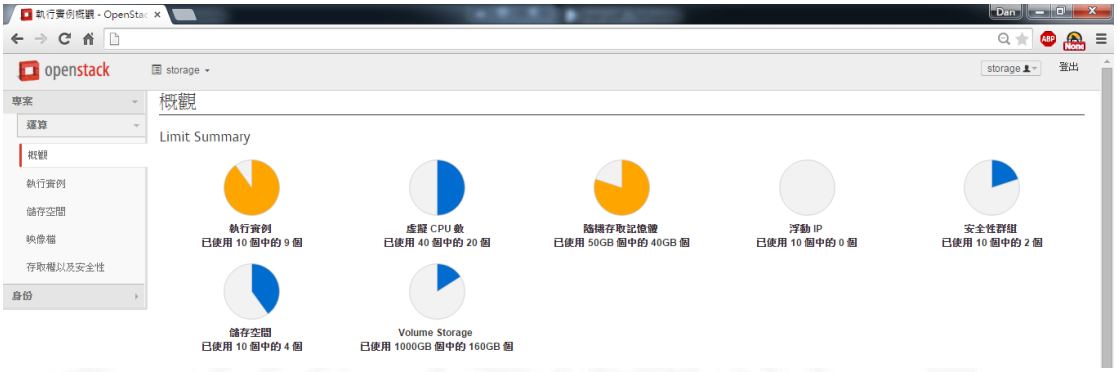


FIGURE 3.3: OpenStack Overview

執行實例名稱	映像檔名稱	IP 位址	容量	密鑰對	狀態	可用區域	任務	電源狀態
Storage-controller	Ubuntu14.04	10.0.0.10 140.128.192.12	Storage-service	-	活躍	nova	None	正在執行
ceph-a702152a-c653-4256-8f0a-510ad6d4f1a3	Ubuntu14.04	10.0.0.9	for-ceph	-	活躍	nova	None	正在執行
ceph-03275290-f3d7-43f5-9842-f5391254f66d	Ubuntu14.04	10.0.0.7	for-ceph	-	活躍	nova	None	正在執行
ceph-83a3b8e4-4600-4ba0-9d16-d029ee5d2b52	Ubuntu14.04	10.0.0.8	for-ceph	-	活躍	nova	None	正在執行
ceph-c3d41f8c-cfd9-4aa8-b8b5-af8cf1923f5b	Ubuntu14.04	10.0.0.6 140.128.192.14	for-ceph	-	活躍	nova	None	正在執行
swift-a61ea06f-d785-40e0-9804-b7b6c69437ae	Swift-ubuntu14.04	10.0.0.5	normal	-	活躍	nova	None	正在執行
swift-d1c6e47f-4c64-4bc0-9c6a-320ca1dd32de	Swift-ubuntu14.04	10.0.0.4	normal	-	活躍	nova	None	正在執行
swift-c4537398-62af-496e-b3da-9552e3cae2fd	Swift-ubuntu14.04	10.0.0.3	normal	-	活躍	nova	None	正在執行
swift-8f8a4b87-7365-42f6-bf0b-92eb7dde797a	Swift-ubuntu14.04	10.0.0.2 140.128.192.13	normal	-	活躍	nova	None	正在執行

Displaying 9 items

FIGURE 3.4: OpenStack instances

Swift deploy

Swift is an object storage service provided by OpenStack. Not all services of Swift are necessary for us. Swift services include proxy server, account server, container server and object server. The proxy server relies on an authentication and authorization mechanism such as the identity service, but proxy server also offers an internal mechanism that allows it to operate without any other OpenStack services. According to Swift deploy requirements, we need to install the following components: identity service, proxy server, account server, container server and object server. The Swift environment in our system is shown in Figure 3.5.

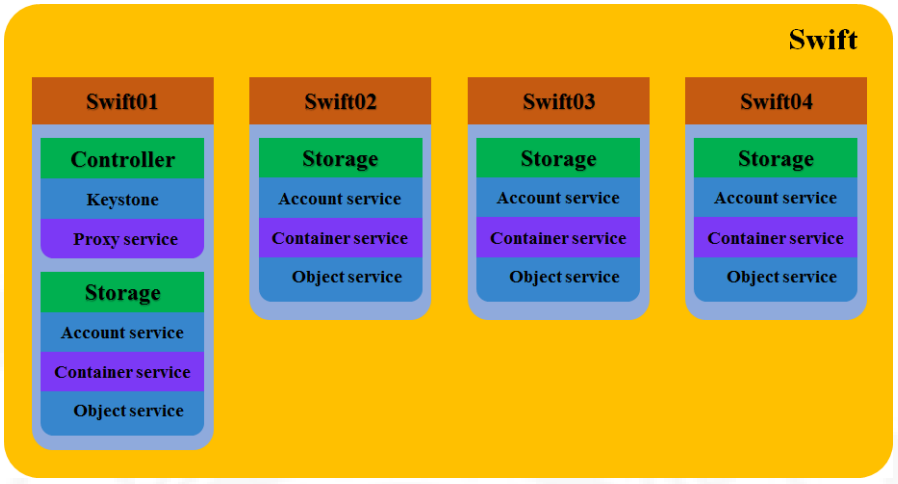


FIGURE 3.5: Swift enviroment

We use 4 virtual machines to deploy the Swift. There are one Controller node and four Storage nodes. Swift01 includes the Controller node and the Storage node. Another virtual machines only include Storage node.

Ceph deploy

Ceph is a storage service, which provides object, block, and file system storage in a single unified storage cluster. Ceph's object storage runs on the file system storage. Ceph has three main softwares: OSD, Monitors and Metadata Server.

According to Object storage deploy requirements, as shown in Figure 2.3, we only need to install OSDs and Monitors. Ceph environment in our system is shown in Figure3.6. Because our environment is not large-scale, it is enough for us to deploy one monitor on this service.

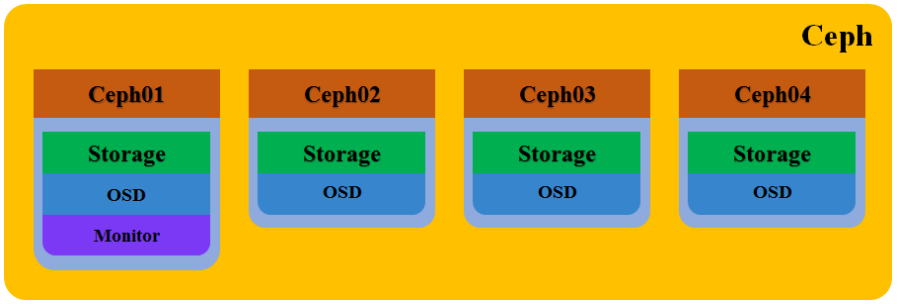


FIGURE 3.6: Ceph enviroment

3.3.2 User Service

Without loss of generality, we adopt web service as the user interface of our system. Because the web service is compatible with a variety of platforms such as PC, mobile and tablet so our client and server used JQuery and PHP language. Also, we use several techniques as shown in Table 3.1.

TABLE 3.1: Software & language Specification

Software & language	Version
PHP	5.5.9
JQuery	1.11.2
Bootstrap	3.3.4
D3.js	3.5.5
Python	2.7.6

As mentioned above, we must consider the compatibility problems. Web design in the past, the web server prepared two web pages. One is for wide screen, the other is for mobile. The index page would detect user's screen size and redirect to suitable page. Probably, there are other different screen sizes, such as tablet and many kinds of mobile. So this solution need to design more than two web pages.

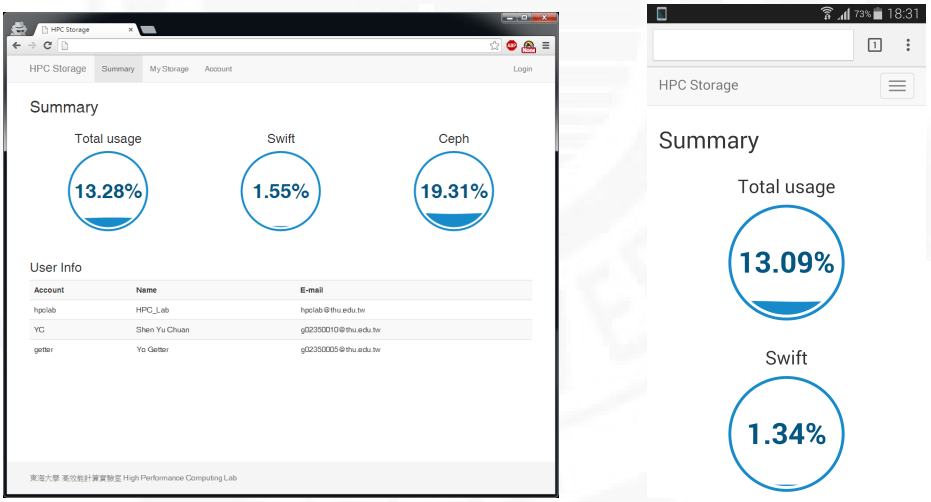
Therefore, our web interface use the Responsive Web Design. Responsive Web Design (RWD) is an approach for web design. The feature is providing an optimal viewing and interaction experience. There are some concepts in the responsive web design:

- Mobile first: Developer create a basic web site and enhance it for smart phone, rather than make a complex and image-heavy site work on mobile.
- The fluid grid concept: This concept calls for all element size and position to be relative units like percentages, rather than absolute units like pixels or points.

- Flexible images: Images need to size in relative units, so as to prevent them from displaying outside their containing element.
- Media queries: This is CSS3 module. It allow the page to use different CSS style rules based on device.

We use Bootstrap develop framework. Bootstrap is the HTML, CSS, and Javascript framework for developing responsive and mobile first projects on the web.

From the personal computer shown in Figure 3.7(a) and from the mobile shown in Figure 3.7(b). There is only one php page on server. This page will reformat the page layout according to user's screen.



(a) PC (b) Mobile

FIGURE 3.7: Responsive Web Design

The user experience is an important contribution to the development Graphical User Interface (GUI). We use PHP to build web server. After the PHP is interpreted and executed, the web server sends resulting output to its client. PHP can generate a web page's HTML code, an image, or some other data. Also the client need better interaction. JavaScript is most commonly used as part of web browsers, whose implementations allow client-side scripts to interact with the user,

control the browser, communicate asynchronously, and alter the document content that is displayed. Server-side use the PHP and client-side use the JavaScript. These makes the html page more dynamic. But PHP process when the http request. Some sql request and other contents needed update by refresh the web page. This situation is not a good interaction experience.

Making a good interaction experience not only use PHP and JavaScript, but also use the AJAX will be better. Asynchronous JavaScript and XML (AJAX) is an interrelated Web development technique used on the client-side to create asynchronous Web applications. Web applications can send data and retrieve from a server in the background without interfering with the display and behavior of the existing page. The biggest advantage of using Ajax is that we can maintain the information without updating the whole page. This makes the Web application more rapid to respond to user actions, and to avoid the transmission of information that did not change on the internet. There are some benefits in the AJAX:

- Callbacks: AJAX makes a quick process to and from the server to retrieve and save data without posting the page back to the server.
- Asynchronous: AJAX makes the web page asynchronous. The client browser to avoid waiting for all data to arrive before allowing the user to act once more.
- User friendly: Because update the web page contents is not using postback. Ajax enabled applications will always be more responsive, faster and more user friendly.

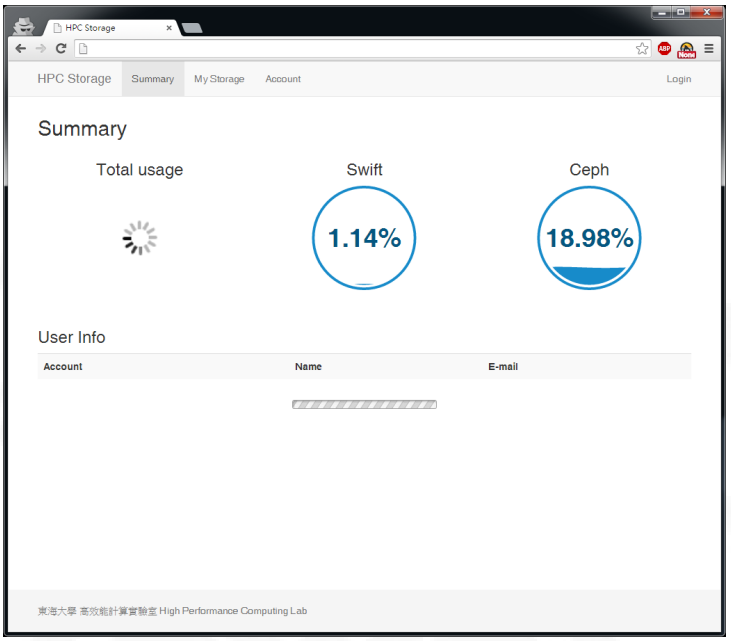


FIGURE 3.8: Asynchronous JavaScript and XML

We use a lot of AJAX technique in our system. As shown in Figure 3.8. There are four area need to display in this example, including total usage, swift usage, ceph usage and user list. We want the display is asynchronous when user open this page. Because these are from different techniques. Each techniques response time are also different. If use general solution, web server is waiting these responses then display. In figure, if the request is not retrieve the result, the web page is showing the loading bar/circle.

3.3.3 File Distribution Mechanism

We make advance storage environment measurement, measuring the size of a file transfer speeds using COSBench. The results of our tests marked on the coordinate diagram, as shown in Figure 3.9. User's file size is not fixed. Files are not the same as our measuring points. Therefore, we need a mechanism to coordinate interpolation drawing into a linear equation. We chose to use a cubic spline interpolation method.

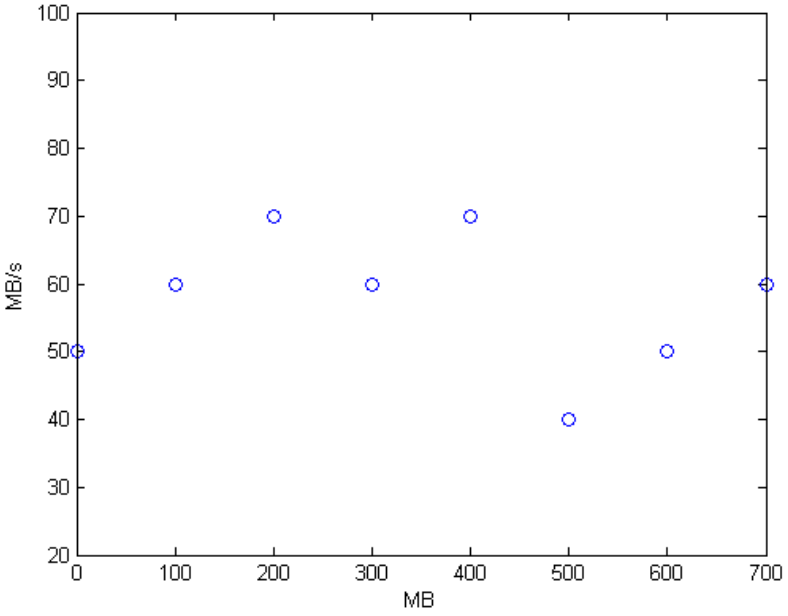


FIGURE 3.9: File transfer speeds

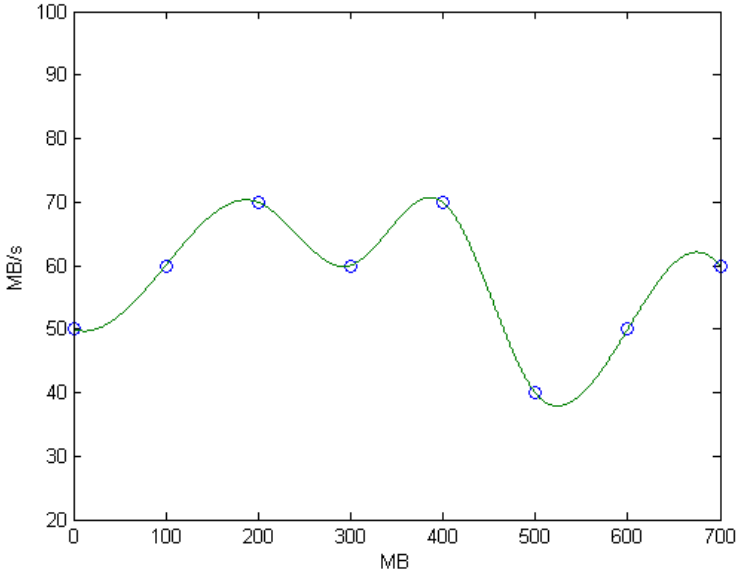


FIGURE 3.10: File transfer speeds using Cubic Spline

Therefore, we'll get a new linear coordinates diagram, as shown in Figure. Then we stacked linear plot Swift and Ceph, as shown in Figure. It can be used as decision criteria when processing files. Certainly, this will not be the only determined way in our mechanism. We also consider the use of storage capacity

for storage of the environmental effect. Like the previous measurement of the way, we do measurements at different capacity status of the storage environment.

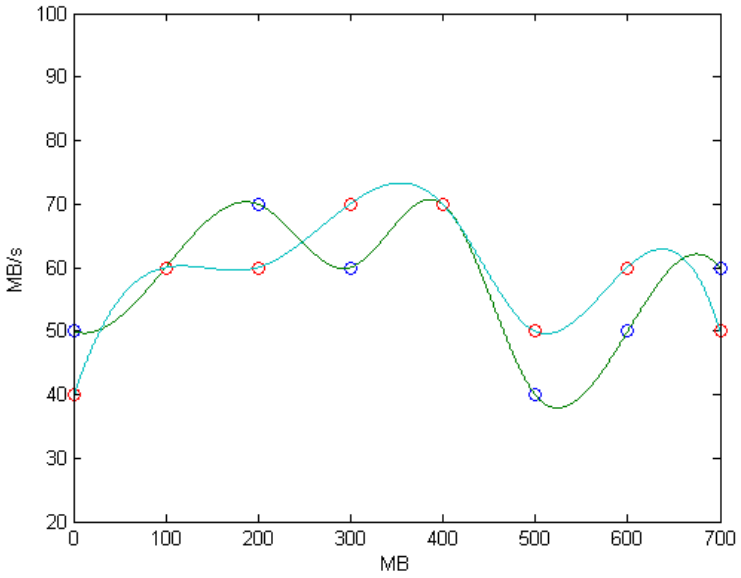


FIGURE 3.11: All file transfer speeds using Cubic Spline

We propose a function to determine which storage technologies is better, as shown in the following function 3.1.

$$f_K(S) = \alpha f_t(S) + \beta f_c(S) \tag{3.1}$$

- $f_t(S)$ is the result of the transfer speed experiment. We substitute the file size S into the function, then the transfer speed can be obtained.
- $f_c(S)$is the result of the storage capacity experiment. We substitute the file size S into the function, then the transfer speed can be obtained.
- α and β are the weights, by default 0.5. α and β must be the sum of one. We can tune the determine mechanism by these parameter.
- $f_K(S)$ is the final result. We can use the function to calculate every storage technologies. Then compare the values.

As above, we do transfer speed experiments on Swift and Ceph, then we obtained two functions, $f_{ts}(S)$ and $f_{tc}(S)$. Other experiments are storage capacity on Swift and Ceph, then we obtained two functions, $f_{cs}(S)$ and $f_{cc}(S)$. The$f_{Swift}(S)$ and $f_{Ceph}(S)$ as shows in Equation 3.2.

$$\begin{aligned} f_{Swift}(S) &= \alpha f_{ts}(S) + \beta f_{cs}(S) \\ f_{Ceph}(S) &= \alpha f_{tc}(S) + \beta f_{cc}(S) \end{aligned} \tag{3.2}$$

$$\begin{cases} f_{Swift}(S), & \text{if } (f_{Swift}(S) > f_{Ceph}(S)) \text{ or} \\ & (f_{Swift}(S) = f_{Ceph}(S) \ \& \ Usage_{Swift} > Usage_{Ceph}) \\ f_{Ceph}(S), & \text{if } (f_{Swift}(S) < f_{Ceph}(S)) \text{ or} \\ & (f_{Swift}(S) = f_{Ceph}(S) \ \& \ Usage_{Swift} < Usage_{Ceph}) \end{cases} \tag{3.3}$$

After calculating, we obtain two values from $f_{Swift}(S)$ and $f_{Ceph}(S)$. Our mechanism will compare two values and determine the storage technology. There is a special case, the two values are equal. As this case, we add the other condition depend on storage usage. The mechanism will choose a lower usage as a decision. The compare cases as shown in Equation 3.3.

Our mechanism is scalable. We can add any condition that may affect the transfer speed. As shown in Equation 3.4. The $f_n(S)$ is an other experiment. The experiment about the transfer speed. There is a control variable making the storage has a different transfer speed. The result of the experiment obtained $f_n(S)$. Like Function 3.1 , we give a weight γ. α, β and γ must be the sum of one.

$$f_K(S) = \alpha f_t(S) + \beta f_c(S) + \gamma f_n(S) \tag{3.4}$$

Chapter 4

Experiments and Results

In this chapter, we present the experiments and system implementation results. First, we test performance of the proposed system infrastructure to know the system well. Next, we measure speeds of storage systems, and this measurement provides the basis of the file distribution mechanism. Finally, we show the user interface used in the system.

4.1 Experimental Environment

We used OpenStack to build our cloud platform, which then was used to create and manage the distributed storage system. In the system we integrated two heterogeneous storage technologies. And we built the storage system by some VMs, in which Ceph was constructed by four VMs with specifications of 2-core CPU, 4 GB memory, and a total of 160 GB storage space. Ceph01 was MON and OSD, the others was OSD. These were components in the Ceph cluster. And the Swift part was constructed by four VMs including one proxy server and four storage nodes with specifications of 2-core CPU, 4 GB memory, and total of 160 GB storage space.

TABLE 4.1: Hardware Specification

Host name	CPU	Memory	Disk	OS
Openstack_Controller	16 cores CPU	48GB	100GB	Ubuntu 14.04
Openstack_compute01	24 cores CPU	48GB	800GB	Ubuntu 14.04
Openstack_compute02	24 cores CPU	48GB	800GB	Ubuntu 14.04

TABLE 4.2: Storage Environment Specification

Host name	CPU	Memory	Disk	OS
Controller	4 cores vCPU	8GB	40GB	Ubuntu 14.04
ceph01	2 cores vCPU	8GB	40GB	Ubuntu 14.04
ceph02	2 cores vCPU	4GB	40GB	Ubuntu 14.04
ceph03	2 cores vCPU	4GB	40GB	Ubuntu 14.04
ceph04	2 cores vCPU	4GB	40GB	Ubuntu 14.04
swift01	2 cores vCPU	4GB	40GB	Ubuntu 14.04
swift02	2 cores vCPU	4GB	40GB	Ubuntu 14.04
swift03	2 cores vCPU	4GB	40GB	Ubuntu 14.04
swift04	2 cores vCPU	4GB	40GB	Ubuntu 14.04

TABLE 4.3: Software Specification

Software	Version
OpenStack	Juno
Ceph	Hammer v0.94
Swift	2.1.0

4.2 Performance

We need to obtain baseline performance statistics for the two main components of our infrastructure: network and disks. We chose four VMs as the experimental nodes on OpenStack. The four VMs were swift01, swift02, swift03 and swift04.

First, network throughput is a key factor affecting the cluster performance. A good tool for this is iperf, which uses a client-server connection to measure TCP and UDP bandwidths. The results are shown in Figure 4.1. In the histogram, the x-axis is the number of tests, and the y-axis is the transmission bandwidth. The results are divided into two groups: group A consisting of Swift01 and Swift03, and group B consisting of Swift02 and Swift04. The bandwidth of Group A is almost about 7,000 Mbits/s, and the bandwidth of Group B is almost about 900 Mbits/s. After analysis, we think the reason of the results is due to different physical hosts used by the two groups. Swift01 and Swift03 are on Openstack_compute01. Swift02 and Swift04 are on Openstack_compute02. VMs use the physical network to communicate between two physical hosts; whereas, VMs use the virtual network to communicate within the internal of a host. This experiment clearly shows that various network scenarios can be used for the distribution of VMs on OpenStack.

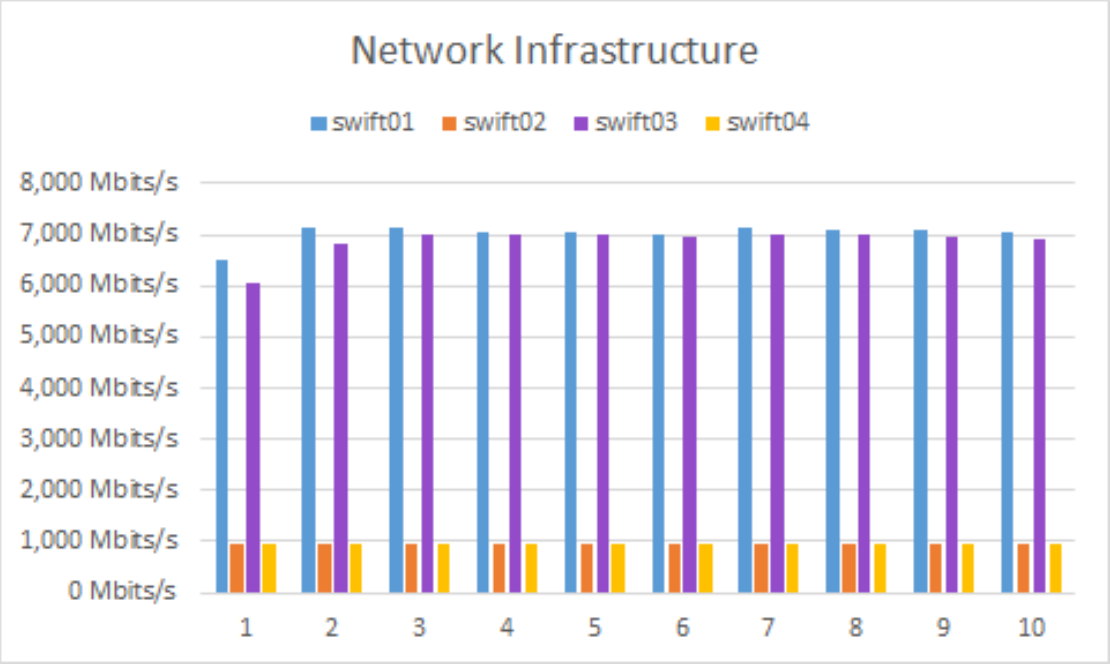


FIGURE 4.1: Network infrastructure speed

Another key factor that affects the system performance is disks. The simple way to measure the performance of a disk is with linux command dd, which is used to convert and copy files. The results of disk read and disk write are shown in Figure 4.3 and 4.2, respectively.

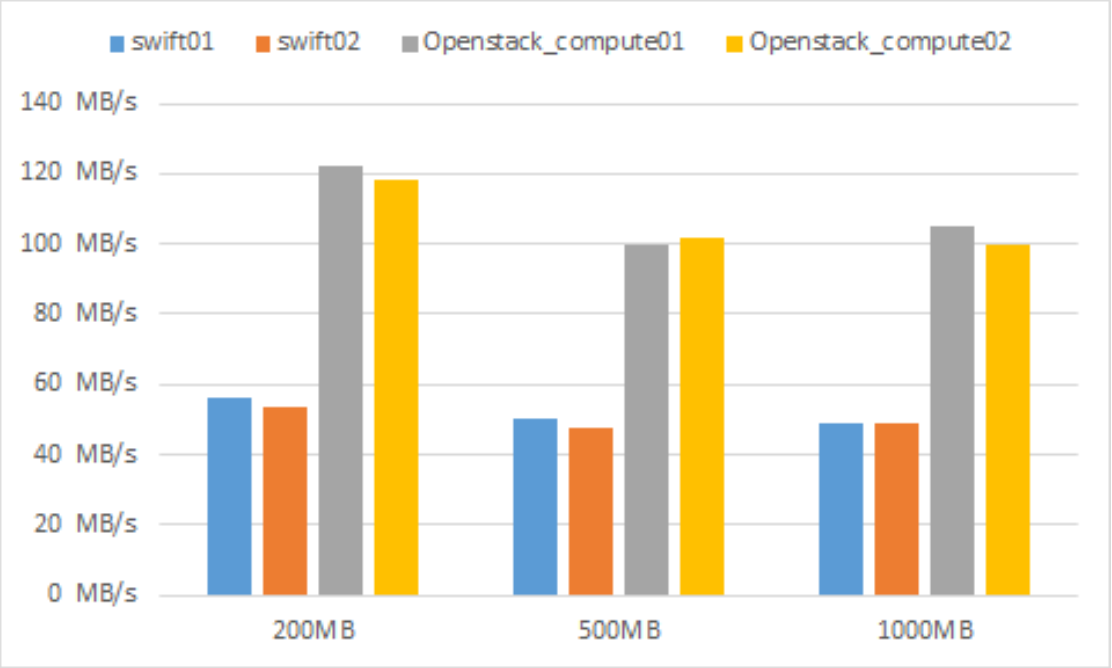


FIGURE 4.2: Disk infrastructure write speed

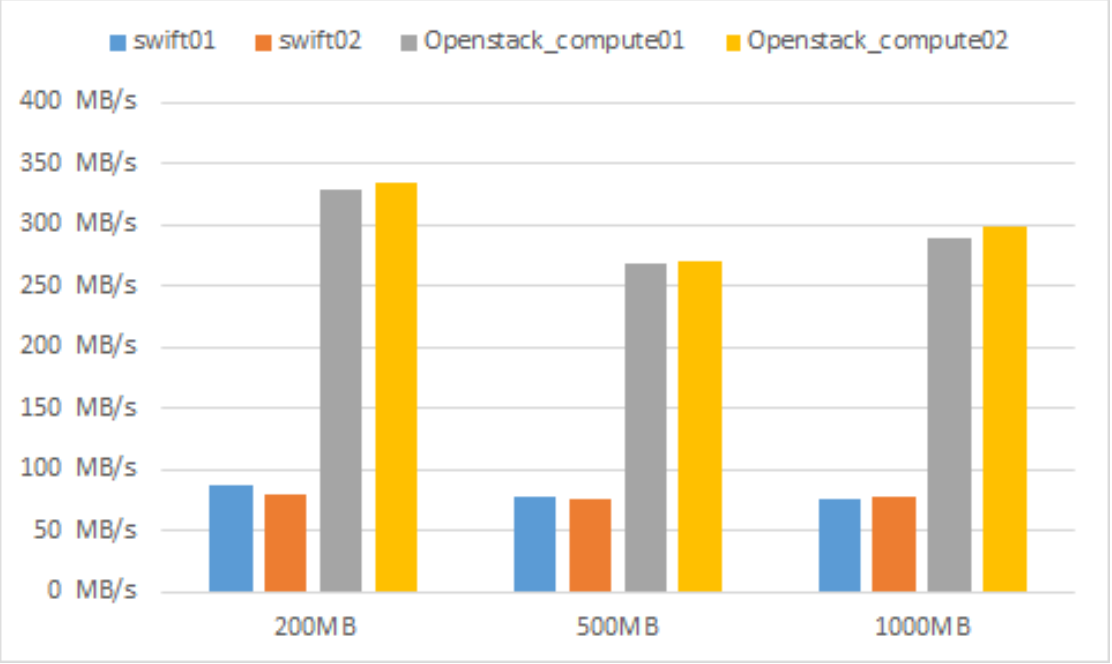


FIGURE 4.3: Disk infrastructure read speed

According to the network experiments, for experiment settings with VMs on the same host, the results are similar. Consequently, the experiment was designed to measurement swift01, swift02, Openstack_compute01 and Openstack_compute02. The write and read results show that the VM cannot use the full speed of the host. This analysis provides the basis for us to deploy the storage system. In the later section of debugging and improving bottlenecks these I/O tests are needed to clarify the problems.

At above two experiments, we have decided the number of VMs on the host and how to deploy the storage cluster. And next, we measure Ceph and Swift storage clusters.

The results of upload speeds are marked on the diagram. The blue circle is Swift, and red one is Ceph. Then the cubic spline is used to get the linear curve, as shown in Figure 4.4. The speed of Swift is stabilized at 20-30MB/s, but its speed increases when the file size is more than 800MB. The overall curve of speed of Ceph is stabilized at 15MB. The two curves intersect at one point, located about 50MB. The speed of Ceph is faster than that of Swift when the file size is less than 50MB. On the other hand, the speed of Swift is faster than that of Ceph when the file size is more 50MB.

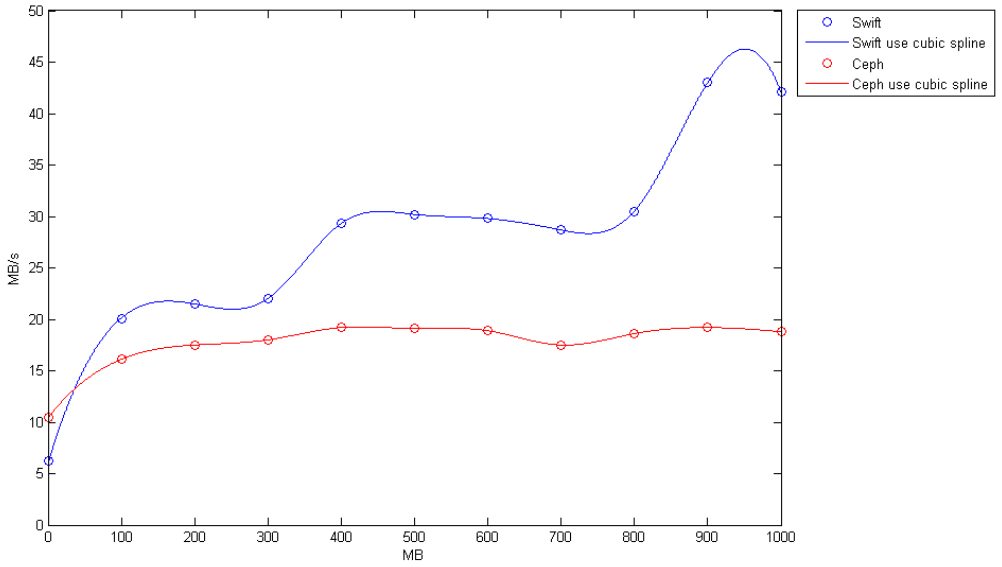


FIGURE 4.4: Measure the upload speed

The other experiment is download measurement. As for the upload measurement, we draw the results in a diagram, as shown in Figure 4.5. We observe that the download speed of Ceph is faster than that of Swift. The average difference between them is about 10MB.

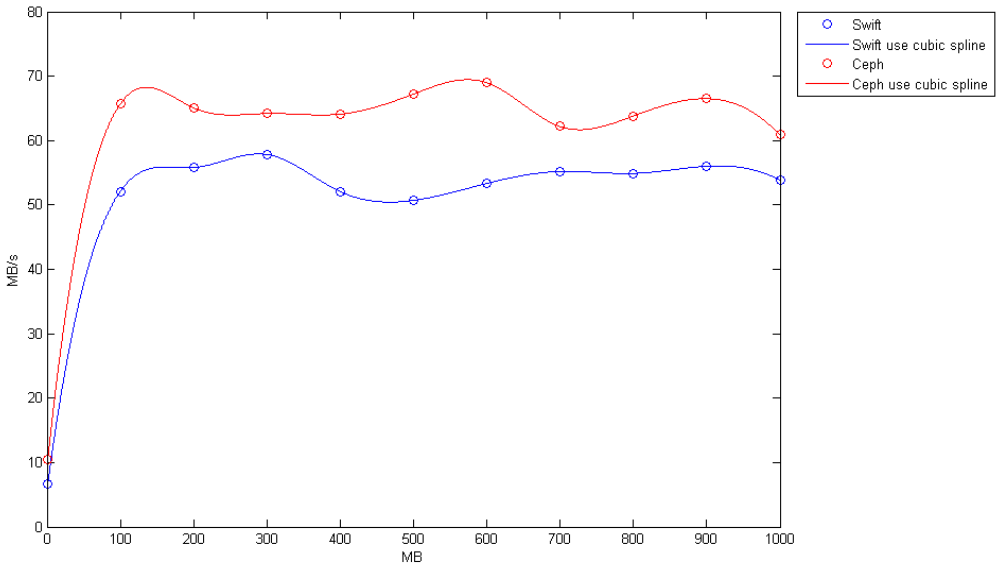


FIGURE 4.5: Measure the download speed

In this work, we measure the storage cluster. And these results as the mechanism to allocate the file store. That mechanism bring each storage advantage in our system.

4.3 User Interface

We used a user interface in our system. First, we overview the website map, as shown in Figure 4.6. There are three parts in this interface. *Overview*, *My storage* and *Account* each is a main page with several functions. *Overview* provides the summary of the user status. Users can view their storage usage and account list. For the user, *My storage* is the major part in this system, and it includes following basic operations: upload, download, remove, and modify. *Account* shows the user information, in which the user can also edit the user account, such as username, password and E-mail.

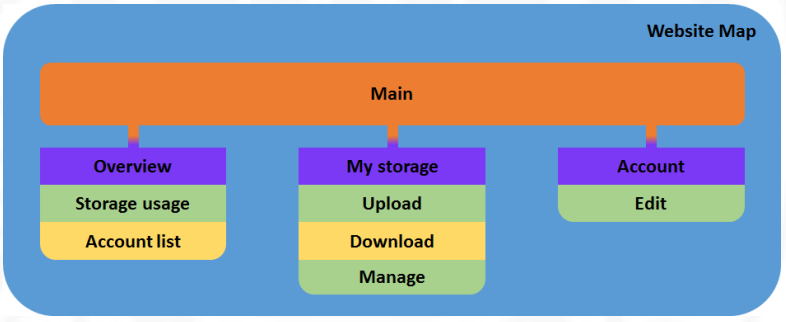
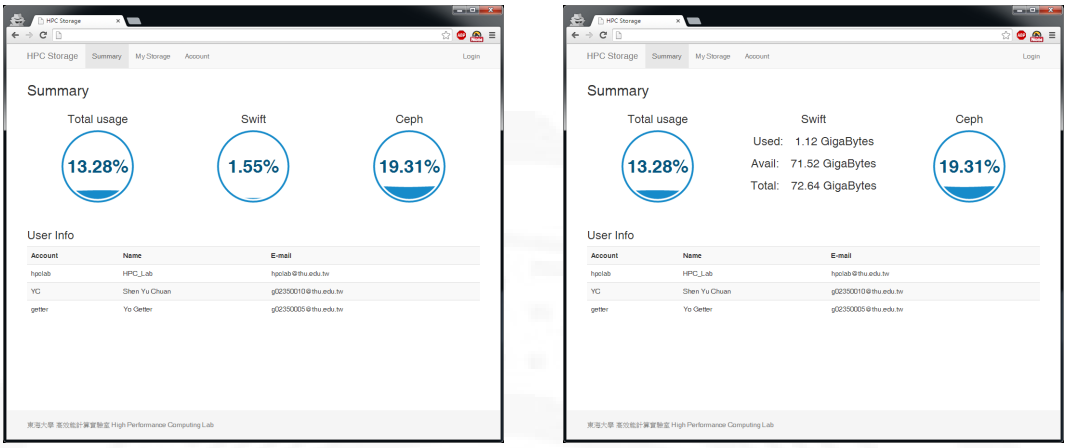


FIGURE 4.6: User interface overview

As shown in Figure 4.7(a), there are two panels in *Overview*. These two panels are the storage usage and account list, respectively. In the current supported storage technology, we have three liquid fill gauges. These liquid fill gauges display usage percentages, including total usage, Swift usage and Ceph usage. The storage usage will change from the liquid fill gauge to usage details when moving the mouse over the liquid fill gauge, as shown in Figure 4.7(b). This use of javascript is to achieve Web2.0. The table at the bottom of the webpage is the account list, which lists all account information. It is shown when the user logs in as an administrator.



(a) (b)

FIGURE 4.7: Overviwe page

The *My storage* page is the major operating page in our system. Upon entering this page, there is a file list in the middle. The drop down menu is shown when the user right clicks the file, as shown in Figure 4.8. The drop down menu has four functions: download, delete, rename, and detail. These four functions plus the upload function compose of all the storage operating functions in this page.

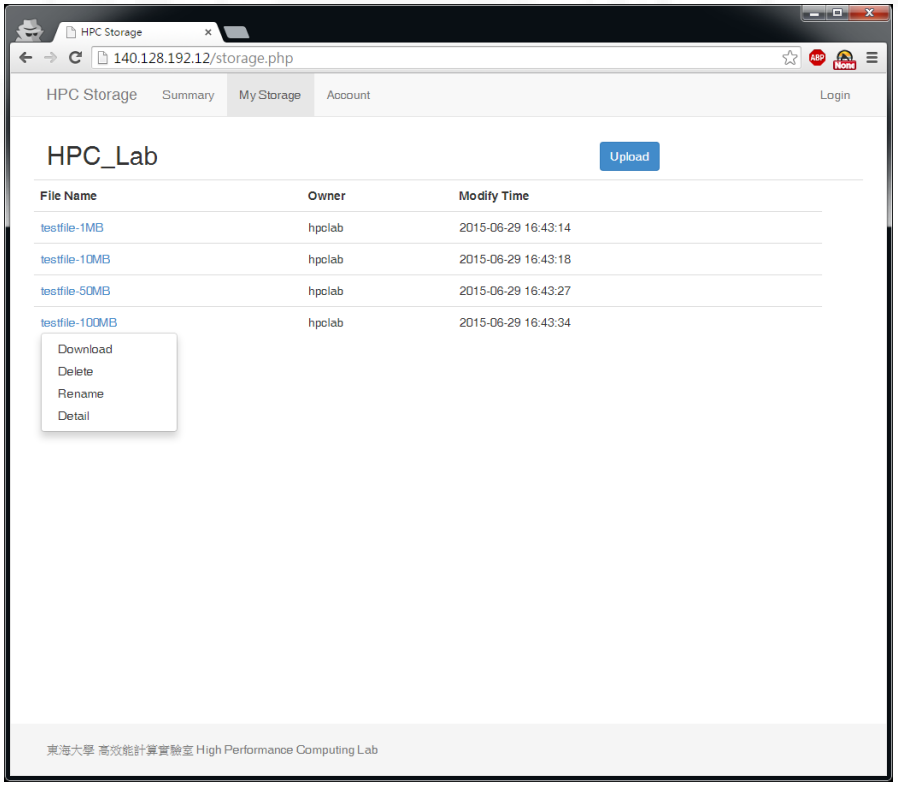


FIGURE 4.8: My storage page view

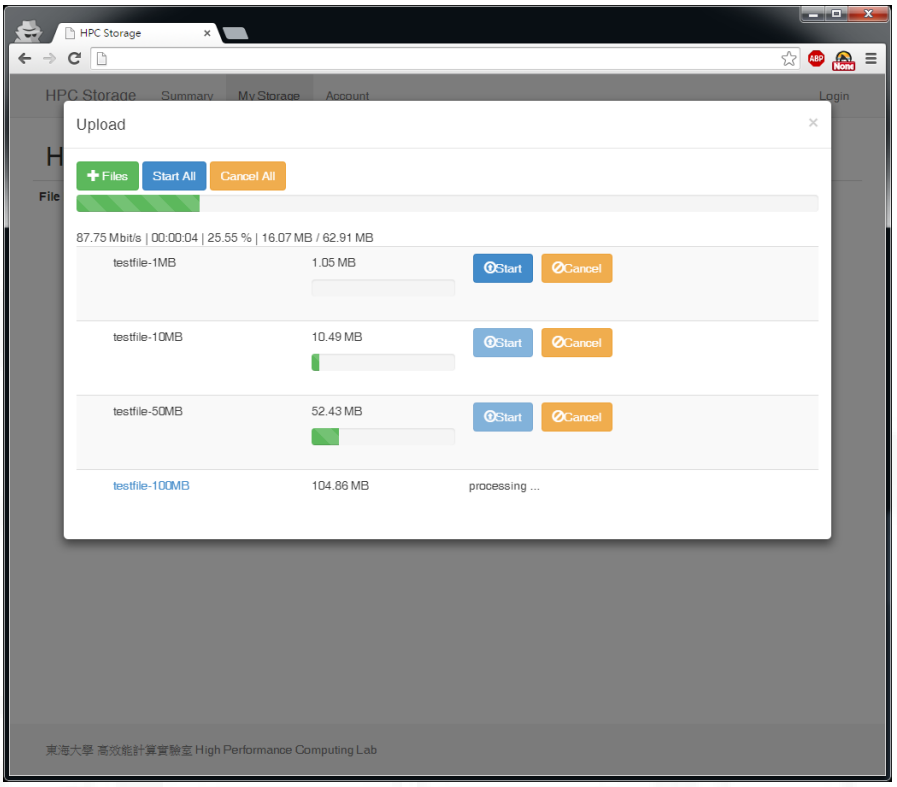


FIGURE 4.9: Upload the files

We use the AJAX, JQuery and Bootstrap to implement the upload function. The web page popups a window when the upload button is clicked, as shown in Figure 4.9. This design does not redirect the page to the upload function page and the upload process works like a background process. There are four files shown in the list. One is ready to upload, the other two are being uploaded, and the last one is under processing. The upload function allows multiple files uploaded at the same time. They are shown in individual progress bars and the total progress bar. The total progress bar shows the upload details, including the transfer speed, remaining time and completed percentage to let user monitor and control the upload progress. The processing state is entered when the upload is finished and the server processes files to storage. By clicking outside of the upload window, the window will be closed and the upload job will proceed in the background. When clickingthe upload button again, the window shows the current upload progress. Several features of the upload function are listed as follows:

- Friendly user interface: The upload progress is visually shown; thus, the user can easily monitor and control the upload jobs.
- Multi-upload: The user can upload multiple files at the same time.
- Background processing: The user can upload files in background and operate other functions at the same time in the *My storage* page.

The other function is Detail, which displays the file details, as shown in Figure 4.10. We use a design similar to the upload function. When clicking on the detail button, the detail web page will pop up. It also includes download and delete functions, so the user not only views the details but also manages files.

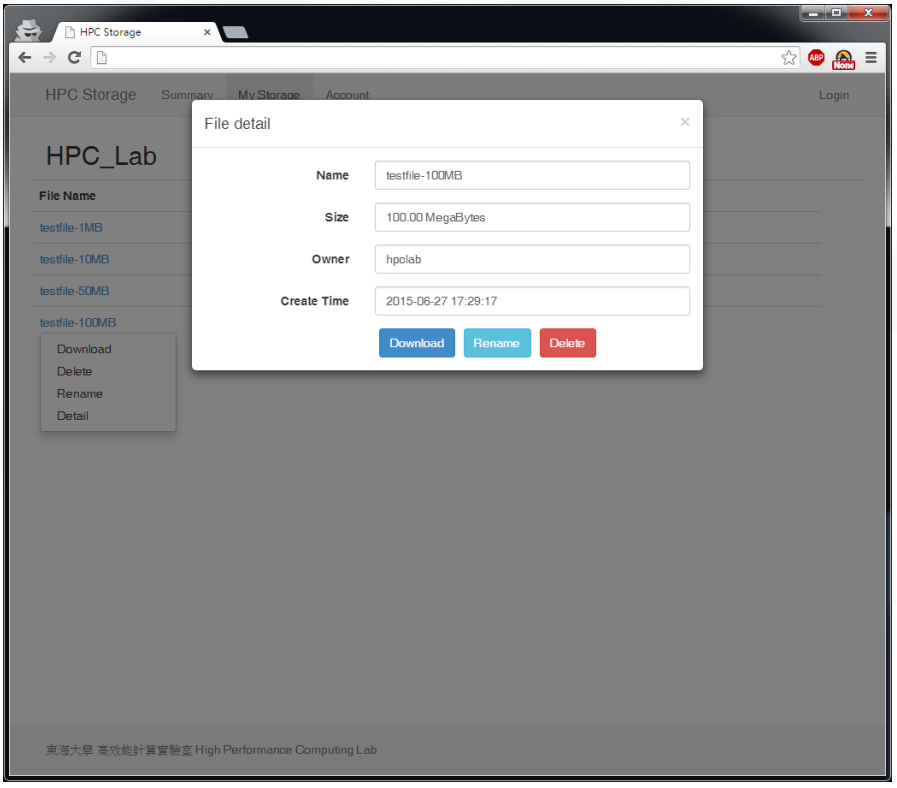


FIGURE 4.10: File details

The last part is *Account*, as shown in Figure 4.11. The *Account* page has two main functions, i.e., viewing and editing the account details. Certainly, the design of it also follows concepts of responsive web design, asynchronous JavaScript, and XML.

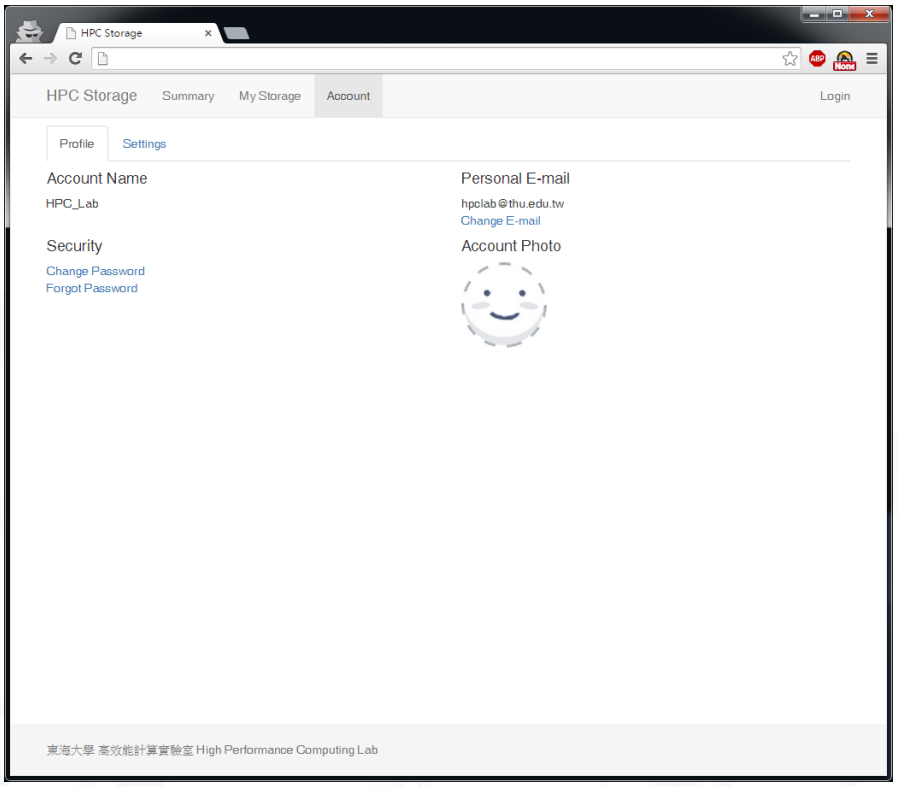
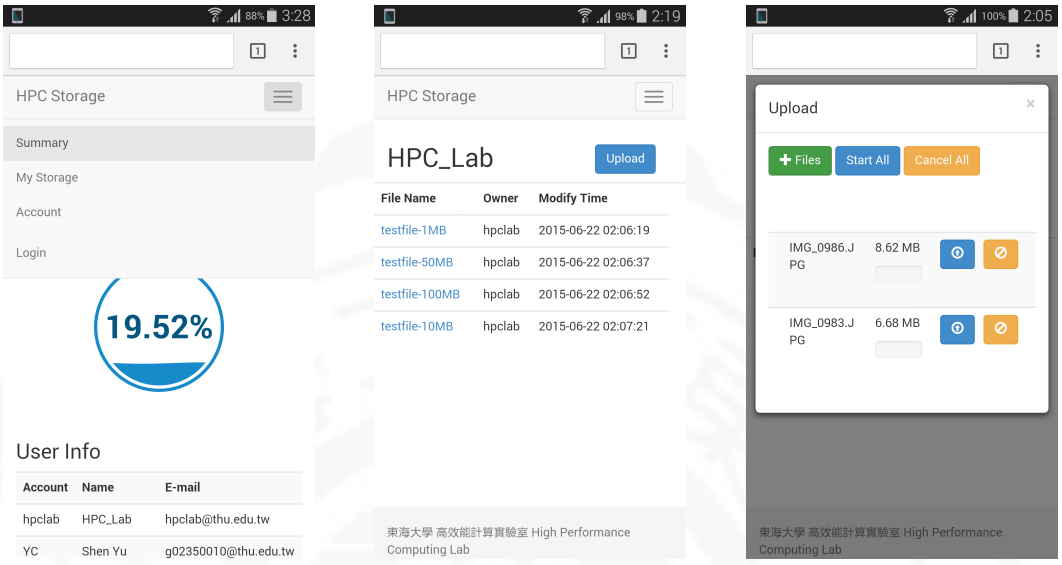


FIGURE 4.11: Account page view

About the responsive web design, all pages of this work follows this design concept. Whatever any device is used, boot-starting the screen size to responsively display a suitable web layout.

As shown in Figure 4.12, navigation bar at the top has been changed. The mobile screen is in a narrow shape; however the banner navigation bar cannot be displayed in one row. Therefore, navigation bar converts to a drop down menu, as shown in Figure 4.12(a). That makes the navigation bar looked better on the narrow screen. The other features are shown in Figure 4.12(c). For the same reason that the mobile screen is narrow, some text is too long to be displayed. The button texts, i.e., Upload and Cancel, replace symbol icons. As described above, we not only reformat the layout also change some content to make the narrow screen looked not too complex.



(a) (b) (c)

FIGURE 4.12: Responsive Web Design

About the other technique, asynchronous JavaScript and XML used in our system, we used the JQuery to achieve the AJAX. We applied it in many functions. One of the applications is for information display. Massive data are obtained through inquiry database and storage systems. The information response time is different. When retrieving the information, we want to display it in real time. There are several data panels. The data panel needs to finish its inquiry before it will update the content. Consequently, every panel does not wait for each other. In our work, other applications such as upload also use the AJAX concept.

Chapter 5

Conclusions and Future Work

This thesis builds a cloud system to achieve the SDS concept. In the cloud system, we use distributed cloud architecture integrated several storage technologies to provide high reliability and high scalability cloud services. Finally, we design a high usability user interface to have a more friendly system. The following are our concluding remark and future work.

5.1 Concluding Remark

We propose a mechanism which works according to the transfer speed of each storage. This mechanism allows heterogeneous storage. Even different clusters using same technology storage can be integrated in our system. For example, we can integrate the two different Swift clusters. The mechanism can allocate files to suitable storage spaces. And this mechanism has scalability; Besides, we can add more factors to customize the mechanism and make it more intelligent.

We also provide a high usability user interface. The user interface is designed as a web application. According to the concept of cloud services, this interface can be used anywhere and anytime. We use a responsive web design to achieve features of the web application. Consequently, it not only achieving anywhere and anytime features but also it can be used in any device with a web browser. Whatever the

user's device screen is, the responsive web design can style the page in a beautiful and useful layout. The user interface design focuses on the user experience; thus, the smooth operation is very important. To improve the user experience, we use asynchronous JavaScript and XML, enabling the web application to directly update its content without redirecting and/or refreshing the web page.

5.2 Future Works

The proposed storage is not large since we only build less than 10 VMs. So we plan to build a larger system to test our method in the future. Similarly, it is possible to integrate more storage technologies into the system although our current system can integrate several heterogeneous storage technologies. The other two future works are described as follows. First, although our system supports open source technologies Swift and Ceph, it is necessary to support the current cloud providers, such as Amazon S3 and Windows Azure. Second, the mechanism runs on a web server. We want to detach the mechanism and develop a service which can be called by other servers. Consequently, we need to develop APIs to provide other servers to use our system.

References

[1] I. Foster, Yong Zhao, I. Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1–10, Nov 2008.

[2] D. Nurmi, R. Wolski, C. Grzegorczyk, G. Obertelli, S. Soman, L. Youseff, and D. Zagorodnov. The eucalyptus open-source cloud-computing system. In *Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on*, pages 124–131, May 2009.

[3] Mahadev Satyanarayanan, P. Bahl, R Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *Pervasive Computing, IEEE*, 8(4): 14–23, Oct 2009.

[4] R. Buyya, Chee Shin Yeo, and S. Venugopal. Market-oriented cloud computing: Vision, hype, and reality for delivering it services as computing utilities. In *High Performance Computing and Communications, 2008. HPCC '08. 10th IEEE International Conference on*, pages 5–13, Sept 2008.

[5] G. Vernik, A. Shulman-Peleg, S. Dippl, C. Formisano, M.C. Jaeger, E.K. Kolodner, and M. Villari. Data on-boarding in federated storage clouds. In *Cloud Computing (CLOUD), 2013 IEEE Sixth International Conference on*, pages 244–251, June 2013.

[6] E.K. Kolodner, S. Tal, D. Kyriazis, D. Naor, M. Allalouf, L. Bonelli, P. Brand, A. Eckert, E. Elmroth, S.V. Gogouvitis, D. Harnik, F. Hernandez, M.C. Jaeger, E.B. Lakew, J.M. Lopez, M. Lorenz, A. Messina, A. Shulman-Peleg,

R. Talyansky, A. Voulodimos, and Y. Wolfsthal. A cloud environment for data-intensive storage services. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 357–366, Nov 2011.

[7] S. Rhea, C. Wells, P. Eaton, D. Geels, B. Zhao, H. Weatherspoon, and J. Kubiatowicz. Maintenance-free global data storage. *Internet Computing, IEEE*, 5(5):40–49, Sep 2001.

[8] M. Mesnier, G.R. Ganger, and E. Riedel. Object-based storage. *Communications Magazine, IEEE*, 41(8):84–90, Aug 2003.

[9] Simon Robinson. Software-defined storage: The reality beneath the hype. `http://www.computerweekly.com/opinion/Software-defined-storage-The-reality-beneath-the-hype`, 2013.

[10] Inc Coraid. The fundamentals of software-defined storage. `http://san.coraid.com/rs/coraid/images/SB-Coraid_SoftwareDefinedStorage.pdf`, 2013.

[11] Margaret Rouse. software-defined storage. `http://searchsdn.techtarget.com/definition/software-defined-storage`, 2013.

[12] Openstack swift. `https://wiki.openstack.org/wiki/Swift`, 2015.

[13] Sage A Weil, Scott A Brandt, Ethan L Miller, Darrell DE Long, and Carlos Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th symposium on Operating systems design and implementation*, pages 307–320. USENIX Association, 2006.

[14] Openstack. `https://www.openstack.org/`, 2015.

[15] Qing Zheng, Haopeng Chen, Yaguang Wang, Jian Zhang, and Jiangang Duan. Cosbench: Cloud object storage benchmark. In *4th ACM/SPEC International Conference on Performance Engineering (ICPE 2013)*. ACM, 2013.

[16] Chao-Tung Yang, Wei-Hsiang Lien, Yu-Chuan Shen, and Fang-Yi Leu. Implementation of a software-defined storage service with heterogeneous storage technologies. In *Advanced Information Networking and Applications Workshops (WAINA), 2015 IEEE 29th International Conference on*, pages 102–107. IEEE, March 2015.

[17] Emc vipr. `http://www.emc.com/vipr`, 2015.

[18] Ankur Agrrawal, Rahul Shankar, Shagun Akarsh, and Piyush Madan. File system aware storage virtualization management. In *2012 IEEE International Conference on Cloud Computing in Emerging Markets (CCEM)*, pages 1–11. IEEE, 2012.

[19] Dejun Wang. An efficient cloud storage model for heterogeneous cloud infrastructures. *Procedia Engineering*, 23:510–515, 2011.

[20] wikipedia. Software-defined storage. `http://en.wikipedia.org/wiki/Software-defined_storage`, 2014.

[21] Mark Carlson, Alan Yoder, and Leah Schoeb. Software defined storage. `http://snia.org/sites/default/files/SNIA%20Software%20Defined%20Storage%20White%20Paper-%20v1.0k-DRAFT.pdf/`, 2014.

[22] Tahani Hussain, Paulvanna Nayaki Marimuthu, and Sami J. Habib. Managing distributed storage system through network redesign. In *Asia-Pacific Network Operations and Management Symposium*, pages 1–6, 2013.

[23] Chengzhang Peng and Zejun Jiang. Building a cloud storage service system. *Procedia Environmental Sciences*, 10, Part A(0):691 – 696, 2011. 2011 3rd International Conference on Environmental Science and Information Application Technology {ESIAT} 2011.

[24] Patrícia Takako Endo, Glauco Estácio Gonçalves, Judith Kelner, and Djamel Sadok. A survey on open-source cloud computing solutions. In *Brazilian Symposium on Computer Networks and Distributed Systems*, 2010.

[25] wikipedia. Cloud computing. `http://en.wikipedia.org/wiki/Cloud_computing`, 2013.

[26] Sergi Figuerola, Mathieu Lemay, Victor Reijs, Michel Savoie, and Bill St. Arnaud. Converged optical network infrastructures in support of future internet and grid services using iaas to reduce ghg emissions. *J. Lightwave Technol.*, 27(12):1941–1946, Jun 2009.

[27] Josef Spillner, Johannes Müller, and Alexander Schill. Creating optimal cloud storage systems. *Future Generation Comp. Syst.*, 29(4):1062–1072, 2013.

[28] Chao-Tung Yang, Wen-Chung Shih, and Chih-Lin Huang. Implementation of a distributed data storage system with resource monitoring on cloud computing. In *GPC*, pages 64–73, 2012.

[29] Chao-Tung Yang, Wen-Chung Shih, Guan-Han Chen, and Shih-Chi Yu. Implementation of a cloud computing environment for hiding huge amounts of data. In *International Symposium on Parallel and Distributed Processing with Applications*, pages 1–7, 2010.

[30] Borja Sotomayor, Rubén S Montero, Ignacio M Llorente, and Ian Foster. Virtual infrastructure management in private and hybrid clouds. *Internet Computing, IEEE*, 13(5):14–22, 2009.

[31] Rafael Moreno-Vozmediano, Ruben S. Montero, and Ignacio M. Llorente. Elastic management of cluster-based services in the cloud. In *Proceedings of the 1st workshop on Automated control for datacenters and clouds*, ACDC '09, pages 19–24, New York, NY, USA, 2009. ACM.

Appendix A

OpenStack Installation

I. OpenStack packages

```
# apt-get install ubuntu-cloud-keyring
# echo "deb http://ubuntu-cloud.archive.canonical.com/ubuntu" \
  "trusty-updates/juno main" > /etc/apt/sources.list.d/cloudarchive-juno.list
# apt-get update && apt-get dist-upgrade
```

II. Database server installation

```
# apt-get install mariadb-server python-mysqldb
# vim /etc/mysql/my.cnf

        [mysqld]
        ...
        bind-address = 10.0.0.11
        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
```

III. Messaging server installation

```
# apt-get install rabbitmq-server
# rabbitmqctl change_password guest RABBIT_PASS
# service rabbitmq-server restart
```

IV. Add the Identity service

```
# 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';

# apt-get install keystone python-keystoneclient
# vim /etc/keystone/keystone.conf

        [DEFAULT]
        ...
        admin_token = ADMIN_TOKEN

        [database]
        ...
        connection = mysql://keystone:KEYSTONE_DBPASS@controller/keystone

        [token]
        ...
        provider = keystone.token.providers.uuid.Provider
        driver = keystone.token.persistence.backends.sql.Token

        [revoke]
        ...
        driver = keystone.contrib.revoke.backends.sql.Revoke

        [DEFAULT]
        ...
        verbose = True

# su -s /bin/sh -c "keystone-manage db_sync" keystone
# service keystone restart
# rm -f /var/lib/keystone/keystone.db
# (crontab -l -u keystone 2>&1 | grep -q token_flush) || \
  echo '@hourly /usr/bin/keystone-manage token_flush > \
  /var/log/keystone/keystone-tokenflush.log 2>&1' >> /var/spool/cron/crontabs/keystone

 # export OS_SERVICE_TOKEN=ADMIN_TOKEN
```

```
# export OS_SERVICE_ENDPOINT=http://controller:35357/v2.0

# keystone tenant-create --name admin --description "Admin Tenant"
# keystone user-create --name admin --pass ADMIN_PASS --email EMAIL_ADDRESS
# keystone role-create --name admin
# keystone user-role-add --user admin --tenant admin --role admin

# keystone service-create --name keystone --type identity \
  --description "OpenStack Identity"
# keystone endpoint-create \
  --service-id $(keystone service-list | awk '/ identity / {print $2}') \
  --publicurl http://controller:5000/v2.0 \
  --internalurl http://controller:5000/v2.0 \
  --adminurl http://controller:35357/v2.0 \
  --region regionOne
```

V. Add the Image Service

```
# 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';

# keystone user-create --name glance --pass GLANCE_PASS
# keystone user-role-add --user glance --tenant service --role admin
# keystone service-create --name glance --type image \
  --description "OpenStack Image Service"
# keystone endpoint-create \
  --service-id $(keystone service-list | awk '/ image / {print $2}') \
  --publicurl http://controller:9292 \
  --internalurl http://controller:9292 \
  --adminurl http://controller:9292 \
  --region regionOne

# apt-get install glance python-glanceclient
# vim /etc/glance/glance-api.conf
        [database]
        ...
        connection = mysql://glance:GLANCE_DBPASS@controller/glance

        [keystone_authtoken]
        ...
        auth_uri = http://controller:5000/v2.0
        identity_uri = http://controller:35357
```

```
        admin_tenant_name = service
        admin_user = glance
        admin_password = GLANCE_PASS

        [paste_deploy]
        ...
        flavor = keystone

        [glance_store]
        ...
        default_store = file
        filesystem_store_datadir = /var/lib/glance/images/

        [DEFAULT]
        ...
        notification_driver = noop
        verbose = True

# vim /etc/glance/glance-registry.conf
        [database]
        ...
        connection = mysql://glance:GLANCE_DBPASS@controller/glance

        [keystone_authtoken]
        ...
        auth_uri = http://controller:5000/v2.0
        identity_uri = http://controller:35357
        admin_tenant_name = service
        admin_user = glance
        admin_password = GLANCE_PASS

        [paste_deploy]
        ...
        flavor = keystone

        [DEFAULT]
        ...
        notification_driver = noop
        verbose = True

# su -s /bin/sh -c "glance-manage db_sync" glance
# service glance-registry restart
# service glance-api restart
# rm -f /var/lib/glance/glance.sqlite
```

VI. Add the Compute service (controller node)

```
# 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';

# keystone user-create --name nova --pass NOVA_PASS
# keystone user-role-add --user nova --tenant service --role admin
# keystone service-create --name nova --type compute \
  --description "OpenStack Compute"
# keystone endpoint-create \
  --service-id $(keystone service-list | awk '/ compute / {print $2}') \
  --publicurl http://controller:8774/v2/%\(tenant_id\)s \
  --internalurl http://controller:8774/v2/%\(tenant_id\)s \
  --adminurl http://controller:8774/v2/%\(tenant_id\)s \
  --region regionOne

# apt-get install nova-api nova-cert nova-conductor nova-consoleauth \
  nova-novncproxy nova-scheduler python-novaclient
# vim /etc/nova/nova.conf
        [database]
        ...
        connection = mysql://nova:NOVA_DBPASS@controller/nova

        [DEFAULT]
        ...
        rpc_backend = rabbit
        rabbit_host = controller
        rabbit_password = RABBIT_PASS
        auth_strategy = keystone
        my_ip = 10.0.0.11
        vncserver_listen = 10.0.0.11
        vncserver_proxyclient_address = 10.0.0.11
        verbose = True

        [keystone_authtoken]
        ...
        auth_uri = http://controller:5000/v2.0
        identity_uri = http://controller:35357
        admin_tenant_name = service
        admin_user = nova
        admin_password = NOVA_PASS

        [glance]
```

```
        ...
        host = controller

# su -s /bin/sh -c "nova-manage db sync" nova
# service nova-api restart
# service nova-cert restart
# service nova-consoleauth restart
# service nova-scheduler restart
# service nova-conductor restart
# service nova-novncproxy restart
# rm -f /var/lib/nova/nova.sqlite
```

VII. Add the Compute service (compute node)

```
# apt-get install nova-compute sysfsutils
# vim /etc/nova/nova.conf
        [DEFAULT]
        ...
        rpc_backend = rabbit
        rabbit_host = controller
        rabbit_password = RABBIT_PASS
        auth_strategy = keystone
        my_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
        vnc_enabled = True
        vncserver_listen = 0.0.0.0
        vncserver_proxyclient_address = MANAGEMENT_INTERFACE_IP_ADDRESS
        novncproxy_base_url = http://controller:6080/vnc_auto.html
        verbose = True

        [keystone_authtoken]
        ...
        auth_uri = http://controller:5000/v2.0
        identity_uri = http://controller:35357
        admin_tenant_name = service
        admin_user = nova
        admin_password = NOVA_PASS

        [glance]
        ...
        host = controller

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

VIII. Add a networking (contoller node)

```
# vim /etc/nova/nova.conf
        [DEFAULT]
        ...
        network_api_class = nova.network.api.API
        security_group_api = nova

# service nova-api restart
# service nova-scheduler restart
# service nova-conductor restart
```

IX. Add a networking (compute node)

```
# apt-get install nova-network nova-api-metadata
# vim /etc/nova/nova.conf
        [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
```

X. Add the dashboard

```
# apt-get install openstack-dashboard apache2 libapache2-mod-wsgi memcached python-memcache
# vim /etc/openstack-dashboard/local_settings.py
        OPENSTACK_HOST = "controller"
        ALLOWED_HOSTS = ['*']
        CACHES = {
           'default': {
```

```
            'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache',
            'LOCATION': '127.0.0.1:11211',
        }
    }

# service apache2 restart
# service memcached restart
```

XI. Add the Block Storage service (controller node)

```
# mysql -u root -p
$ CREATE DATABASE cinder;
$ GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'localhost' \
  IDENTIFIED BY 'CINDER_DBPASS';
$ GRANT ALL PRIVILEGES ON cinder.* TO 'cinder'@'%' \
  IDENTIFIED BY 'CINDER_DBPASS';
# keystone user-create --name cinder --pass CINDER_PASS
# keystone user-role-add --user cinder --tenant service --role admin
# keystone service-create --name cinder --type volume \
  --description "OpenStack Block Storage"
# keystone endpoint-create \
  --service-id $(keystone service-list | awk '/ volume / {print $2}') \
  --publicurl http://controller:8776/v1/%\(tenant_id\)s \
  --internalurl http://controller:8776/v1/%\(tenant_id\)s \
  --adminurl http://controller:8776/v1/%\(tenant_id\)s \
  --region regionOne
# keystone endpoint-create \
  --service-id $(keystone service-list | awk '/ volumev2 / {print $2}') \
  --publicurl http://controller:8776/v2/%\(tenant_id\)s \
  --internalurl http://controller:8776/v2/%\(tenant_id\)s \
  --adminurl http://controller:8776/v2/%\(tenant_id\)s \
  --region regionOne

# apt-get install cinder-api cinder-scheduler python-cinderclient
# vim /etc/cinder/cinder.conf
        [database]
        ...
        connection = mysql://cinder:CINDER_DBPASS@controller/cinder

        [DEFAULT]
        ...
        rpc_backend = rabbit
        rabbit_host = controller
        rabbit_password = RABBIT_PASS
        auth_strategy = keystone
        my_ip = 10.0.0.11
```

```
        verbose = True

        [keystone_authtoken]
        ...
        auth_uri = http://controller:5000/v2.0
        identity_uri = http://controller:35357
        admin_tenant_name = service
        admin_user = cinder
        admin_password = CINDER_PASS

# su -s /bin/sh -c "cinder-manage db sync" cinder
# service cinder-scheduler restart
# service cinder-api restart
# rm -f /var/lib/cinder/cinder.sqlite
```

XII. Add the Block Storage service (storage node)

```
# apt-get install cinder-volume python-mysqldb
# vim /etc/cinder/cinder.conf
        [database]
        ...
        connection = mysql://cinder:CINDER_DBPASS@controller/cinder

        [DEFAULT]
        ...
        rpc_backend = rabbit
        rabbit_host = controller
        rabbit_password = RABBIT_PASS
        auth_strategy = keystone
        my_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
        glance_host = controller
        verbose = True

        [keystone_authtoken]
        ...
        auth_uri = http://controller:5000/v2.0
        identity_uri = http://controller:35357
        admin_tenant_name = service
        admin_user = cinder
        admin_password = CINDER_PASS

# service tgt restart
# service cinder-volume restart
# rm -f /var/lib/cinder/cinder.sqlite
```

Appendix B

Swift Installation

I. OpenStack packages

```
# apt-get install ubuntu-cloud-keyring
# echo "deb http://ubuntu-cloud.archive.canonical.com/ubuntu" \
  "trusty-updates/juno main" > /etc/apt/sources.list.d/cloudarchive-juno.list
# apt-get update && apt-get dist-upgrade
```

II. Database server installation

```
# apt-get install mariadb-server python-mysqldb
# vim /etc/mysql/my.cnf

        [mysqld]
        ...
        bind-address = 10.0.0.11
        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
```

III. Messaging server installation

```
# apt-get install rabbitmq-server
# rabbitmqctl change_password guest RABBIT_PASS
# service rabbitmq-server restart
```

IV. Add the Identity service

```
# 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';

# apt-get install keystone python-keystoneclient
# vim /etc/keystone/keystone.conf

        [DEFAULT]
        ...
        admin_token = ADMIN_TOKEN

        [database]
        ...
        connection = mysql://keystone:KEYSTONE_DBPASS@controller/keystone

        [token]
        ...
        provider = keystone.token.providers.uuid.Provider
        driver = keystone.token.persistence.backends.sql.Token

        [revoke]
        ...
        driver = keystone.contrib.revoke.backends.sql.Revoke

        [DEFAULT]
        ...
        verbose = True

# su -s /bin/sh -c "keystone-manage db_sync" keystone
# service keystone restart
# rm -f /var/lib/keystone/keystone.db
# (crontab -l -u keystone 2>&1 | grep -q token_flush) || \
  echo '@hourly /usr/bin/keystone-manage token_flush > \
  /var/log/keystone/keystone-tokenflush.log 2>&1' >> /var/spool/cron/crontabs/keystone

 # export OS_SERVICE_TOKEN=ADMIN_TOKEN
```

```
# export OS_SERVICE_ENDPOINT=http://controller:35357/v2.0

# keystone tenant-create --name admin --description "Admin Tenant"
# keystone user-create --name admin --pass ADMIN_PASS --email EMAIL_ADDRESS
# keystone role-create --name admin
# keystone user-role-add --user admin --tenant admin --role admin

# keystone service-create --name keystone --type identity \
  --description "OpenStack Identity"
# keystone endpoint-create \
  --service-id $(keystone service-list | awk '/ identity / {print $2}') \
  --publicurl http://controller:5000/v2.0 \
  --internalurl http://controller:5000/v2.0 \
  --adminurl http://controller:35357/v2.0 \
  --region regionOne
```

V. Add Object Storage (controller node)

```
# keystone user-create --name swift --pass SWIFT_PASS
# keystone user-role-add --user swift --tenant service --role admin
# keystone service-create --name swift --type object-store \
  --description "OpenStack Object Storage"
# keystone endpoint-create \
  --service-id $(keystone service-list | awk '/ object-store / {print $2}') \
  --publicurl 'http://controller:8080/v1/AUTH_%(tenant_id)s' \
  --internalurl 'http://controller:8080/v1/AUTH_%(tenant_id)s' \
  --adminurl http://controller:8080 \
  --region regionOne
# apt-get install swift swift-proxy python-swiftclient python-keystoneclient \
  python-keystonemiddleware memcached
# curl -o /etc/swift/proxy-server.conf \
  https://raw.githubusercontent.com/openstack/swift/stable/juno/etc/proxy-server.conf-sample
# vim /etc/swift/proxy-server.conf
        [DEFAULT]
        ...
        bind_port = 8080
        user = swift
        swift_dir = /etc/swift

        [pipeline:main]
        pipeline = authtoken cache healthcheck keystoneauth proxy-logging proxy-server

        [app:proxy-server]
        ...
        allow_account_management = true
        account_autocreate = true
```

```
        [filter:keystoneauth]
        use = egg:swift#keystoneauth
        ...
        operator_roles = admin,_member_

        [filter:authtoken]
        paste.filter_factory = keystonemiddleware.auth_token:filter_factory
        ...
        auth_uri = http://controller:5000/v2.0
        identity_uri = http://controller:35357
        admin_tenant_name = service
        admin_user = swift
        admin_password = SWIFT_PASS
        delay_auth_decision = true

        [filter:cache]
        ...
        memcache_servers = 127.0.0.1:11211
```

VI. Add Object Storage (storage node)

```
# apt-get install xfsprogs rsync
# mkfs.xfs /dev/sdb1
# mkdir -p /srv/node/sdb1
# vim /etc/fstab
        /dev/sdb1 /srv/node/sdb1 xfs noatime,nodiratime,nobarrier,logbufs=8 0 2

# mount /srv/node/sdb1
# vim /etc/rsyncd.conf
        uid = swift
        gid = swift
        log file = /var/log/rsyncd.log
        pid file = /var/run/rsyncd.pid
        address = MANAGEMENT_INTERFACE_IP_ADDRESS

        [account]
        max connections = 2
        path = /srv/node/
        read only = false
        lock file = /var/lock/account.lock

        [container]
        max connections = 2
        path = /srv/node/
        read only = false
```

```
        lock file = /var/lock/container.lock

        [object]
        max connections = 2
        path = /srv/node/
        read only = false
        lock file = /var/lock/object.lock

# vim /etc/default/rsync
        RSYNC_ENABLE=true

# service rsync start
# apt-get install swift swift-account swift-container swift-object
# curl -o /etc/swift/account-server.conf \
  https://raw.githubusercontent.com/openstack/swift/stable/juno/etc/account-server.conf-sample
# curl -o /etc/swift/container-server.conf \
  https://raw.githubusercontent.com/openstack/swift/stable/juno/etc/container-server.conf-sample
# curl -o /etc/swift/object-server.conf \
  https://raw.githubusercontent.com/openstack/swift/stable/juno/etc/object-server.conf-sample

# vim /etc/swift/account-server.conf
        [DEFAULT]
        ...
        bind_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
        bind_port = 6002
        user = swift
        swift_dir = /etc/swift
        devices = /srv/node

        [pipeline:main]
        pipeline = healthcheck recon account-server

        [filter:recon]
        ...
        recon_cache_path = /var/cache/swift

# vim /etc/swift/container-server.conf
        [DEFAULT]
        ...
        bind_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
        bind_port = 6001
        user = swift
        swift_dir = /etc/swift
        devices = /srv/node

        [pipeline:main]
        pipeline = healthcheck recon container-server
```

```
        [filter:recon]
        ...
        recon_cache_path = /var/cache/swift

# vim /etc/swift/object-server.conf
        [DEFAULT]
        ...
        bind_ip = MANAGEMENT_INTERFACE_IP_ADDRESS
        bind_port = 6000
        user = swift
        swift_dir = /etc/swift
        devices = /srv/node

        [pipeline:main]
        pipeline = healthcheck recon object-server

        [filter:recon]
        ...
        recon_cache_path = /var/cache/swift

# chown -R swift:swift /srv/node
# mkdir -p /var/cache/swift
# chown -R swift:swift /var/cache/swift
```

VII. Create initial rings

```
# swift-ring-builder account.builder create 10 3 1
# swift-ring-builder account.builder add r1z1-10.0.0.51:6002/sdb1 100
# swift-ring-builder account.builder rebalance

# swift-ring-builder container.builder create 10 3 1
# swift-ring-builder container.builder add r1z1-10.0.0.51:6001/sdb1 100
# swift-ring-builder container.builder rebalance

# swift-ring-builder object.builder create 10 3 1
# swift-ring-builder object.builder add r1z1-10.0.0.51:6000/sdb1 100
# swift-ring-builder object.builder rebalance

# scp account.ring.gz storagenode:/etc/swift
# scp container.ring.gz storagenode:/etc/swift
# scp object.ring.gz storagenode:/etc/swift
```

VII. Add Object Storage (storage node)

```
# curl -o /etc/swift/swift.conf \
  https://raw.githubusercontent.com/openstack/swift/stable/juno/etc/swift.conf-sample

# vim /etc/swift/swift.conf
        [swift-hash]
        ...
        swift_hash_path_suffix = HASH_PATH_PREFIX
        swift_hash_path_prefix = HASH_PATH_SUFFIX

        [storage-policy:0]
        ...
        name = Policy-0
        default = yes

# chown -R swift:swift /etc/swift
# service memcached restart
# service swift-proxy restart
# swift-init all start
```

Appendix C

Ceph Installation

I. Ceph deploy setup

```
# wget -q -O- 'https://ceph.com/git/?p=ceph.git;a=blob_plain;f=keys/release.asc'
# sudo apt-key add -
# echo deb http://ceph.com/debian-{ceph-stable-release}/ $(lsb_release -sc) main
# sudo tee /etc/apt/sources.list.d/ceph.list
# sudo apt-get update && sudo apt-get install ceph-deploy
```

II. Create a cluster

```
# ceph-deploy new node1
# ceph-deploy install admin-node node1 node2 node3
# ceph-deploy mon create-initial

# ssh node2
# sudo mkdir /var/local/osd0
# exit

# ssh node3
# sudo mkdir /var/local/osd1
# exit

# ceph-deploy osd prepare node2:/var/local/osd0 node3:/var/local/osd1

# ceph-deploy osd activate node2:/var/local/osd0 node3:/var/local/osd1

# ceph-deploy admin admin-node node1 node2 node3
```

```
# sudo chmod +r /etc/ceph/ceph.client.admin.keyring
```