

東海大學

資訊工程研究所

碩士論文

指導教授：楊朝棟博士

在異質儲存雲上實作出支援檔案均勻分佈的軟體定義
儲存服務

The Implementation of Supporting Uniform Data
Distribution with Software-Defined Storage Service on
Heterogeneous Cloud Storage

研究生：鄭暉勳

中華民國一零五年六月

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 鄭 暉 勳 所提之論文

在異質儲存雲上實作出支援檔案均勻分佈
的軟體定義儲存服務

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

伍朝欽 簽章

委員

吳守山

劉榮春

時文中

指導教授

楊朝棟 簽章

中華民國 105 年 6 月 27 日

摘要

近年來，資訊產業的發展趨勢逐漸邁向雲端，各種相關的技術與應用與日俱增。隨著產業的普及，企業需求與個人使用的數量也將快速提升。與此同時，雲端產業發展過程中所需的建置成本與使用效能將成為如何在大環境中脫穎而出的關鍵。雲端服務主要的三大部份分別為網路、運算及儲存。其中以雲端儲存的部分為目前使用者普及率較高的服務。如何提供低成本高效率的儲存服務將成為我們所關心的議題。虛擬化技術為目前在此議題上的解決方案之一，透過虛擬化技術建置軟體定義儲存〔Software-Defined Storage, SDS〕，將儲存資源透過軟體集中化，整合異質環境，提升設備可用度，給予使用者客製化的使用方案，是 SDS 技術發展中所追求的共同目標。本論文首先將針對現有的普及化 SDS 服務作使用上的效能測試，透過架設 OpenStack 作為雲端環境，於其上整合 HDFS、Ceph、Swift 等不同的儲存系統，模擬各系統於不同規模時所表現出的效能反映。接著透過架設監控節點，獲取各個儲存節點上的資源使用率，最後由控制節點根據所獲取的各節點資源使用率資訊，實現一套機制使檔案能夠自動分配至適當的儲存系統，避免系統資源使用上的不均，造成叢集效能上的浪費。

關鍵字: 雲端服務，儲存服務，軟體定義儲存，檔案分佈，異質儲存

Abstract

In order to improve accessibility and efficiency of a cloud system, this work proposed a mechanism to integrate HDFS and Swift based on the OpenStack. We first build a heterogeneous storage environment including HDFS and Swift based on the open source OpenStack and then measure their performances. To integrate storage services of HDFS and Swift, we propose a proportion-based file distribution mechanism. The proportion for file partition is dependent on the remaining storage capacity so that we can distribute those sub files to different storage. This mechanism also enhances the file security. In addition, a high usability user interface is provided so as to make the proposed system more friendly. Experimental results show the efficiency of our system.

Keywords: Cloud service, Storage Service, Software-Defined Storage, Data distribution, Heterogeneous storage

致謝詞

能夠完成這篇論文，首先我要特別感謝我的指導教授楊朝棟老師，謝謝老師兩年來的指導，帶領我學習在資訊領域上的種種知識。老師以身傳教，教導我研究的方法和做事的態度，並指點我在學習過程中的正確方向，讓我能夠在磨練中成長。若沒有這些經歷，我想我便無法順利完成這篇論文。

感謝特地抽空前來參加的口試委員們：伍朝欽教授、呂芳懌教授、劉榮春教授以及時文中教授，對於各位教授所給予的寶貴建議，因為有你們的幫助，才能令我將這篇論文撰寫的更加完善。

論文的完成也需要感謝實驗室的各學長學弟、同學們的協助，在碩士兩年的生活中，幫助我解決各種問題，無論是技術上的瓶頸，或是生活上的困難，讓我能順利完成這兩年的學業。

感謝我的父母對我從小的栽培與包容，因為有你們各方面的支持，才能讓我無後顧之憂的在資訊領域完成碩士學業。

最後，感謝所有一路上幫助我和陪伴我的人，因為有你們的付出才能成就今天的我，謝謝你們。

東海大學 資訊工程學系 高效能計算實驗室 鄭暉勳 105 年 07 月

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	2
1.2 Thesis Goal and Contributions	2
1.3 Thesis Organization	3
2 Background Review and Related Work	4
2.1 Background Review	4
2.1.1 Virtualization	4
2.1.2 Software-Defined Storage	5
2.1.3 Swift	6
2.1.4 Ceph	7
2.1.5 HDFS	10
2.1.6 OpenStack	11
2.1.7 COSBench	14
2.1.8 Cubic Spline	15
2.2 Related works	16
3 System Design and Implementation	18
3.1 System Design Architecture	18
3.2 System Implementation	20
3.2.1 Storage service deployment	20
3.2.2 File distribution mechanism	23
3.2.3 User services	25

4	Experimental Results	29
4.1	Experimental Environment	29
4.2	Performance	30
4.3	User Interface	35
5	Conclusions and Future Work	40
5.1	Concluding Remark	40
5.2	Future Works	41
	References	42
	Appendix	46
A	OpenStack Installation	46
B	Ceph Installation	53
C	Hadoop Installation	55
D	Swift Installation	59

List of Figures

2.1	Swift architecture	6
2.2	Ceph architecture	8
2.3	HDFS architecture	11
2.4	Openstack Kilo arch	12
2.5	COSBench Architecture	15
2.6	Cubic Spline Schematic Diagram	16
3.1	System Architecture	19
3.2	Controller Architecture	19
3.3	OpenStack Overview	20
3.4	VM Instances	21
3.5	Ceph environment	21
3.6	Ceph instances	22
3.7	Swift environment	22
3.8	Swift instances	22
3.9	HDFS environment	23
3.10	HDFS instances	23
3.11	Data distribution method flow chart	25
3.12	Responsive Web Design	27
3.13	Web Language Architecture	28
4.1	Measure the upload speed	31
4.2	Network infrastructure speed	32
4.3	Storage capacity convergence speed when $\gamma = 0.3$	32
4.4	Storage capacity convergence speed when $\gamma = 0.5$	33
4.5	Storage capacity convergence speed when $\gamma = 0.9$	33
4.6	Similar capacity distribution with $\gamma = 0.9$	34
4.7	Expand a storage space with $\gamma = 0.5$	34
4.8	Upload time comparison	35
4.9	User interface overview	36
4.10	Overview page	37
4.11	Trash can page	38
4.12	Account page	38
4.13	Admin page	39
4.14	Admin functions	39

List of Tables

3.1	Software & language Specification	26
4.1	Hardware Specification	30
4.2	Virtual Machine Specification	30
4.3	Software Specification	30

Chapter 1

Introduction

In recent years, the rapid development of the information focus on the techniques for Cloud services [1,2] which is a concept that users can upload their requirement via internet to Cloud environment and then receive a response by post-processing, for example Cloud computation [3,4] and Cloud storage [5]. Most studies about the Cloud storage focus on data pretreatment method on client side [6] or data storage method on server side [7]. For example, cloud storage service user may concern their data security and storage availability, they may used some data encryption technology to improve their data security and used multiple cloud storage systems to address the important storage availability issue in the Cloud.

Software-defined storage (SDS) [8–10] is a kind of virtualization technology for cloud storage. It uses the software to integrate different resources. The software enabling SDS environments can provide policy management for feature options such as deduplication, replication, thin provisioning, snapshots and backup so as to improve accessibility and usability. Thus, SDS is a better choice for users to build their cloud. With out of generality, the cost and security are two important issues.

1.1 Motivation

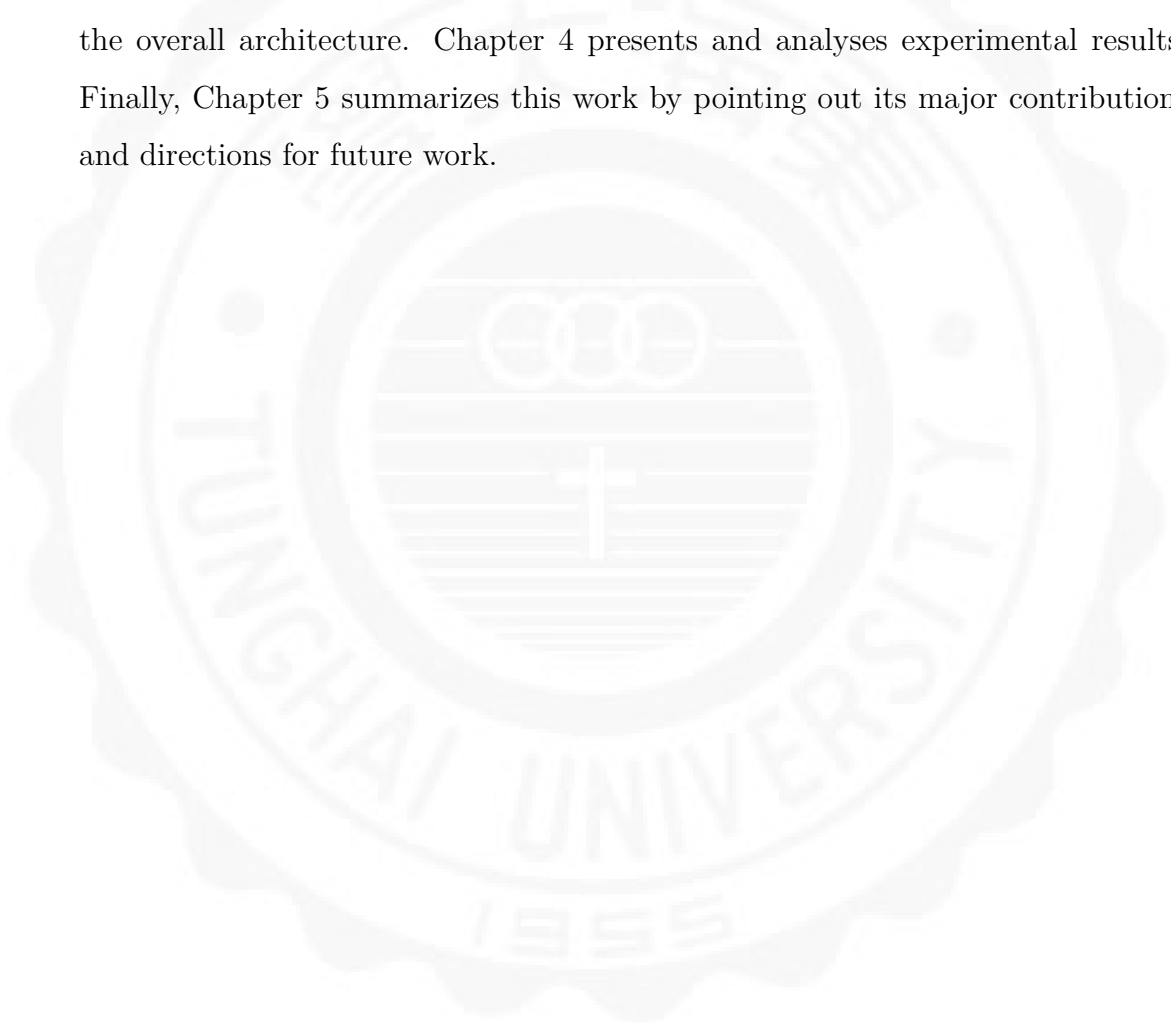
Cloud storage services have more convenience and provide data redundancy [11, 12]. It is an important indicators when the user selects this kind of service. However, due to the data shall be placed on the third-party storage platforms. Data Security in the cloud storage do not easily trust by user. To solve this issue, most of research used encrypt algorithm or erasure-code technology to improve data security [13, 14]. In this way, they might need to have more than one cloud storage services to distribute data. Therefore, we hope that through a variety of open-source software to implement private heterogeneous cloud storage service. Simulate the complex environment of cloud storage service and evaluating their effectiveness. Next, we will propose a method so that data can be assigned to different storage space in the process of uploading. Finally, we will provide a graph user interface which supports file sharing. User and administrator can operate our storage service through the web interface, achieve cloud services any time, anywhere access to any network device properties.

1.2 Thesis Goal and Contributions

This work will implement an integration cloud services with Software-defined storage technology. In the system architecture, we will use some open-source software, making it a better compatibility. And implement a method. The method can be automatically assigned files to an appropriate storage system when the user upload files. In this process, the files will be split as some small data. These data will be store in each storage service and enhanced security. Finally, we provide a graph user interface to manage our system.

1.3 Thesis Organization

Chapter 2 will describe some background information, including Virtualization, Software-Defined storage, OpenStack, Swift, Ceph, HDFS, COSBench and Cubic Spline. 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 work by pointing out its major contributions and directions for future work.



Chapter 2

Background Review and Related Work

In this section, we review some background knowledges for later use of system design and implementation.

2.1 Background Review

2.1.1 Virtualization

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 [15, 16].

Virtualization is commonly referred to virtualized resources including computing power and data storage [17]; in this paper virtualization is specifically referred to server virtualization. Server virtualization software technology refers to the use of one or more of the host hardware settings. It has the flexibility to configure the virtual hardware platform and operating system, like real hardware. In this way, a variety of different operating environments (for example, Windows, Linux, etc.) can operate simultaneously on the same physical host, and be independent

as being operating in different physical hosts. Virtualization solutions can be broadly divided into three categories: full virtualization, para-virtualization, and hardware-assisted virtualization.

2.1.2 Software-Defined Storage

Software-Defined Storage (SDS) [8–10] is a term 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.
- 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.

2.1.3 Swift

Swift [18] is a scalable redundant storage system. It is part of OpenStack component. 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.

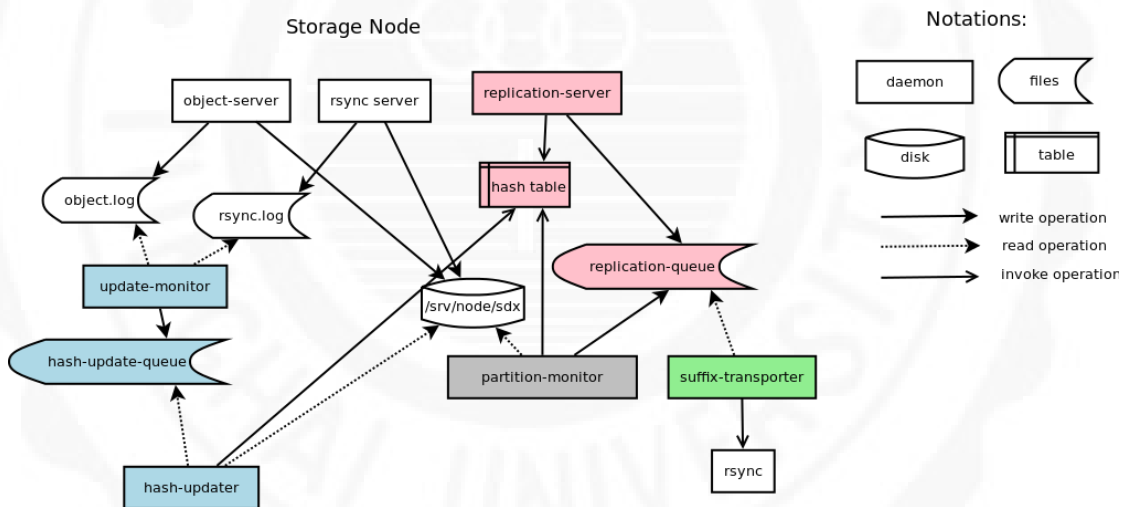


FIGURE 2.1: Swift architecture

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.

2.1.4 Ceph

Ceph [19] 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. Ceph was made possible by a global community of enthusiastic storage engineers and researchers. It is open source and freely-available. Ceph software runs on commodity hardware. The system is designed to be both self-healing and self-managing and strives to cut both administrator and budget costs.

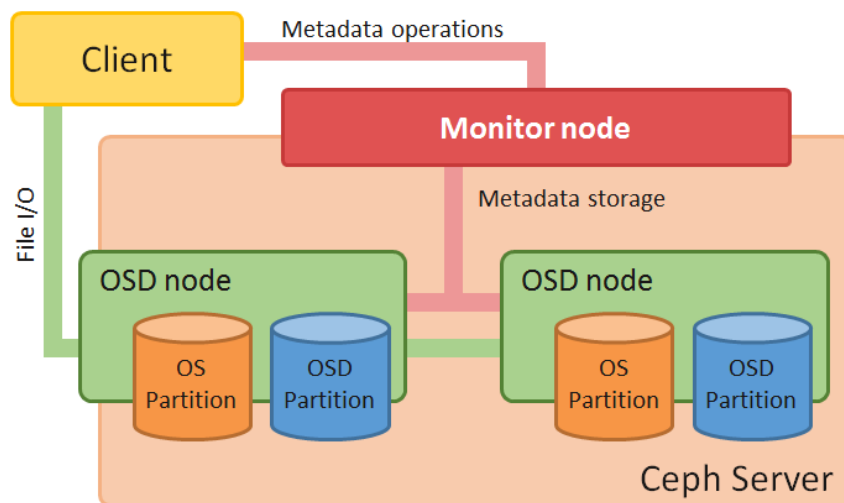


FIGURE 2.2: Ceph architecture

- **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 [20], 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
- **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.

2.1.5 HDFS

The Hadoop Distributed File System (HDFS) [21,22] is an Apache Software Foundation project and a subproject of the Apache Hadoop project. HDFS is a distributed file system designed to hold very large amounts of data (terabytes or even petabytes), and to provide high-throughput access to this information. Files are stored in a redundant fashion across multiple machines to ensure durability to failure and high availability to parallel applications. HDFS has many similarities with other distributed file systems, but is different in several respects. One noticeable difference is HDFS's write-once-read-many model that relaxes concurrency control requirements, simplifies data coherency, and enables high-throughput access. Another unique attribute of HDFS is the viewpoint that it is usually better to locate processing logic near the data rather than moving the data to the application space. HDFS rigorously restricts data write to one write at a time. Bytes are always appended to the end of a stream, and byte streams are guaranteed to be stored in the written order.

HDFS is comprised of interconnected clusters of nodes where files and directories reside. There are two type of nodes in HDFS. One is NameNode ,and the other is DataNode. A HDFS cluster consists of a single NameNode, which manages the file system namespace and regulates client access to files. In addition, DataNodes store data as blocks within files and satisfy client I/O requests. Within HDFS, a given name node manages file system namespace operations like opening, closing, and renaming files and directories. A NameNode also maps data blocks to data nodes that handle read and write requests from HDFS clients. DataNodes also create, delete, and replicate data blocks according to instructions from the governing NameNode. The architecture of HDFS is shown in Figure 2.3.

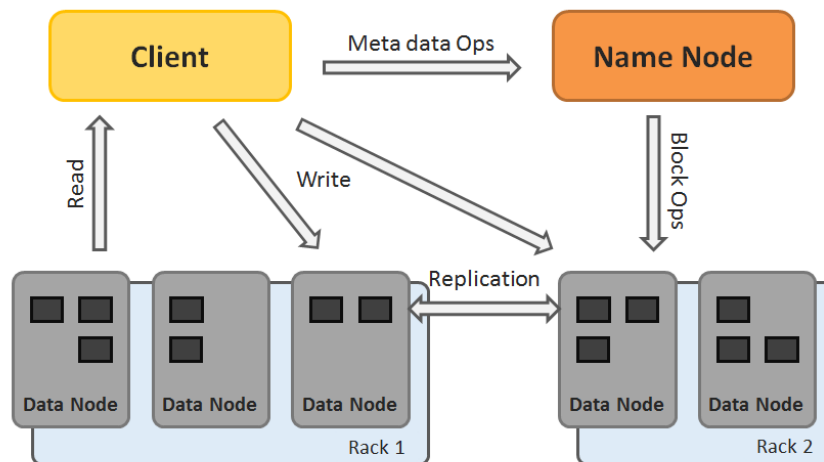


FIGURE 2.3: HDFS architecture

2.1.6 OpenStack

OpenStack [23,24] 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.

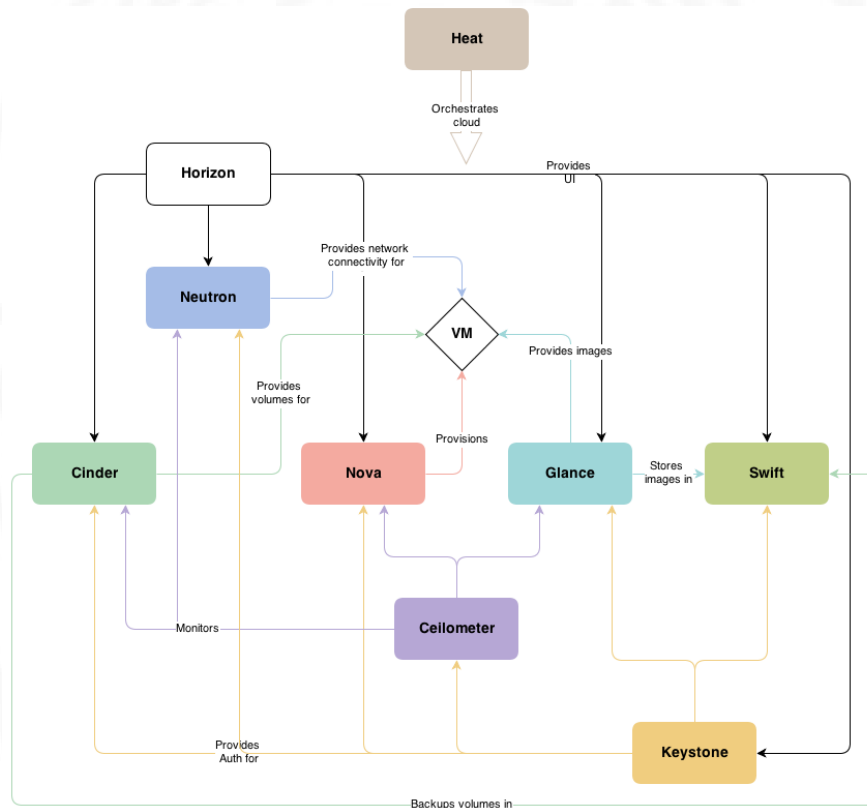


FIGURE 2.4: Openstack Kilo arch

In this work, we use version Kilo. The architecture is as shown in Figure 2.4.

- Identity Service (Keystone) Keystone controls all authentication in OpenStack. It provides a central directory of users mapped to the OpenStack services they can access and also supports multiple forms of authentication including standard username and password credentials, token-based systems and AWS-style (i.e. Amazon Web Services) logins.

- **Image Service (Glance)** Glance manage the OpenStack images. It provides discovery, registration, and delivery services for disk and server images. It can also be used to store and catalog an unlimited number of backups and the stored images can be used as a template.
- **Compute (Nova)** Nova is virtual machine provisions. 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 configurations. User can choose KVM, VMware, and Xen as their hypervisor technology, together with Hyper-V and Linux container technology such as LXC.
- **Networking (Neutron)** Neutron is used to manage networks and IP addresses. It ensures the network is not a bottleneck or limiting factor in a cloud deployment, and gives users self-service ability, even over network configurations.
- **Object Storage (Swift)** Swift is a scalable redundant object storage system. Data 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.
- **Block Storage (Cinder)** Cinder provides volumes for virtual machines. It is designed to allow the use of either a reference implementation to present storage resources to end users that can be consumed by the OpenStack Compute Project.
- **Dashboard (Horizon)** Horizon provides provides administrators and users a graphical interface to access, provision, and automate cloud-based resources. The sites built with django and also provide the API for developer to automate access or build tools to manage resources.
- **Telemetry (Ceilometer)** Ceilometer providing all the counters they need to establish customer billing, across all current and future OpenStack components.

- 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.
- 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.

We used Keystone, Keystone, Nova, Glance, Neutron, Cinder and Horizon in our model.

2.1.7 COSBench

Cloud Object Storage Benchmark (COSBench) [25] 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.

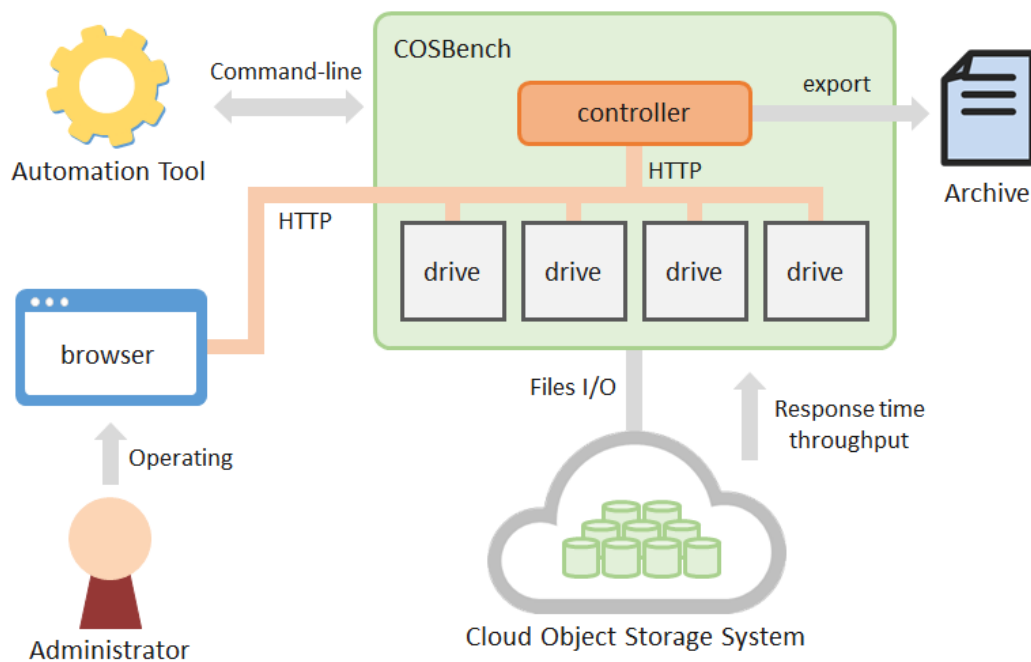


FIGURE 2.5: COSBench Architecture

2.1.8 Cubic Spline

In mathematics, a spline [26] 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.

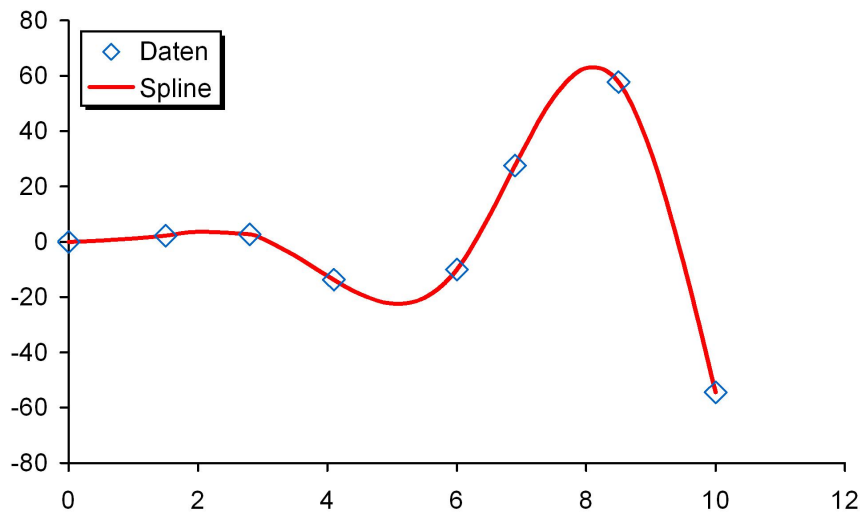


FIGURE 2.6: Cubic Spline Schematic Diagram

A cubic spline [27,28] 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

Since VMware propose the concept of "Software-defined data center", the research of Software-Defined Storage development and cloud storage have become more and more. Through the concept of virtualization, integrate hardware resource to a system, make expansion of storage cluster easier. Through the release of every kind of open source software, deploy a private storage cluster has become a choice for some enterprise.

Software-Defined may be different concept. However, cloud computing is brewing more possibilities. Hardware and software architecture has been gradually

change. These will become the custom functions and automation of operations. There are software-defined storage research papers and products released.

Against small organizations or personal user, there have many enterprise provide the service of cloud storage to suit every kind of requirement. Such as Windows, amazon, and Google have provide different kind of service for different user groups. How to choose the most suitable service for their self has become the target of many research.

Chengzhang Penga and Zejun Jiangb proposed a cloud storage service system [29], in which a solution is suggested to build a cloud storage service system based on the open-source distributed database.

Josef Spillner propose detailed insight for life cycle of cloud service, and propose an integration platform to compatibility of every cloud storage service to avoid single cloud service has some problems to bring about storage interrupted [6]. But the system is mainly for experience of user and link up for every kinds of cloud platform, it's less for processing files.

Suzhen Wu propose a method that store in different cloud service after separation of file, and propose a dissertation for copies requirement of file [30]. But they did not talk about the method of file distribution.

Yu-Chuan Shen propose a method against different size of file upload to the most suitable system [31]. But if user is conventional to use large or small file, this method will bring a result that all file store in the same system.

Whether use the backend platform of private cloud or public cloud, use all storage system in acceptable speed of I/O and security of user, is a target for every cloud storage system. Therefore, this work will base on [31], to propose a method of supporting file distribute average, make the system whenever expand node, it's can achieve to storage load balance.

Chapter 3

System Design and Implementation

The main goal of our system is to build a cloud platform contains a variety of storage technologies, and achieve a uniform distribution of data stored thereon. This section describes our overall system design architecture with some applications of open source softwares. The proposed heterogeneous storage system is implemented based on this architecture and our proposed distribution mechanism. Moreover, we provide a graphical user interface for users to enjoy the whole system.

3.1 System Design Architecture

In the proposed system architecture shown in Figure 3.1, OpenStack is adopted as the basis in order to achieve storage virtualization and unified management. Based on OpenStack, we can create a virtual machine to provide storage service, control service and monitoring service. The storage service is the basis of the heterogeneous storage platforms, such as Swift, HDFS and Ceph. The control service is built on the Controller to manage the storage services and the heterogeneous storages. The monitor service is used to monitor the remaining capacity of each

heterogeneous storage platform. To allocate the files received from users, a distribution mechanism is proposed for Controller. The mechanism can automatically assign files to an appropriate storage after users upload files. The detail concept of the Controller is shown in Figure 3.2. In addition, we also provide graphical user interface on web browser so that users can enjoy the proposed cloud system by web browser anytime and anywhere.

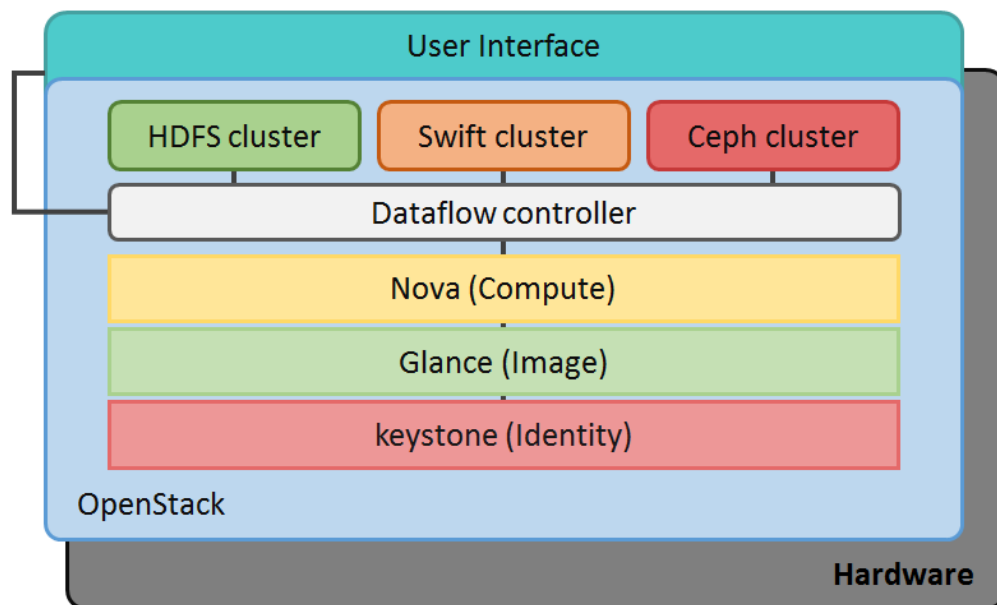


FIGURE 3.1: System Architecture

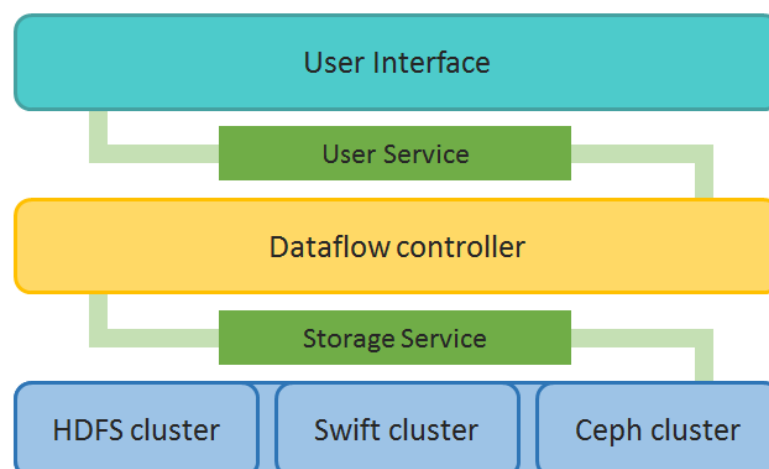


FIGURE 3.2: Controller Architecture

3.2 System Implementation

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

3.2.1 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 overview of the system is shown in Figure 3.13 and Figure 3.4.

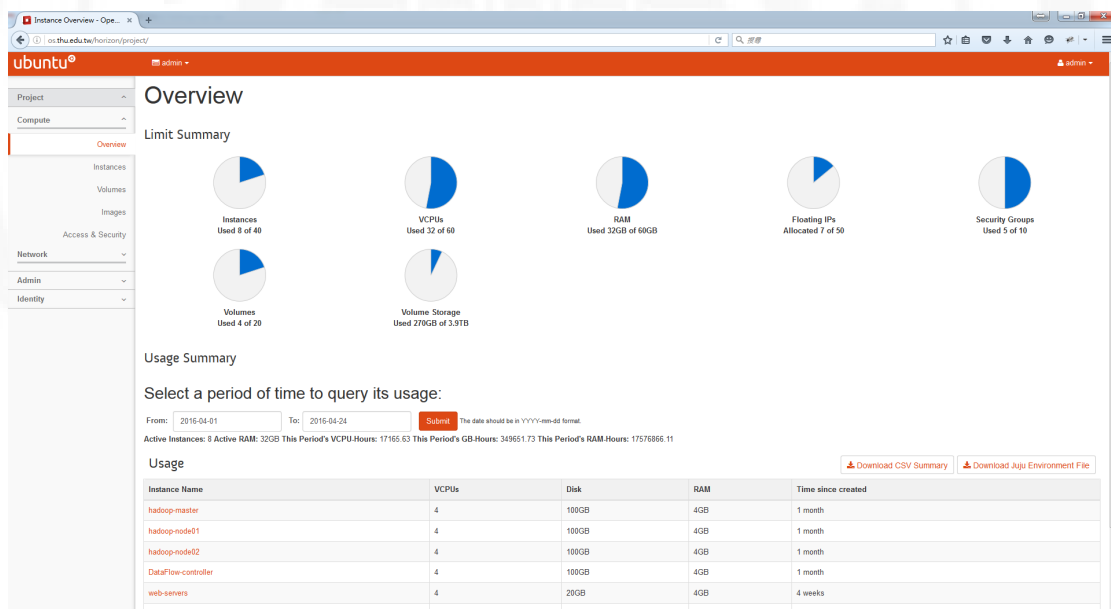


FIGURE 3.3: OpenStack Overview

Ceph deployment

Ceph, a free-software storage platform, implements distributed object storage and file system, and provides interfaces for object-, block- and file-level storage. It has excellent performance, reliability and scalability. To achieve the functions above, Ceph has three kind of physical nodes: Object Storage Daemon (OSD), Monitors (MON) and Metadata (MDS) service. According to object storage deploy

雲實例名稱	映像名稱	IP 地址	容量	密碼對	狀態	可用底層	任務	電源狀態	壽命	Actions
ceph-OSD2	ubuntu-20G	10.0.0.75	ubuntu_4.4.20	demo-key	使用中	compute02	無	正在執行	0分	新增即時存檔
ceph-OSD1	ubuntu-20G	10.0.0.74	ubuntu_4.4.20	demo-key	使用中	compute01	無	正在執行	9分	新增即時存檔
ceph-mon	ubuntu-20G	10.0.0.73	ubuntu_4.4.20	demo-key	使用中	compute01	無	正在執行	17分	新增即時存檔
swift-node02	ubuntu-20G	10.0.0.72	ubuntu_4.4.20	demo-key	使用中	compute02	無	正在執行	22分	新增即時存檔
swift-node01	ubuntu-20G	10.0.0.71	ubuntu_4.4.20	demo-key	使用中	compute02	無	正在執行	26分	新增即時存檔
swift-service	ubuntu	10.0.0.66 浮動 IP : 140.128.98.41	HPC	demo-key	使用中	compute02	無	正在執行	2週	新增即時存檔
sp-01	-	10.0.0.62 浮動 IP : 140.128.98.46	monitor	demo-key	使用中	compute02	無	正在執行	2週, 4日	新增即時存檔
web-servers	ubuntu-20G	10.0.0.59 浮動 IP : 140.128.98.45	monitor	demo-key	使用中	compute02	無	正在執行	4週	新增即時存檔
DataFlow-controller	ubuntu	10.0.0.57	HPC	demo-key	使用中	compute02	無	正在執行	1月	新增即時存檔
hadoop-node02	ubuntu	10.0.0.51	HPC	demo-key	使用中	compute02	無	正在執行	1月	新增即時存檔
hadoop-node01	ubuntu	10.0.0.50	HPC	demo-key	使用中	compute02	無	正在執行	1月	新增即時存檔
hadoop-master	ubuntu	10.0.0.49 浮動 IP : 140.128.98.42	HPC	demo-key	使用中	compute02	無	正在執行	1月	新增即時存檔

FIGURE 3.4: VM Instances

requirements, as shown in Figure 2.2. We only need to install OSDs and MONs. The overview of our Ceph architecture is shown in Figure 3.5 and Figure 3.6.

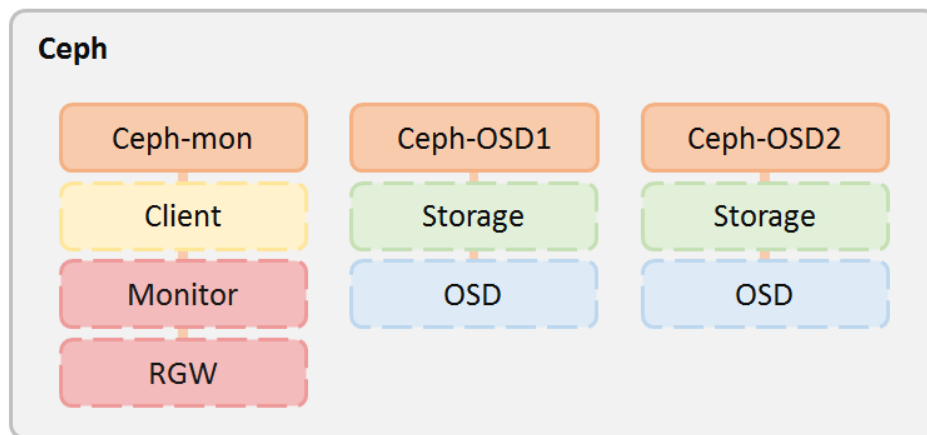


FIGURE 3.5: Ceph environment

Swift deployment

Swift is one of component in OpenStack. The overview of Swift architecture is shown in Figure 2.1. Swift service includes 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

<input type="checkbox"/>	雲實例名稱	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	壽命	Actions
<input type="checkbox"/>	ceph-OSD2	ubuntu-20G	10.0.0.75	ubuntu_4.4.20	demo-key	使用中	compute02	無	正在執行	0 分	新增即時存檔
<input type="checkbox"/>	ceph-OSD1	ubuntu-20G	10.0.0.74	ubuntu_4.4.20	demo-key	使用中	compute01	無	正在執行	8 分	新增即時存檔
<input type="checkbox"/>	ceph-mon	ubuntu-20G	10.0.0.73	ubuntu_4.4.20	demo-key	使用中	compute01	無	正在執行	17 分	新增即時存檔

FIGURE 3.6: Ceph instances

an internal mechanism that allows it to operate without any other OpenStack services. According to Swift deploy requirements. We need 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 3.7 and Figure 3.8.

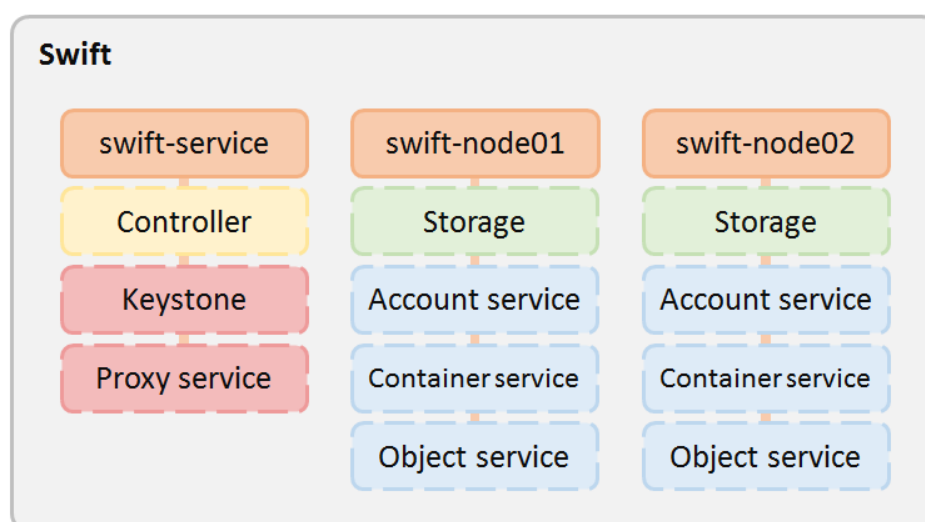


FIGURE 3.7: Swift environment

<input type="checkbox"/>	雲實例名稱	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	壽命	Actions
<input type="checkbox"/>	swift-node02	ubuntu-20G	10.0.0.72	ubuntu_4.4.20	demo-key	使用中	compute02	無	正在執行	22 分	新增即時存檔
<input type="checkbox"/>	swift-node01	ubuntu-20G	10.0.0.71	ubuntu_4.4.20	demo-key	使用中	compute02	無	正在執行	26 分	新增即時存檔
<input type="checkbox"/>	swift-service	ubuntu	10.0.0.66 浮動 IP : 140.128.98.41	HPC	demo-key	使用中	compute02	無	正在執行	2 週	新增即時存檔

FIGURE 3.8: Swift instances

HDFS deployment

Hadoop has two kinds of nodes: master node and slave node. Master node uses NameNode service to control DataNode service which is running on slave nodes. We built a HDFS architecture consisting of one master node and two slave nodes, as shown in Figure 3.9 and Figure 3.10. The NameNode executes file system namespace operations and also determines the mapping of blocks DataNodes. DataNodes are responsible for serving read and write requests from clients of file system.

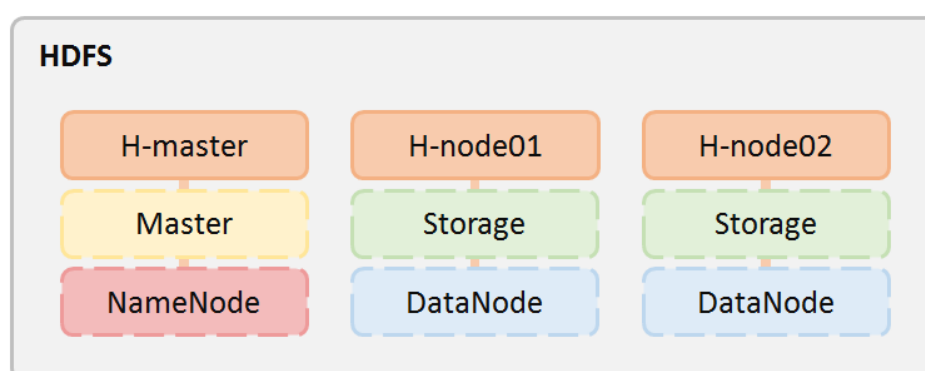


FIGURE 3.9: HDFS environment

<input type="checkbox"/>	雲實例名稱	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	壽命	Actions
<input type="checkbox"/>	hadoop-node02	ubuntu	10.0.0.51	HPC	demo-key	使用中	compute02	無	正在執行	1月	新增即時存檔
<input type="checkbox"/>	hadoop-node01	ubuntu	10.0.0.50	HPC	demo-key	使用中	compute02	無	正在執行	1月	新增即時存檔
<input type="checkbox"/>	hadoop-master	ubuntu	10.0.0.49 浮動 IP : 140.128.98.42	HPC	demo-key	使用中	compute02	無	正在執行	1月	新增即時存檔

FIGURE 3.10: HDFS instances

3.2.2 File distribution mechanism

The purpose of this section is to propose a mechanism for file distribution and appropriate file allocation according to the environment and specification of the heterogeneous storage platforms, Swift, HDFS and Ceph. In the step of the proposed mechanism, we can observe remaining capacity information of each storage

platform through our monitor node. In the second step, we split each file into several sub-files with the size similar to the proportion of the remaining capacity of all storage platforms. Based on the above information, we allocate each sub-file to a proper storage platform. The following mathematical equations show the detail of the proposed mechanism..

$$X = \sum_{i=1}^n S_i \quad (3.1)$$

$$S_i = X * R_i, \quad i = 1 \text{ to } n \quad (3.2)$$

- X is the user-uploaded file size. We use it as file segmentation data.
- S is the result of the split file size. Also be uploaded to the storage system's actual size.
- R is the split ratio. We use it to find the value S .
- n is the number of storage system selection. In our case, we have three storage system so that n is 3.

As above, we know that R is an important parameter which influences platform capacity convergence speed. We show how to determine this value in the following:

$$R_j = \frac{C_j - \gamma * \min(C)}{\sum_{i=1}^n C_i - \gamma * \min(C)}, \quad j = 1 \text{ to } n \quad (3.3)$$

- C is the system storage capacity ratio which satisfied $\sum_{i=1}^n C_i = 1$.
- γ is a filtering value satisfied $0 < \gamma \leq 1$.

As a result, we can get R by equation 3.3. After the process described above, we start to split our file. Finally, all the split files will be uploaded to the specified platform. The flow chart of the proposed file distribution mechanism is shown in Figure 3.11.

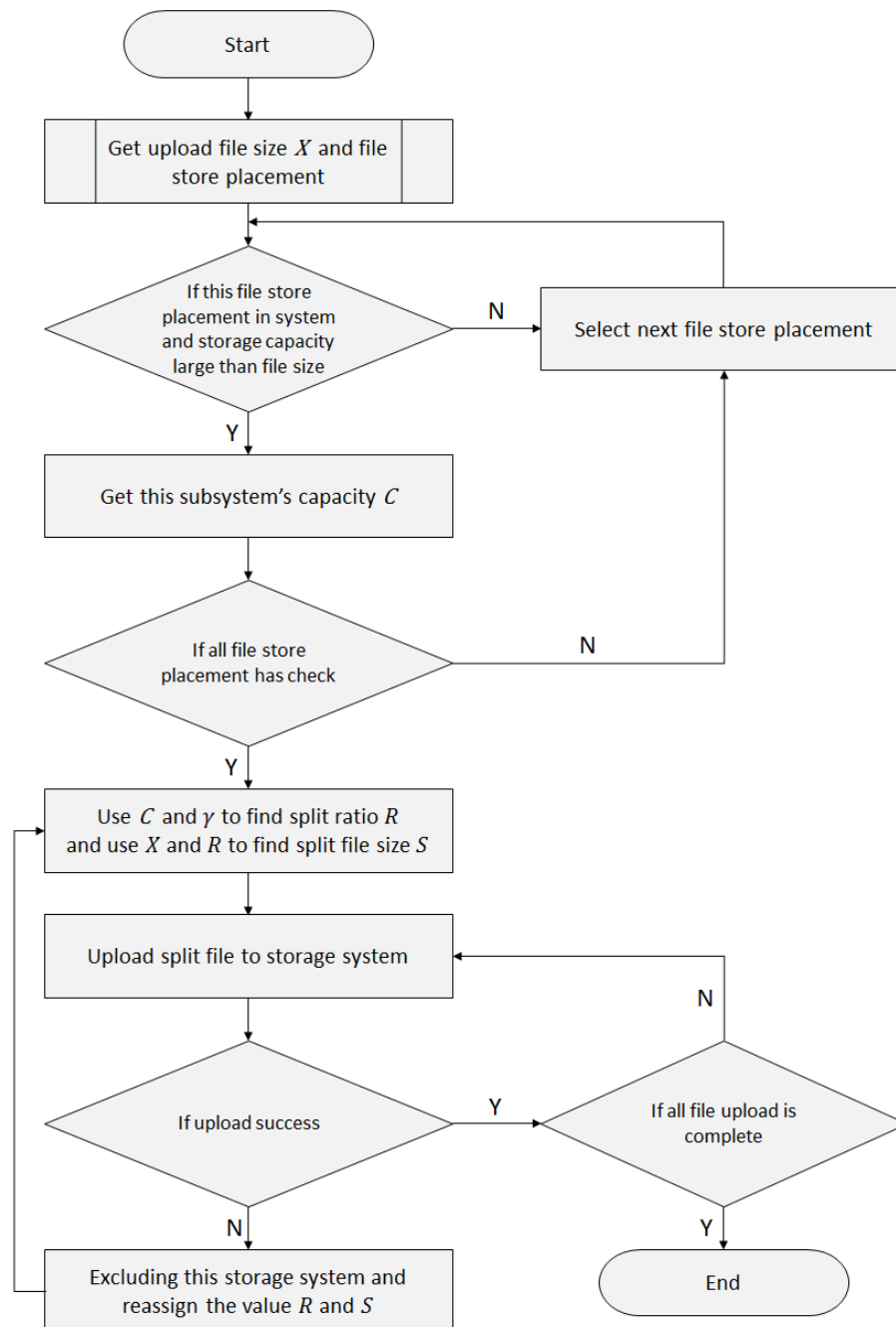


FIGURE 3.11: Data distribution method flow chart

3.2.3 User services

In order to solve the platform compatibility issues, we designed a web site as our user interface. Because the web service is compatible with a variety of platforms such as PC, mobile and tablet. Our client and server adopt JQuery and PHP

language. Also, we used several techniques to help us build our interface, as shown in 3.1.

TABLE 3.1: Software & language Specification

Software & language	Version
PHP	5.5.9
JQuery	3.0.0
MariaDB	10.1.14
Bootstrap	3.3.6
D3.js	3.5.16
Python	2.7.6

Besides above, we used the Responsive Web Design (RWD) to design our web interface. The RWD is an approach to web design. This feature provides users with a best visual and interactive experience. There are some concept in the responsice 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.

In order to reduce the complexity of development. We use Bootstrap as our website framework. Bootstrap is a powerful front-end framework for faster and easier web development. It includes HTML and CSS based design templates for common user interface components.

The personal computer screen shown in 3.12(a). The mobile screen shown in 3.12(b). We can seen the different kind of web layout but in the same style.

Because there is only one html page on server. The web will show different layout according to user's screen.

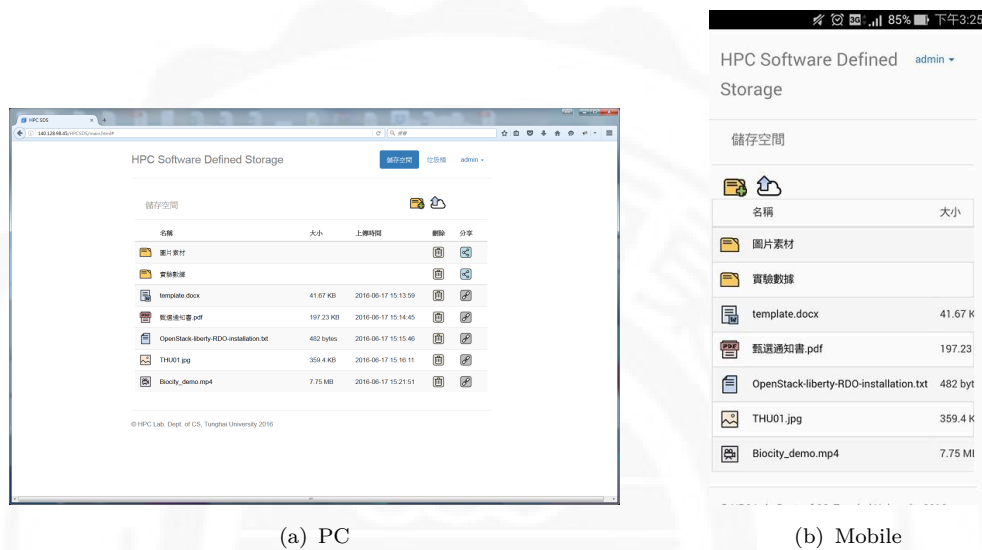


FIGURE 3.12: Responsive Web Design

The user experience is an important basis for the development of our Graphical User Interface (GUI). Our web service using PHP as the backend language. After the PHP is interpreted and executed, the web server sends resulting output to its client. Also, the client need better interaction. We using JavaScript as the front-end language. 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 some sql request and other contents needed update by refresh the web page. This situation is not a good interaction experience.

Therefore, we also use the Ajax and JQuery to solve this problem. There are some benefits in 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.

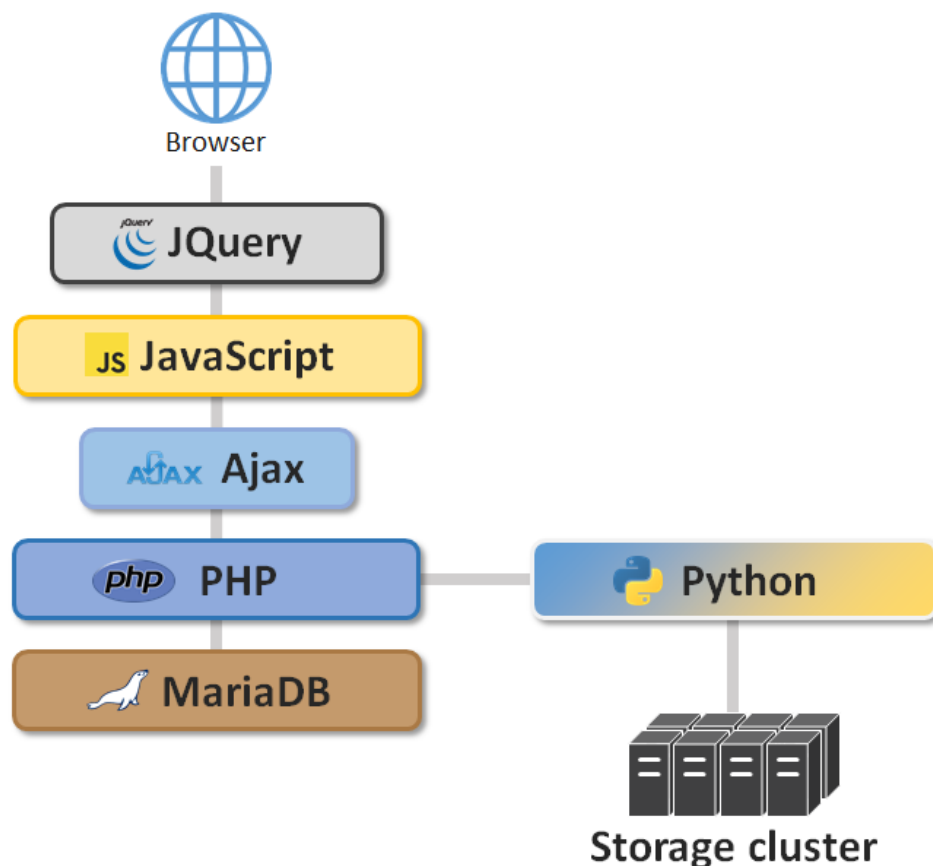


FIGURE 3.13: Web Language Architecture

- 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.

Asynchronous JavaScript and XML is a interrelated Web development technique used on the client-side to create asynchronous web applications. When client sent a request to server, JavaScript will call PHP by Ajax. After process, PHP will return a XML data to JavaScript. Then, JavaScript will refresh the designated area. This methods will reduce duplication of data transmission on the client-side.

Chapter 4

Experimental Results

In this section, we present the experiments and system implementation results. First, we measure speeds of each storage platform, and this measurement provides the basis of the file distribution mechanism. Next, we measure spent time of file split since it provides our system evaluation the extra time. Third, we compare the time gap of file upload time. Finally, we show the user interface we designed in the system.

4.1 Experimental Environment

We used OpenStack to build our cloud platform, which then was used to create and manage the storage distribution. As a simple example, we integrated two heterogeneous storage technologies. And we built the storage system by some VMs, in which HDFS was constructed by three VMs with specifications of 4-core CPU, 4 GB memory, and a total of 200 GB storage space. Table 4.1, 4.2, 4.3 are our experimental environment specification.

TABLE 4.1: Hardware Specification

Host name	CPU	Memory	Disk	OS
Openstack Controller	12 cores	64GB	2TB	Ubuntu 14.04.02
Openstack Network	24 cores	64GB	2TB	Ubuntu 14.04.02
Openstack Compute01	64 cores	96GB	2TB	Ubuntu 14.04.02
Openstack Compute02	64 cores	96GB	2TB	Ubuntu 14.04.02
Openstack Block Storage	64 cores	48GB	8TB	Ubuntu 14.04.02

TABLE 4.2: Virtual Machine Specification

Host name	CPU	Memory	Disk	OS
client	4 cores vCPU	4GB	20GB	Ubuntu 14.04.02
Ceph mon	4 cores vCPU	4GB	20GB	Ubuntu 14.04.02
Ceph OSD1	4 cores vCPU	4GB	20GB	Ubuntu 14.04.02
Ceph OSD2	4 cores vCPU	4GB	20GB	Ubuntu 14.04.02
Swift controller	4 cores vCPU	4GB	20GB	Ubuntu 14.04.02
Swift node01	4 cores vCPU	4GB	20GB	Ubuntu 14.04.02
Swift node02	4 cores vCPU	4GB	20GB	Ubuntu 14.04.02
HDFS master	4 cores vCPU	4GB	20GB	Ubuntu 14.04.02
HDFS node01	4 cores vCPU	4GB	20GB	Ubuntu 14.04.02
HDFS node02	4 cores vCPU	4GB	20GB	Ubuntu 14.04.02

TABLE 4.3: Software Specification

Software	Version
OpenStack	Kilo
Ceph	10.2.1 Jewel
Swift	2.1.0
Hdfs	2.7.1

4.2 Performance

Measure upload speed

The results of upload speeds are marked on the diagram, as shown in Figure 4.1. The blue line is Ceph, red line is HDFS and gray line is Swift. We can see in our configuration, Ceph and Swift has a poor upload performance in large file.

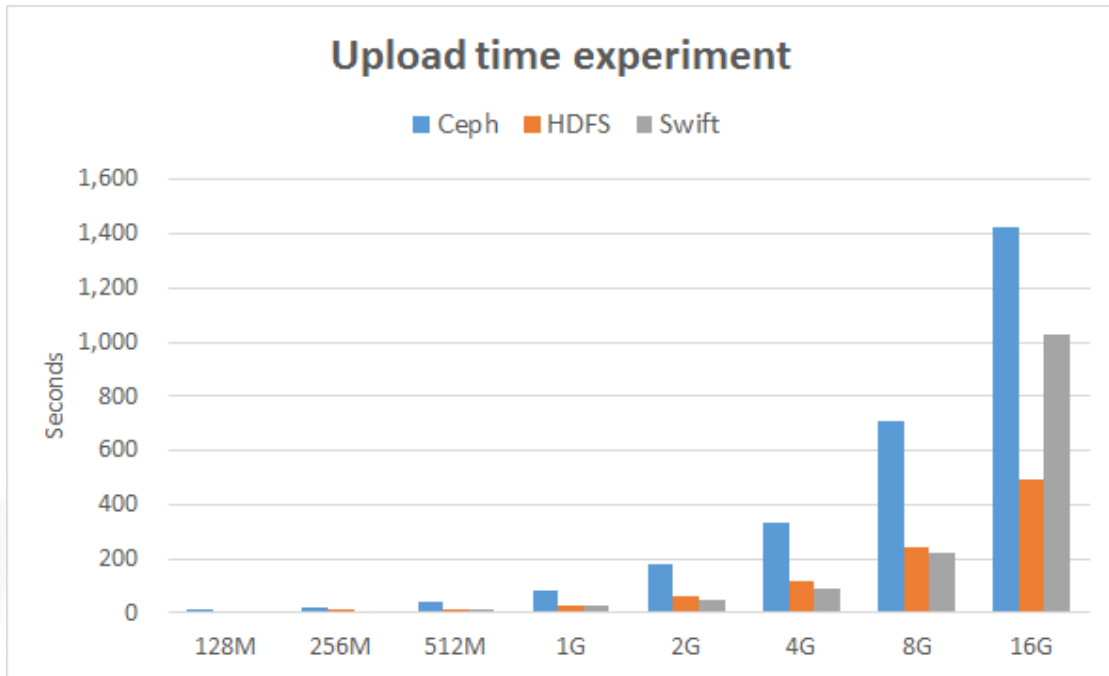


FIGURE 4.1: Measure the upload speed

Measure network infrastructure speed

Network throughput is a key factor affecting cluster performance. To determine our network performance, we choose iperf which uses a client-server connection to measure TCP and UDP bandwidth as our testing tool. The results shown in Figure 4.2. In histogram, horizontal axis is the number of tests, and vertical axis is a transmission bandwidth. Bandwidth of Ceph is almost about 560 Mbits/s. Bandwidth of HDFS is almost about 520 Mbits/s. Bandwidth of Swift is almost about 545 Mbits/s. Evidently, the result is not much difference. The reason of the result is our hosts have the same physical machine specifications.

Data distribution experiment

Our methods allow user set the γ value to control the storage capacity convergence speed. To evaluate our distribution effectiveness in different γ value. We given Ceph 2000 Gigabyte, HDFS 1000 Gigabyte, Swift 1500 Gigabyte capacity. Then we have compare three case which γ value equal to 0.3, 0.5 and 0.9. The result is shown in Figure 4.3, 4.4 and 4.5.

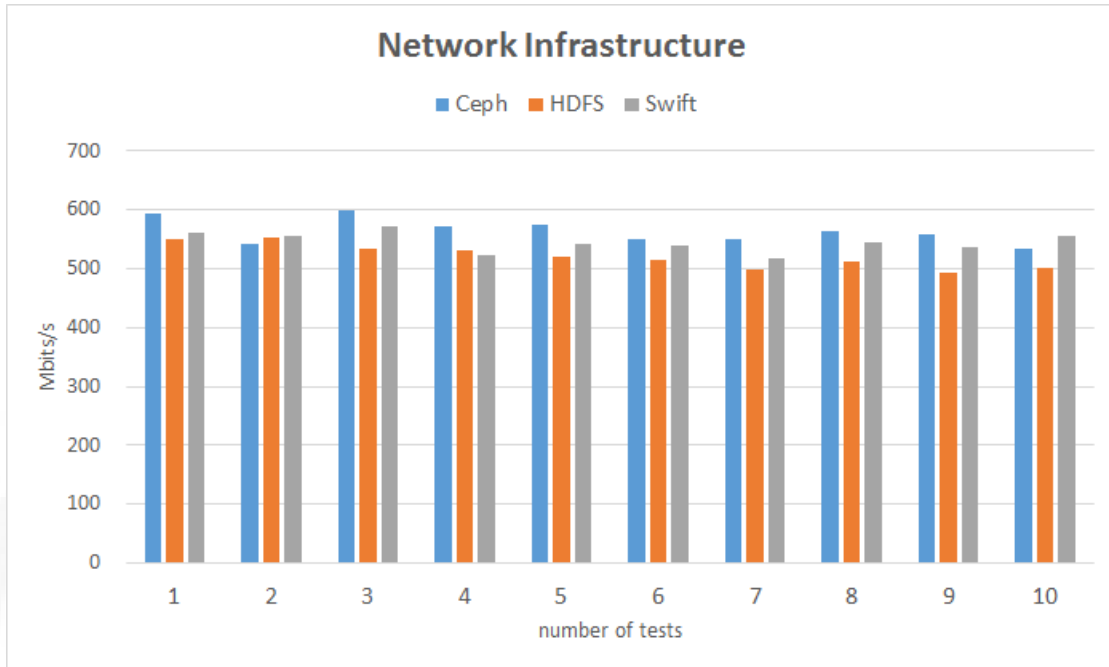
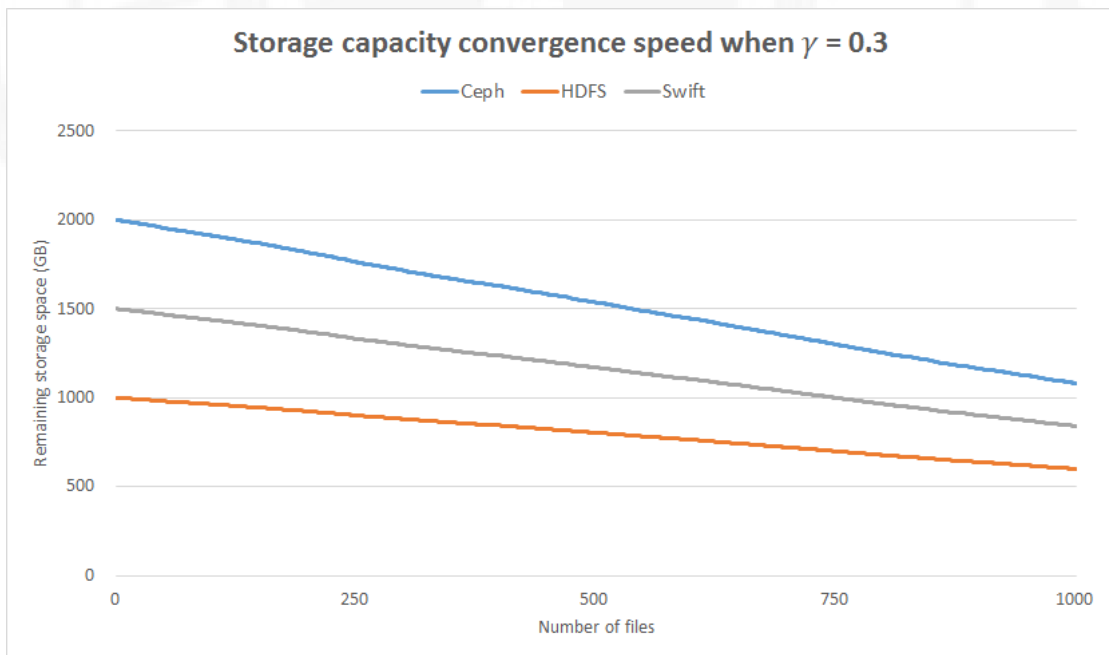
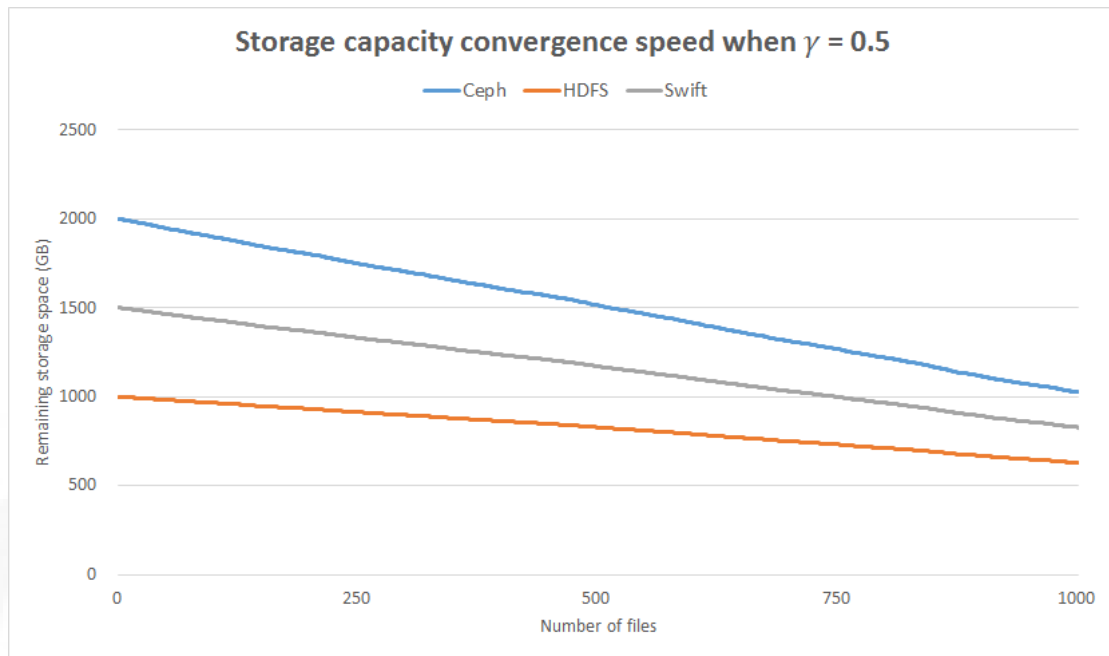
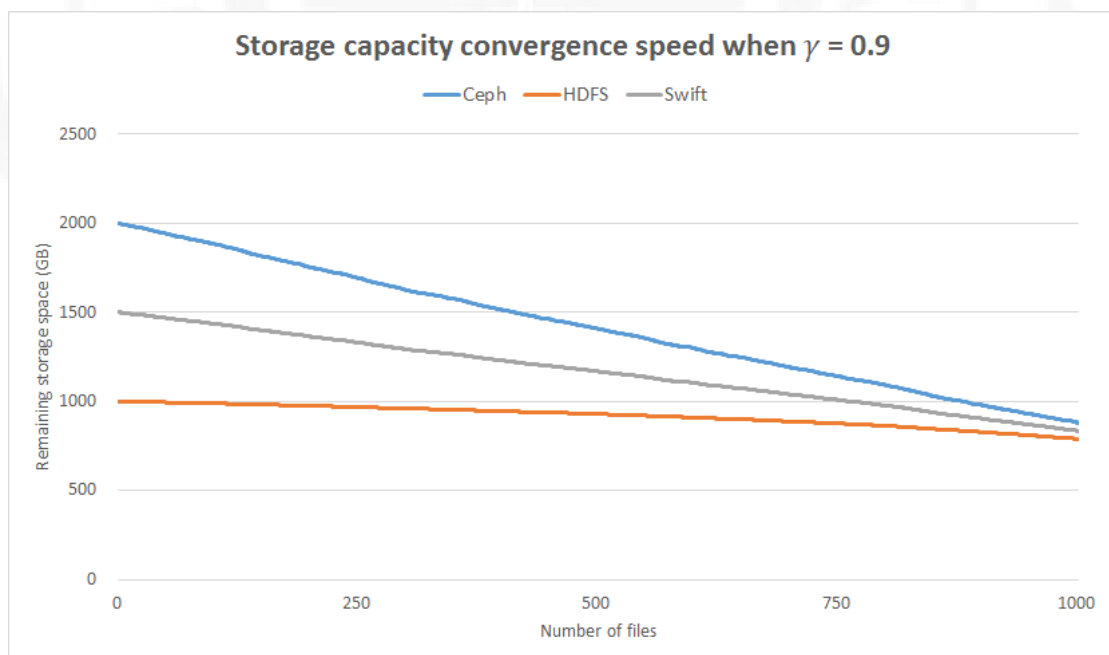


FIGURE 4.2: Network infrastructure speed

FIGURE 4.3: Storage capacity convergence speed when $\gamma = 0.3$

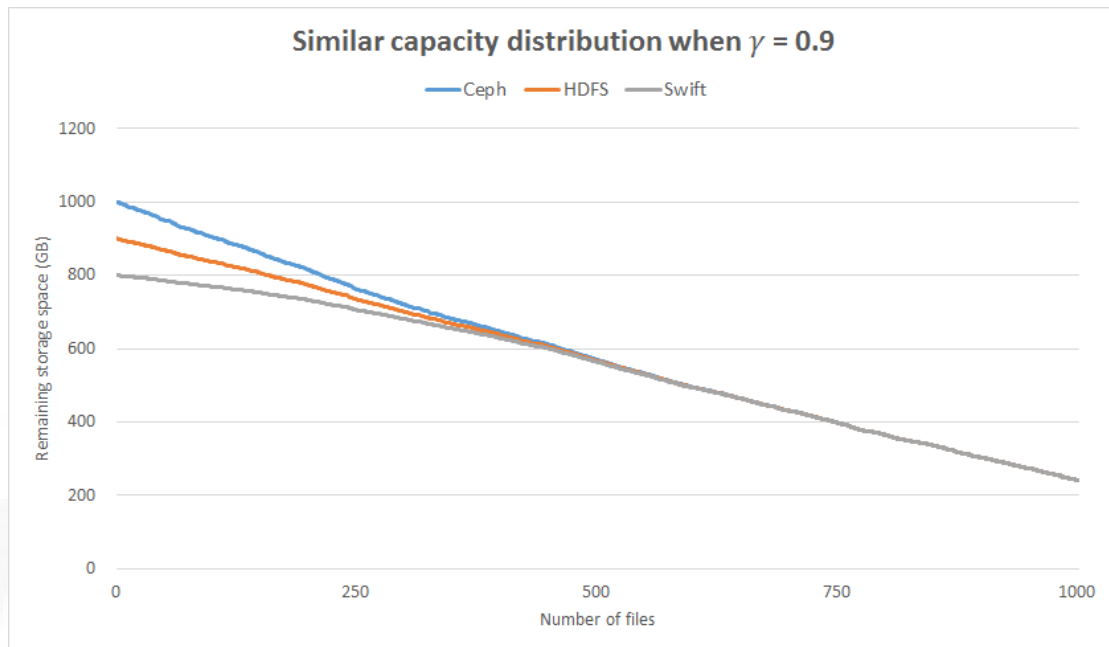
In these three experiments, we given 1000 random size file around 1 Mb to 4 Gb upload to our system. We can see the large γ value may caused the more obvious convergence rate. That means if user set the large γ value, the upload file will split more ratio to the large capacity storage system.

Also, we want to know if the storage system capacity is similar, our method

FIGURE 4.4: Storage capacity convergence speed when $\gamma = 0.5$ FIGURE 4.5: Storage capacity convergence speed when $\gamma = 0.9$

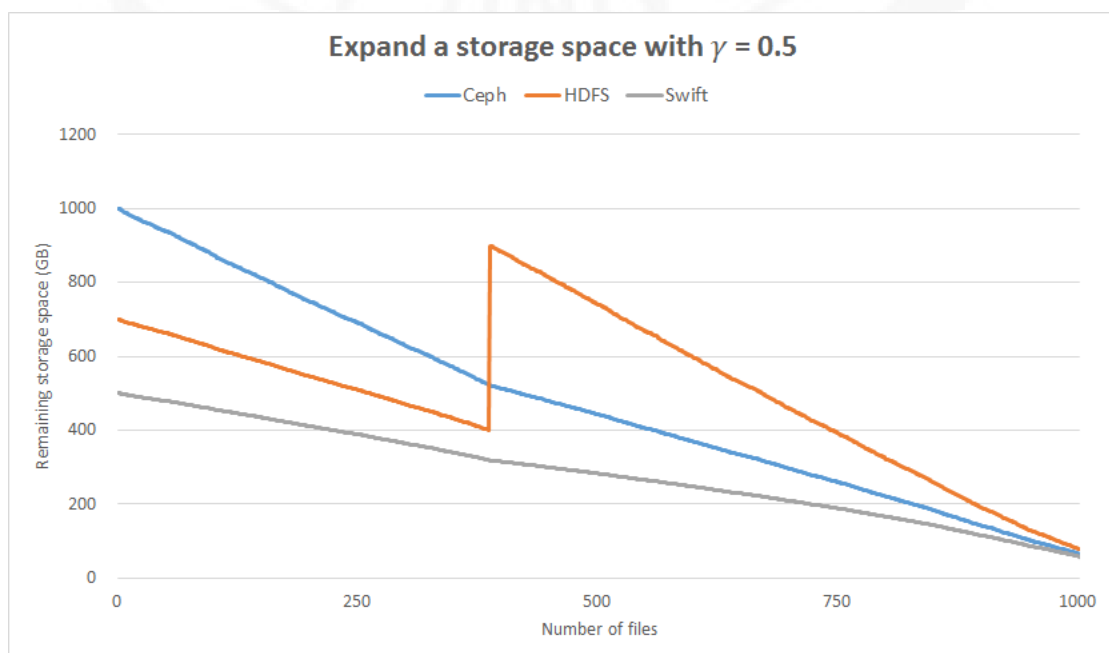
will show what kind of distribution. In next experiment, we given Ceph 1000 Gigabyte, HDFS 900 Gigabyte, Swift 800 Gigabyte capacity, $\gamma = 0.9$ and 1000 random size file around 1 Mb to 4 Gb upload to our system. The result is shown in Figure 4.6.

As the result above, we can see that the storage platform's capacity will become

FIGURE 4.6: Similar capacity distribution with $\gamma = 0.9$

the same. After that, the γ value will not be important for the file split. That means the split data will become an equal distribution.

In a cloud storage system, adding new storage nodes is a frequently occurring situation. We designed a case that expands our storage space during the experiment. The result is shown in Figure 4.7.

FIGURE 4.7: Expand a storage space with $\gamma = 0.5$

As above experiment, we add a additional 500 Gigabyte storage space on HDFS when its storage space close to the 400 Gigabyte. we can can be found that when the storage space increase, its reduction ratio has been changed. It proves that our system will be based on the current storage capacity of the system to split upload files.

Finally, we have compare our system upload time with the single storage system. The result is shown in Figure 4.8.

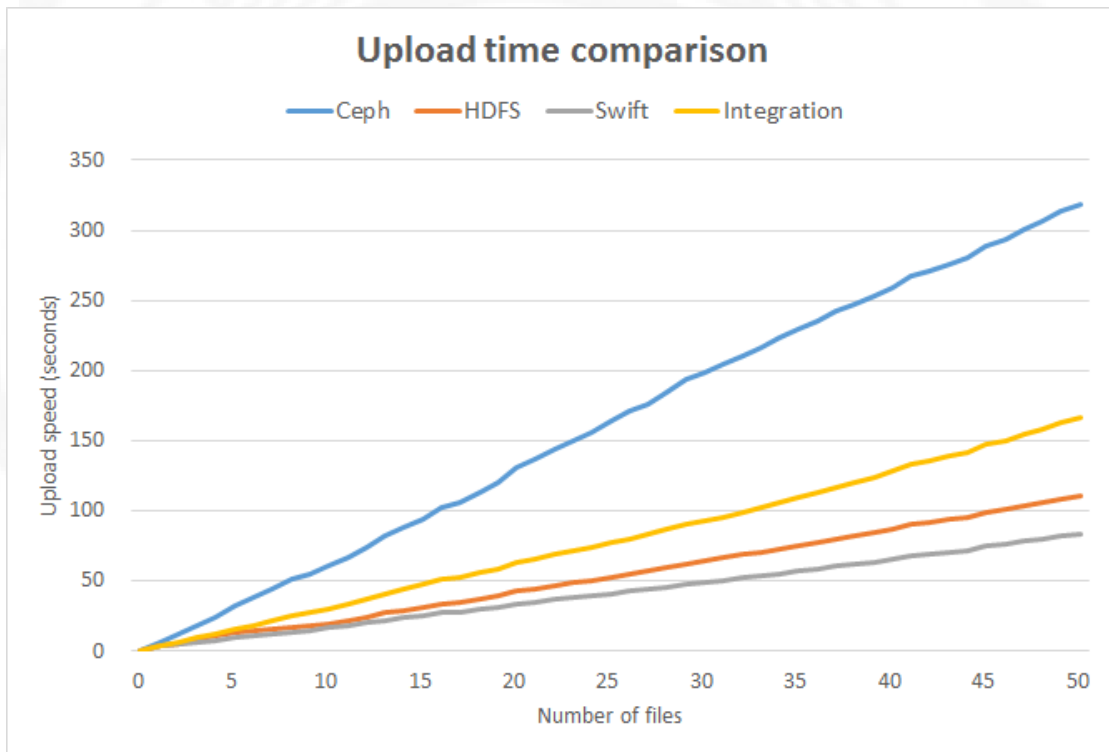


FIGURE 4.8: Upload time comparison

This diagram data is from the Figure 4.5. We sorting the data size and pick 50 piece of data with 20 piece of data gap. This experiment clearly showing that the complex upload speed will be homogenized by sub-system's upload speed.

4.3 User Interface

We provided an user interface in our system. The user interface overview is shown in Figures 4.9. There are four part of methods in this interface: *Admin*, *Account*,

Overview and *Trash can*. Each parts a main page include several functions. First of all, we provide a simple account verification for web login. After verification, user will entering the *Overview*.

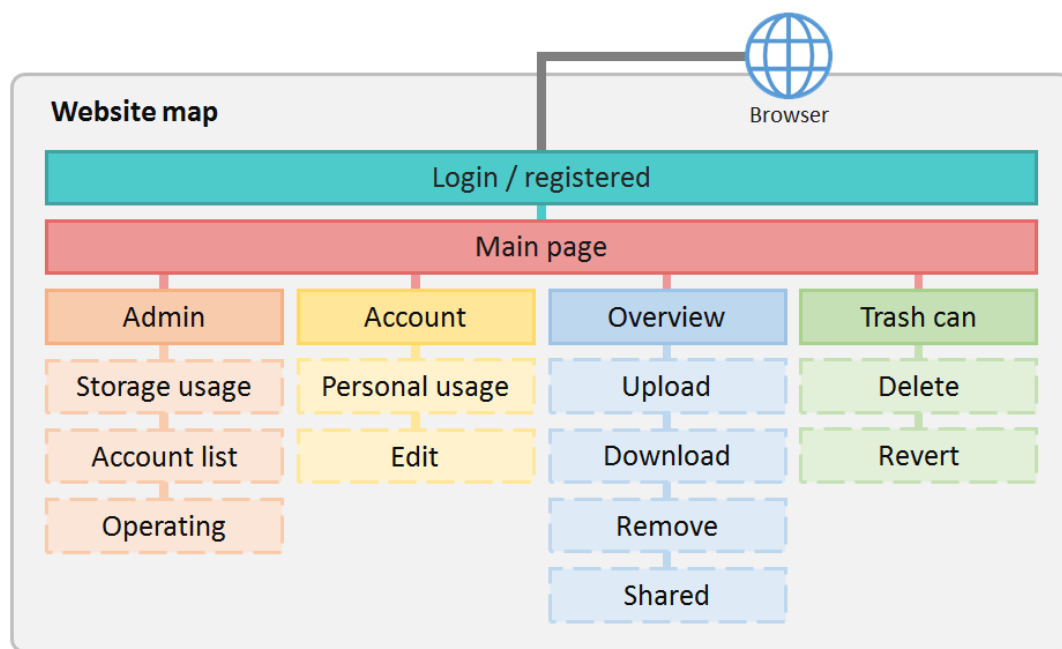


FIGURE 4.9: User interface overview

Overview

Overview is the major part in this system, as shown in Figures 4.10. It includes following basic operations: upload, download, remove, and shared folder. Also, we provide a file link for user to download file without login. We use the AJAX, JQuery and Bootstrap to implement upload. That means the web can get the upload progress and show the percentage of the progress bar to user. It can enhance the user experience when uploading the large file. The interface also provided Multi-file upload feature.

Trash can

Trash can let user recovery deleted files, as shown in Figures 4.11. In this page, user can delete files manually to free their storage space. If they did not delete

files. The system will automatically delete files after seven days.

HPC Software Defined Storage

管理員選單 儲存空間 垃圾桶 admin

儲存空間

名稱	大小	上傳時間	刪除	分享
圖片素材				
實驗數據				
template.docx	41.67 KB	2016-06-17 15:13:59		
甄選通知書.pdf	197.23 KB	2016-06-17 15:14:45		
OpenStack-liberty-RDO-installation.txt	482 bytes	2016-06-17 15:15:46		
THU01.jpg	359.4 KB	2016-06-17 15:16:11		
Biocity_demo.mp4	7.75 MB	2016-06-17 15:21:51		

© HPC Lab. Dept. of CS, Tunghai University 2016

FIGURE 4.10: Overview page

Account






















Account shows the user information, in which the user can also edit the user account, such as user name, password and E-mail, as shown in Figures 4.12.

Admin

Admin page is only for system administrator to visit. It provides three part of functions: *Storageusage*, *Storage pool statue* and *user list*, as shown in Figures 4.13. *Storageusage* let administrator know the total storage capacity, *Storage pool statue* can see each storage nodes and their disks statue, as shown in Figures 4.14(a), *user list* allow administrator to observed the user usage, as shown in Figures 4.14(b).

HPC Software Defined Storage 管理員選單 儲存空間 垃圾桶 admin ▾

垃圾桶

名稱	大小	上傳時間	還原	刪除
 file.png	1.95 KB	2016-06-17 15:12:35		
 archive.png	2.99 KB	2016-06-17 15:12:35		
 MSexcel.png	2.23 KB	2016-06-17 15:12:35		
 audio.png	6.09 KB	2016-06-17 15:12:35		
 MSppt.png	3.67 KB	2016-06-17 15:12:35		
 image.png	2.92 KB	2016-06-17 15:12:35		
 MSword.png	3 KB	2016-06-17 15:12:35		

© HPC Lab. Dept. of CS, Tunghai University 2016

FIGURE 4.11: Trash can page

HPC Software Defined Storage 儲存空間 垃圾桶 YS ▾

個人設定

YS [變更](#)
一般用戶

個人電子郵件
a8118fz@hotmail.com
[變更電子郵件](#)

[變更密碼](#)
[刪除帳戶](#)

⚙️ 儲存空間使用量 (19.81%)

■ 已使用: 1014.11 MB ■ 未使用: 4.01 GB

FIGURE 4.12: Account page

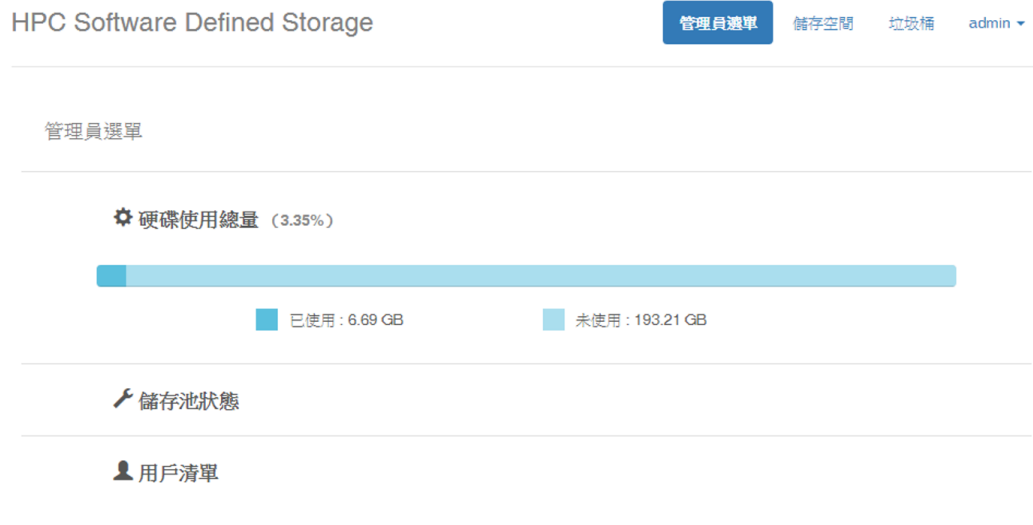


FIGURE 4.13: Admin page



(a) Storage pool statue

(b) User list

FIGURE 4.14: Admin functions

Chapter 5

Conclusions and Future Work

This work has built a cloud system to integrate heterogeneous storage platforms based on the concept of software-defined storage. For the file storage, we propose a method which support uniform data distribution to achieve storage resources load balance. Finally, we provide a web platform which support all type of storage space. This platform provides a high usability user interface to let user use this system more friendly.

5.1 Concluding Remark

We have established a heterogeneous storage system that integrates three different kind of software-defined storage kit. In addition to evaluating each of the storage kit, we can also stimulate the local system convergence heterogeneous public cloud storage conditions by using this environment.

For the file storage mechanism, we proposed a method which supporting uniform data distribution. The γ is an user-defined value which influences the storage space convergence speed. The larger γ value, the faster storage space convergence speed. This method allow user using, adding different size of storage space and the final storage resources can achieve storage load balancing.

We also provide a high usability user interface. This interface is designed as a web application and based on RESTful architecture. Therefore, no matter which kind of device, we can provide the user a good use of screen. To enhance the user experience, we also optimizing the user interface by using asynchronous JavaScript and XML technology. Thereby making the web application can update the content without the redirect, reduce resource load by refresh all the page.

5.2 Future Works

In our system, due to the lack of hardware resources quantity. Our storage cluster is building by virtualization technology. We hope to have the opportunity to use more physical machine environment. Then we can build more large storage system in the future.

The back-end storage system we currently using is limited to open source lit. We have built a platform supporting most of storage system. Therefore, we hope to develop our system including public and private cloud as a hybrid cloud storage environment.

For the file storage function, we hope to use erasure code to improve security and availability of our system and provide more useful features for file operations in our user interface.

References

- [1] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *2008 Grid Computing Environments Workshop*, pages 1–10. IEEE, 2008.
- [2] Mark D. Ryan. Cloud computing security: The scientific challenge, and a survey of solutions. In *Journal of Systems and Software*, pages 2263–2268, 2013.
- [3] Alessio Botta, Walter de Donato, Valerio Persico, and Antonio Pescapé. Integration of cloud computing and internet of things: A survey. *Future Generation Computer Systems*, 56:684–700, 2016.
- [4] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4): 50–58, 2010.
- [5] Chengzhang Peng and Zejun Jiang. Building a cloud storage service system. *Procedia Environmental Sciences*, 10, Part A(0):691–696, 2011.
- [6] Josef Spillner, Johannes Müller, and Alexander Schill. Creating optimal cloud storage systems. *Future Generation Computer Systems*, 29, Issue 4:1062–1072, 2013.
- [7] Cong Wang, Qian Wang, Kui Ren, and Wenjing Lou. Privacy-preserving public auditing for data storage security in cloud computing. In *INFOCOM, 2010 Proceedings IEEE*, pages 1–9. IEEE, 2010.

- [8] Margaret Rouse. Software-defined storage. <http://searchsdn.techtarget.com/definition/software-defined-storage>, 2013.
- [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] Gauri Joshi, Emina Soljanin, and Gregory Wornell. Efficient redundancy techniques for latency reduction in cloud systems. *arXiv preprint arXiv:1508.03599*, 2015.
- [12] Hong Xia Mao, Xiao Ling Shu, Kun Huang, and Li Zhang. Research of data reliability technology based on erasure code redundancy technology in cloud storage. In *Advanced Materials Research*, volume 912, pages 1345–1348. Trans Tech Publ, 2014.
- [13] Santosh Kumar Majhi and Sunil Kumar Dhal. Placement of security devices in cloud data centre network: Analysis and implementation. *Procedia Computer Science*, 78:33–39, 2015.
- [14] Manish M. Potey, C.A. Dhote, and Deepak H. Sharma. Homomorphic encryption for security of cloud data. *Procedia Computer Science*, 79:175–181, 2016.
- [15] Stephen Soltész, Herbert Pötzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson. Container-based operating system virtualization: a scalable, high-performance alternative to hypervisors. In *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, volume 41, Issue 3, pages 275–287, 2007.
- [16] R. Uhlig, G. Neiger, D. Rodgers, and A. L. Santoni. Intel virtualization technology. *Computer*, 38, Issue 5:48–56, 2005.

- [17] Jyotiprakash Sahoo, Subasish Mohapatra, and Radha Lath. Virtualization: A survey on concepts, taxonomy and associated security issues. *Second International Conference on Computer and Network Technology*, pages 222–226, 2010.
- [18] Joe Arnold. *OpenStack Swift: Using, Administering, and Developing for Swift Object Storage*. O’Reilly Media, 2014.
- [19] Sage A. Weil, Scott A. Brandt, Ethan L. Miller, Darrell D. E. 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.
- [20] Sage A. Weil, Andrew W. Leung, Scott A. Brandt, and Carlos Maltzahn. Rados: a scalable, reliable storage service for petabyte-scale storage clusters. *Proceedings of the 2nd international workshop on Petascale data storage: held in conjunction with Supercomputing*, pages 35–44, 2007.
- [21] Hadoop hdfs. https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html, 2016.
- [22] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *2010 IEEE 26th Symposium on Mass Storage Systems and Technologies*, 2010.
- [23] Openstack. <https://www.openstack.org/>, 2016.
- [24] Omar Sefraoui, Mohammed Aissaoui, and Mohsine Eleuldj. Openstack: Toward an open-source solution for cloud computing. *International Journal of Computer Applications*, 55(3), 2012.
- [25] 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.

- [26] Jianqing Fan and Qiwei Ya. Spline methods. In *Nonlinear time series: non-parametric and parametric methods*. Springer Science and Business Media, 2005.
- [27] Cubic spline. <http://mathworld.wolfram.com/CubicSpline.html>, 2016.
- [28] RH Bartels, JC Beatty, and BA Barsky. Hermite and cubic spline interpolation. *An Introduction to Splines for Use in Computer Graphics and Geometric Modelling*, pages 9–17, 1998.
- [29] Chengzhang Peng and Zejun Jiang. Building a cloud storage service system. *Procedia Environmental Sciences*, 10:691–696, 2011.
- [30] Suzhen Wu, Kuan-Ching Li, Bo Mao, and Minghong Liao. Dac: Improving storage availability with deduplication-assisted cloud-of-clouds. *Future Generation Computer Systems*, 2016.
- [31] Yu-Chuan Shen, Chao-Tung Yang, Shuo-Tsung Chen, and Wei-Hsun Cheng. Implementation of software-defined storage service with heterogeneous object storage technologies. In *ASE BigData and SocialInformatics 2015*, volume 29, Issue 4. ACM, 2015.

Appendix A

OpenStack Installation

I. Network Time Protocol (NTP)

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

II. Database (Controller node setup)

```
$ sudo apt-get install python-mysqldb mysql-server
#==== MySQL configure ====
[mysqld]
...
bind-address = CONTROLLER_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] N
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_IP/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_IP:35357/v2.0
$ keystone user-create --name=admin --pass=ADMIN_PASS
$ keystone role-create --name=admin
$ keystone tenant-create --name=admin --description="Admin Tenant"
$ 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"
$ 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_IP:5000/v2.0 \
  --internalurl=http://CONTROLLER_IP:5000/v2.0 \
  --adminurl=http://CONTROLLER_IP:35357/v2.0
$ unset OS_SERVICE_TOKEN OS_SERVICE_ENDPOINT
$ keystone --os-tenant-name admin --os-username admin --os-password ADMIN_PASS \
  --os-auth-url http://CONTROLLER_IP:35357/v2.0 token-get
$ keystone --os-tenant-name admin --os-username admin --os-password ADMIN_PASS \
  --os-auth-url http://CONTROLLER_IP:35357/v2.0 tenant-list
$ keystone user-list
$ keystone user-role-list
##### 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://CONTROLLER_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_IP/glance
[keystone_authtoken]
auth_uri = http://CONTROLLER_IP:5000
auth_host = CONTROLLER_IP
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
$ 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_IP:9292 \
  --internalurl=http://CONTROLLER_IP:9292 \
  --adminurl=http://CONTROLLER_IP:9292
$ service glance-registry restart
$ service glance-api restart

$ mkdir /tmp/images
$ wget -P /tmp/images http://download.cirros-cloud.net/0.3.4/cirros-0.3.4-x86_64-disk.img
$ source admin-openrc.sh
$ glance image-create --name "cirros-0.3.4-x86_64" --disk-format qcow2 \
  --container-format bare --is-public True --progress < cirros-0.3.4-x86_64-disk.img

```

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_IP
rabbit_password = RABBIT_PASS
...
my_ip = CONTROLLER_IP
vncserver_listen = CONTROLLER_IP
vncserver_proxyclient_address = CONTROLLER_IP
...
auth_strategy = keystone
[keystone_authtoken]
...
auth_uri = http://CONTROLLER_IP:5000
auth_host = CONTROLLER_IP
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = nova
admin_password = NOVA_PASS
[database]
connection = mysql://nova:NOVA_DBPASS@CONTROLLER_IP/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
$ 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_IP:8774/v2/%(tenant_id)s \
  --internalurl=http://CONTROLLER_IP:8774/v2/%(tenant_id)s \
  --adminurl=http://CONTROLLER_IP:8774/v2/%(tenant_id)s
$ service nova-api restart
$ service nova-cert restart
$ service nova-consoleauth restart
$ service nova-scheduler restart
$ service nova-conductor restart
$ service nova-novncproxy restart

```

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_IP
rabbit_password = RABBIT_PASS
...
my_ip = COMPUTER_IP
vnc_enabled = True
vncserver_listen = 0.0.0.0
vncserver_proxyclient_address = COMPUTER_IP
novncproxy_base_url = http://CONTROLLER_IP:6080/vnc_auto.html
...
glance_host = CONTROLLER_IP
[database]
# The SQLAlchemy connection string used to connect to the database
connection = mysql://nova:NOVA_DBPASS@CONTROLLER_IP/nova
[keystone_auth_token]
auth_uri = http://CONTROLLER_IP:5000
auth_host = CONTROLLER_IP
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
```

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 boot --flavor m1.tiny --image cirros-0.3.4-x86_64 --nic net-id=DEMO_NET_ID \
  --security-group default --key-name demo-key demo-instance1
Permit ICMP (ping):
$ nova secgroup-add-rule default 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
```

Appendix B

Ceph Installation

I. Create Ceph user

```
# ssh user@ceph-mon
# sudo useradd -d /home/ceph -m ceph
# sudo passwd ceph
```

II. Add root competence for every Ceph cluster

```
#echo "ceph ALL = (root) NOPASSWD:ALL" | sudo tee /etc/sudoers.d/ceph
#sudo chmod 0440 /etc/sudoers.d/ceph
```

III. Set SSH no password login and copy to every node

```
# ssh-keygen -t rsa -f ~/.ssh/id_rsa -P ""
# cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
# ssh-copy-id ceph@mon
# ssh-copy-id ceph@osd1
# ssh-copy-id ceph@osd2
```

IV. Install Ceph

```
$ 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-jewel/ $(lsb_release -sc) main
| sudo tee /etc/apt/sources.list.d/ceph.list
$ sudo apt-get update && sudo apt-get install -y ceph-deploy
```

V. Storage Cluster Quick Start

```
$ mkdir ~/ceph && cd ~/ceph
$ ceph-deploy new {mon-nodes}
$ sed -i '$a osd pool default size = 2' ceph.conf
ceph-deploy install --release jewel {mon-node} {osd1-node} {osd2-node}

$ ceph-deploy mon create-initial

$ ssh {OSD-node1}
$ sudo mkdir /var/local/osd0
$ sudo chown ceph:ceph /var/local/osd0
$ exit

$ ssh {OSD-node2}
$ sudo mkdir /var/local/osd1
$ sudo chown ceph:ceph /var/local/osd1
$ exit

$ ceph-deploy osd prepare {OSD-node1}:/var/local/osd0 {OSD-node2}:/var/local/osd1
$ ceph-deploy osd activate {OSD-node1}:/var/local/osd0 {OSD-node2}:/var/local/osd1

$ ceph-deploy admin {all nodes}
$ sudo chmod +r /etc/ceph/ceph.client.admin.keyring

$ ceph health
HEALTH_OK
```

Appendix C

Hadoop Installation

I. Modify hosts and hostname

```
# sudo vim /etc/hosts
# sudo vim /etc/hostname
```

II. Install Java JDK

```
# sudo apt-get -y install openjdk-7-jdk
# sudo ln -s /usr/lib/jvm/java-7-openjdk-amd64 /usr/lib/jvm/jdk
```

III. Add hadoop user

```
# sudo addgroup hadoop
# sudo adduser --ingroup hadoop hduser
# sudo adduser hduser sudo
```

IV. Creat SSH authentication login

```
# ssh-keygen -t rsa -f ~/.ssh/id_rsa -P ""
# cat ~/.ssh/id_rsa.pub >> ~/.ssh/authorized_keys
```

V. Download hadoop

```
# wget http://ftp.twaren.net/Unix/Web/apache/hadoop/common/hadoop-2.7.1/hadoop-2.7.1.tar.gz
# tar zxf hadoop-2.7.1.tar.gz
# mv hadoop-2.7.1.tar.gz hadoop
```

VI. Add the environment variable


```
# vim .bashrc
```

```
export JAVA_HOME=/usr/lib/jvm/jdk/  
export HADOOP_INSTALL=/home/hduser/hadoop  
export PATH=$PATH:$HADOOP_INSTALL/bin  
export PATH=$PATH:$HADOOP_INSTALL/sbin  
export HADOOP_MAPRED_HOME=$HADOOP_INSTALL  
export HADOOP_COMMON_HOME=$HADOOP_INSTALL  
export HADOOP_HDFS_HOME=$HADOOP_INSTALL  
export YARN_HOME=$HADOOP_INSTALL
```

VII. Set hadoop config

a. edit hadoop-env.sh

```
# cd hadoop/etc/hadoop  
# vim hadoop-env.sh
```

```
export JAVA_HOME=/usr/lib/jvm/jdk/
```

b. edit core-site.xml

```
# vim core-site.xml
```

```
<property>  
  <name>fs.default.name</name>  
  <value>hdfs://hadoop1-master:9000</value>  
</property>
```

c. edit yarn-site.xml

```
# vim yarn-site.xml
```

```
<property>  
  <name>yarn.nodemanager.aux-services</name>  
  <value>mapreduce_shuffle</value>  
</property>  
<property>  
  <name>yarn.resourcemanager.hostname</name>  
  <value>hadoop1</value>  
</property>
```

d. edit mapred-site.xml

```
# cp mapred-site.xml.template mapred-site.xml  
# vim mapred-site.xml
```

```
<property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
</property>
```

e. edit hdfs-site.xml

```
# mkdir -p ~/mydata/hdfs/namenode
# mkdir -p ~/mydata/hdfs/datanode
# vim hdfs-site.xml
```

```
<property>
  <name>dfs.replication</name>
  <value>2</value>
</property>
<property>
  <name>dfs.namenode.name.dir</name>
  <value>/home/hduser/mydata/hdfs/namenode</value>
</property>
<property>
  <name>dfs.datanode.data.dir</name>
  <value>/home/hduser/mydata/hdfs/datanode</value>
</property>
```

f. edit slaves

```
# vim slaves
```

```
hadoop1
hadoop2
hadoop3
```

VIII. Copy hadoop to all nodes

```
# scp -r /home/hduser/hadoop hadoop1:/home/hduser
# scp -r /home/hduser/hadoop hadoop2:/home/hduser
# scp -r /home/hduser/hadoop hadoop3:/home/hduser
```

IX. Format HDFS

```
# hdfs namenode -format
```

X. Start hadoop

```
# start-all.sh
```

XI. Use jps to see java running program

```
# jps
```

XII. MapReduce JobTracker monitoring website

```
# hadoop1:50030
```



Appendix D

Swift Installation

I. Install and configure the controller node

```
$ source admin-openrc.sh
$ openstack user create --password-prompt swift
$ openstack role add --project service --user swift admin
$ openstack service create --name swift \
  --description "OpenStack Object Storage" object-store
$ openstack endpoint create \
  --publicurl 'http://CONTROLLER_IP:8080/v1/AUTH_%(tenant_id)s' \
  --internalurl 'http://CONTROLLER_IP:8080/v1/AUTH_%(tenant_id)s' \
  --adminurl http://CONTROLLER_IP:8080 \
  --region RegionOne \
  object-store
```

II. To install and configure the controller node components

```
# apt-get install swift swift-proxy python-swiftclient \
  python-keystoneclient python-keystonemiddleware memcached
# mkdir /etc/swift
# curl -o /etc/swift/proxy-server.conf \
  https://git.openstack.org/cgit/openstack/swift/plain/etc/proxy-server.conf-sample?h=stable/kilo
# vim /etc/swift/proxy-server.conf
[DEFAULT]
bind_port = 8080
user = swift
swift_dir = /etc/swift
...
[pipeline:main]
pipeline = catch_errors gatekeeper healthcheck proxy-logging cache container_sync
```

```

bulk ratelimit authtoken keystoneauth container-quotas account-quotas slo dlo
proxy-logging proxy-server
...
[app:proxy-server]
account_autocreate = true
...
[filter:keystoneauth]
use = egg:swift#keystoneauth
operator_roles = admin,user
...
[filter:authtoken]
paste.filter_factory = keystonemiddleware.auth_token:filter_factory
auth_uri = http://CONTROLLER_IP:5000
auth_url = http://CONTROLLER_IP:35357
auth_plugin = password
project_domain_id = default
user_domain_id = default
project_name = service
username = swift
password = SWIFT_PASS
delay_auth_decision = true
...
[filter:cache]
memcache_servers = 127.0.0.1:11211

```

III. Install and configure the storage nodes

```

# apt-get install xfsprogs rsync
# mkfs.xfs /dev/sdb1
# mkfs.xfs /dev/sdc1
# mkdir -p /srv/node/sdb1
# mkdir -p /srv/node/sdc1

# vim /etc/fstab
/dev/sdb1 /srv/node/sdb1 xfs noatime,nodiratime,nobarrier,logbufs=8 0 2
/dev/sdc1 /srv/node/sdc1 xfs noatime,nodiratime,nobarrier,logbufs=8 0 2

# mount /srv/node/sdb1
# mount /srv/node/sdc1

# 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

```

IV. Install and configure storage node components

```

# apt-get install swift swift-account swift-container swift-object
# curl -o /etc/swift/account-server.conf \
  https://git.openstack.org/cgit/openstack/swift/plain/etc/account-server.conf-sample?h=stable/kilo
# curl -o /etc/swift/container-server.conf \
  https://git.openstack.org/cgit/openstack/swift/plain/etc/container-server.conf-sample?h=stable/kilo
# curl -o /etc/swift/object-server.conf \
  https://git.openstack.org/cgit/openstack/swift/plain/etc/object-server.conf-sample?h=stable/kilo
# curl -o /etc/swift/container-reconciler.conf \
  https://git.openstack.org/cgit/openstack/swift/plain/etc/container-reconciler.conf-sample?h=stable/kilo
# curl -o /etc/swift/object-expirer.conf \
  https://git.openstack.org/cgit/openstack/swift/plain/etc/object-expirer.conf-sample?h=stable/kilo

# vim /etc/swift/account-server.conf
[DEFAULT]
bind_ip = CONTROLLER_IP
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 = CONTROLLER_IP
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 = CONTROLLER_IP
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
recon_lock_path = /var/lock

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

V. Create initial rings

```
# cd /etc/swift

# swift-ring-builder account.builder create 10 3 1
# swift-ring-builder account.builder \
  add r1z1-CONTROLLER_IP:6002/DEVICE_NAME DEVICE_WEIGHT
# swift-ring-builder account.builder rebalance
```

```
# swift-ring-builder container.builder create 10 3 1
# swift-ring-builder container.builder \
  add r1z1-CONTROLLER_IP:6001/DEVICE_NAME DEVICE_WEIGHT
# swift-ring-builder container.builder rebalance

# swift-ring-builder object.builder create 10 3 1
# swift-ring-builder object.builder \
  add r1z1-CONTROLLER_IP:6000/DEVICE_NAME DEVICE_WEIGHT
# swift-ring-builder object.builder rebalance

(Copy the account.ring.gz, container.ring.gz, and object.ring.gz files to the /etc/swift directory on each storage node.)
```

VI. Finalize installation

```
# curl -o /etc/swift/swift.conf \
  https://git.openstack.org/cgit/openstack/swift/plain/etc/swift.conf-sample?h=stable/kilo
# vim /etc/swift/swift.conf
[swift-hash]
swift_hash_path_suffix = HASH_PATH_SUFFIX
swift_hash_path_prefix = HASH_PATH_PREFIX
...
[storage-policy:0]
name = Policy-0
default = yes

(Copy the swift.conf file to the /etc/swift directory on each storage node)

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