國立交通大學

資訊科學與工程研究所

碩 士 論 文

Openstack 與 Hadoop 整合與研究

An integration and application of OpenStack and Hadoop

研 究 生：方智誼

指導教授：蔡錫鈞 教授

中 華 民 國 一 百 零 三 年 一 月


E S A
1896

Openstack 與 Hadoop 整合與研究

An integration and application of OpenStack and Hadoop

研 究 生：方智誼　　Student : Jhih-Yi Fang

指導教授：蔡錫鈞　　Advisor : Shi-Chun Tsai

國立交通大學

資訊科學與工程研究所

碩士論文

A Dissertation

Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science and Engineering

January 2014

Hsinchu, Taiwan, Republic of China

中華民國一百零三年一月


E S A
1896

摘　　要

近年來網際網路應用的發展，使得雲端計算變得是一項熱門的技術，透過使用雲端的資源，來達到各種不同的網路應用，而 Amazon Elastic MapReduce(EMR) 是目前知名的商業雲端計算服務，可以透過服務建立出各種不同規模的 Hadoop Cluster 以進行 Hadoop MapReduce 的計算，可用來進行多種類型的應用，如:Log 分析、網站索引、資料倉儲、機器學習... 等，因此將如何使用雲端資源將會是值得研究的課題。

目前 Amazon Elastic MapReduce 雖然提供了一個整合 Hadoop 服務的使用方式，但對於已既有設備的公司或學校單位，要想架設此類型的服務仍尚缺軟體上的支援，因此本篇論文最主要的目的是透過各種不同的 OpenSouce 整合，利用 OpenStack、Hadoop、Ceph 來提供一個類似於 EMR 服務的平台出來，以提供大家可以透過介面的使用，可以很迅速的在 OpenStack 上架設出 Hadoop Cluster，並且也能透過介面來進行 Hadoop Cluster 的管理與操作，在 Cluster 中也整合了 Object Storage 的儲存服務，可用來提供給 Hadoop MapReduce 計算使用，本篇論文已透過本校的 OpenStack 平台上完成架設，並且進行測試實驗以檢驗 Instance 與 Object Storage 的服務品質，除此之外透過本篇論文的設計方式，也可將介面程式碼進行 OpenSource，使得其他人可取得程式碼後進行客製化的修改與使用。


E S A
1896

Abstract

Cloud computing has become a popular IT technology in recent years. Applications can be built to provide elastic services through network. Amazon is the most famous vender to provide cloud services. Elastic MapReduce (EMR) is one of the most popular services provided by Amazon, where users can rent and build scalable Hadoop cluster. The cluster can be used in various of applications, such as log analysis, web indexing, data warehousing, machine learning, etc.

In this thesis, based on the open sources OpenStack and Hadoop, we integrate both technologies and provide an EMR service on OpenStack. Users can set up and run a Hadoop cluster via the GUI interface. Usually, Hadoop program is handled with command line approach. Along the way, (1) we use the Ceph technology to support object storage, such that our Hadoop cluster can process really big data and the results can be stored even after the cluster is released; (2) we use S3 storage for the cluster communication and authentication; (3) we have modified the Dashboard of OpenStack to provide better user interface. Based on 4 servers, our experimental results show that our platform can process data up to 1T within few hours. The capacity can be further improved with hardware upgraded and we leave it as a future work.


E S A
1896

致　謝

進來實驗室兩年了，在這段期間中得到了許多人的教導與照顧，過程中學習到了許多東西，其中要先感謝蔡錫鈞老師的教導，在經過老師的授課後，我才在這裡見識到什麼叫『數學』，讓我明白以前在課堂所學會的，都只能說是『算術』而已，世界上存在著許多詭妙的問題與解決方式，認知到不是所有問題都可以解決，也同時明白還有許多問題等著我們去解決，除此之外還要感謝老師能夠給我機會在校計中協助校務系統與 OpenStack 的維護，也因此開始了 OpenStack 與 Hadoop 整合進行討論，從原本認為的『不可能』到現在成果的使用，學到許多 OpenSource 的開發與使用方式，最後才能夠順利的完成這篇論文的撰寫。

接下來要感謝的是實驗室的學長們，首先感謝名全學長在實驗室裡總是能夠回答我所提問的『數學』問題，才能夠導正我一些『數學』上的認知錯誤，還要感謝宜謙、權昱學長與同樣是系計中的助教們，經過你們的『傳承』，我才能夠見識到各種不同 Cluster 的使用方式，因此我才能對此篇論文進行研究，在此還要感謝韜瑋能夠在我遇到一些大大小小的問題時，不管是修課上、數學上、報告上、日常上以及一些奇奇怪怪的看法上，還能夠與我進行討論，另外我也要感謝俊憲與奕任學長、裕堡、煥博、上全、大慶、韋翔學弟能夠在 meeting 時，能夠適時地指出我報告的問題所在，最後還要感謝王協源老師與陳昌盛老師，能夠在我參加 OpenFlow 與 BotNet meeting 報告時能夠給予我指導，因此我才能夠對網路與系統能夠有更深入的了解與研究。


E S A
1896

目錄

表目錄

圖目錄

第 一 章　緒論

近年來由於網際網路迅速的發展，使得人人可以透過使用各處網路服務來得到便捷的資訊，而這些網路服務的提供者，往往是擁有著各自的機房來運行著這些服務的設備，而這些機房中裏頭可能有著數十台甚至到數百台以上的設備，隨時需要有人維護確保設備能夠正常的運作，同時虛擬化技術也逐漸進步與發展，使得一台 Computer 上頭可以有效率的運行多個不同類型的 OS，分別提供給不同使用者使用，硬體資源可以被有效的分享與利用，因此雲端運算從只能運行特定的應用程式，逐漸發展為能夠運行特定的 OS，直到 2006 年 Amazon 開始提供 Elastic Compute Cloud(EC2)[1] 服務後，從此要運行網路服務有了更靈活的作法，能夠透過 EC2 服務架設出自己所欲提供的網路服務，並且能夠隨時控制所需欲執行的機器數量，使得服務提供者不必再擁有大量的實體設備，便能夠建構出如同規模的環境，因此該如何架設如同 EC2 般的服務與使用，將會是推進雲端運算技術發展的重要課題。

1.1　研究背景與動機

由於開放式原始碼 OpenStack 的發展，使得人人有機會可以打造自己的私有雲，OpenStack 提供了許多 API，可讓使用者控制私有雲上的操作，以達到更彈性的擴增或縮減所使用資源，因此要如何在私有雲上管理與使用便成了一個值得研究的問題。

Hadoop 是一個分散式計算工具，可提供使用者在多台電腦上建立一個分散式的儲存系統，並且可利用這儲存系統進行自己所需運行的 MapReduce 程式，由於這工具是一個需要在多台系統上進行安裝，並且可能會依使用者的需求，需在系統上安裝其他的函數庫或工具，因此有可能會因執行的程式不同，在不同 Hadoop Cluster 環境下，導致造成程式無法正常執行，在安裝執行環境上就成了一項問題，因此若能利用 OpenStack 私有雲的特性，客製化建立一個專屬的 Cluster，來運行自己所需的程式，將會是進行雲端運算的一種方式。

目前在雲端運算上，以 Amazon Elastic MapReduce 為最出名的商業服務，可以透過雲端的資源使用，進行海量資料的程式計算，但是對於已有主機設備的公司或學校，想要進行這樣的服務使用，目前仍尚缺軟體上的支持，因此本論文透過 OpenSource 的軟體，來提供類似這樣的服務使用，將可對設備進行有效的資源分享與使用。

1.2　研究目標與成果

本論文的研究目標是希望可以透過 OpenStack 的環境，提供出一個類似 Amazon Elastic MapReduce(EMR) 的服務出來，目前已有成果在本校計中所提供的 OpenStack 平台上運行，能夠透過瀏覽器介

面的操作，迅速架設出不同規模的 Hadoop Cluster 環境，並且整合了 S3 Object Storage 功能，能夠利用 S3 API Key 進行操作，因此也在其他機器上透過其他 S3 API 工具進行資料的儲存與蒐集，再透過 Hadoop Cluster 進行 MapReduce 的運算，透過本篇論文的設計，使用者也可進行控制介面的架設與修改，將可對 Cluster 的啟動流程進行調整，而其中所使用的可客製化介面原始碼也已在 GitHub 上進行公開，因此除了直接透過本校所提供的介面外，也可從 GitHub 取得原始碼後進行修改與使用，另外本篇論文也在本校的 OpenStack 與 Object Storage 透過 Hadoop MapReduce 進行效能測試，以檢驗目前 OpenStack 與 Object Storage 上的執行效率為何。

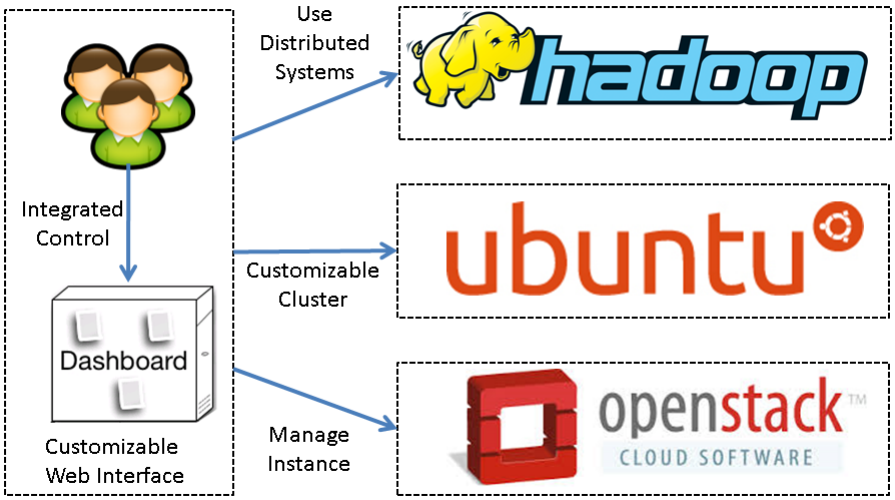

圖 1.1: 整合控制介面設計

1.3 各章節介紹

本篇論文將在第二章將介紹目前現有的相關技術，包含 OpenStack、Hadoop 與 Ceph 所提供的 Object Storage，而在第三章將會介紹系統的實作構想與問題，在實作的部分會在第四章進行說明，呈現的結果與測試會在第五章介紹，最後在第六章作一個總結。

第 二 章　背景知識

2.1 OpenStack

OpenStack 是一個雲端作業系統 (Cloud Operating System)，由美國太空總署和 Rackspace 在 2010 年 6 月提出 [2]，主要是用來提供一個 IaaS[3] 環境，可用來建置公有雲或私有雲的服務，OpenStack 包含了許多子專案，主要是以 Python 語言為主，是一系列 Apache 授權的 OpenSource。

在小型 Web Service 中，通常程式間的溝通只會有資料庫與 Web Server(如:Apache HTTP Server) 之間，因此在架設時只須考慮到資料庫、Web Server、動態網頁語言 (如:php、aspx、jsp) 之間上關係的即可，而 OpenStack 是考慮在資料中心規模上的環境下 (Data Center)，因此所有元件設計成可分開運行與服務，其中在 Dashboard 使用了 Django Web Framework[4] 設計提供瀏覽器的操作介面，在 API Service 上則使用了 Paste Deployment[5] 與 Eventlet Concurrent Networking Library[6]，來分別運行與提供 REST (Representational State Transfer)[8] 服務，因此所有元件在開發同時也都符合了 Python 所定義的 WSGI(Web Server Gateway Interface) 規範 [7] 來進行，另外 API Service 在進行內部程式的溝通，會使用 AMQP(Advanced Message Queuing Protocol)[9] 來進行 RPC(remote procedure call) 的動作，因此使得結構上相對於一般的 Web Service 複雜了許多。

使用者主要可以利用 Dashboard 或者是 API 使用 OpenStack 所提供的資源，其資源主要可分為網路、計算、儲存三種類型 (圖 2.1)，管理者可定義使用者的 Quota 以限制使用者在 OpenStack 上使用的資源。

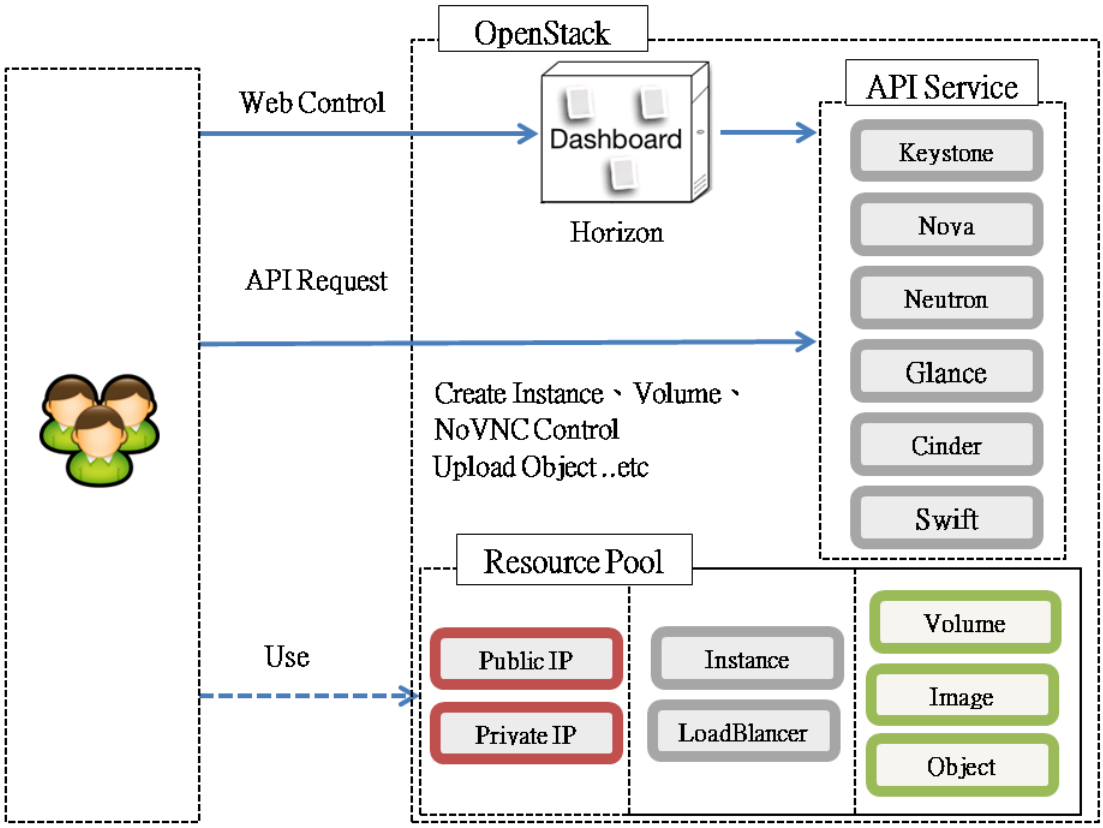

圖 2.1: OpenStack 架構

在 OpenStack 中包含了許多元件，Keystone 是提供 Identity Service 元件，用於提供使用者的身分認證與 Token 管理的功能，以及提供各項服務的 EndPoint，其他服務則可以透過使用者請求時所附加的 Token 進行身分的驗證，

Nova 是一個 Compute Service 元件，主要是包含 OpenStack Compute API 與 Amazon’s EC2 API 的服務，另外還有 Instance 所使用的 Metadata Server 服務以及 NoVNC 的提供，以及在 Hypervisor 上的分配與管理。

Neutron 是一個 Netwrok Service 元件，使用者可自訂 Instance 間的網路環境並且可透過 API 動態聯結浮動 IP 給 Instance 使用，可透過 Neutron Plugin 建立 Router、DHCP、Load Balancer 服務，使用者可透過 API 的呼叫，Neutron 將會於不同 Hypervisor 間與 Plugin Agent 自動建立網路拓譜與結構，目前可採取 VLAN 或 GRE 方式來進行 Instance 間網路封包的傳送。

Glance 是一個 Image Service 元件，可透過 API 的呼叫，上傳 Instance 所使用的 Image，可在 Instance 啟動時提供給 Nova-Compute 所使用，當 Instance 進行 Snapshot 時也會透過此服務進行 Image 的儲存。

Cinder 是一個 Block Storage Service 元件，可透過 API 的呼叫，使 Hypervisor 透過 iSCSI、RBD... 等方式 [11]，動態提供額外的 Volume 給予 Instance 所使用。

Swift 是一個 Object Storage Service 元件，可透過 S3 或 Swift API 的呼叫，在 Bucket/Container 上進行 Key/Vaule 型式的資料儲存，也可當作一個雲端檔案儲存空間使用，目前 Rackspace 擁有 Swift 系統在商業服務上使用。

Horizon 是一個 End User UI 元件，以 Django 為基礎的 Project，可以讓使用者透過瀏覽器取得

一個 Dashboard 介面，進行管理與使用 OpenStack 上的資源。

2.2 Apache Hadoop

Hadoop 是一個由 Apache 所開發的 OpenSource 專案，是一個分散式運算平台，可以透過架設 Hadoop Cluster 進行儲存與計算 petabyte 等級的資料，主要是提供分散式儲存以及分散式運算的架構(圖 2.2)，利用 NameNode 與 DataNode 來提供 HDFS 儲存服務，利用 Jobtracker 與 TaskTracker 提供 MapReducer 計算服務，因此可以分散儲存與計算的負載，以提供計算海量的資料服務。

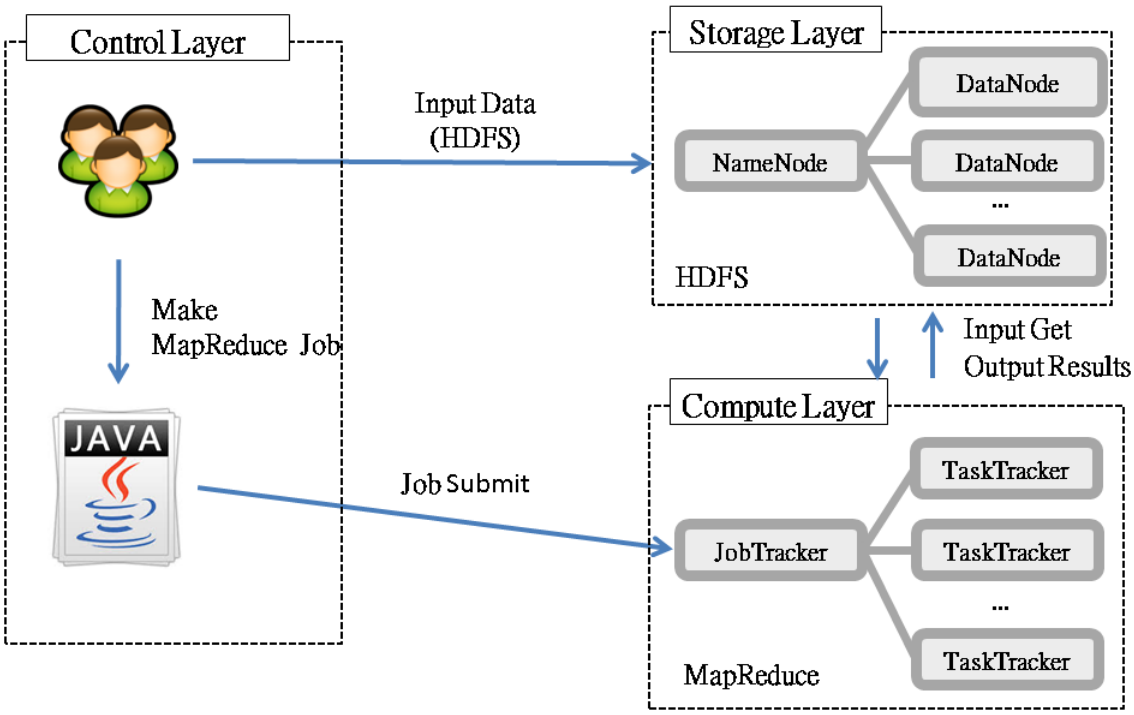

圖 2.2: Hadoop 架構

2.2.1 HDFS(Hadoop Distributed File System)

HDFS 是一個分散式儲存檔案系統 [16]，其運作方式是透過 NameNode 與 DataNode 服務來達到檔案的存取 (圖 2.3)[12]，NameNode 在 HDFS 中是負責管理整個檔案系統結構的程序，因此儲存整個檔案系統的目錄結構，以及保存 Block 索引，因此它不儲存真正的檔案資料，另外 NameNode 在 Hadoop 的 2.x 之前版本是一個 Single Point of Failure 的服務，因此當它發生問題時 HDFS 將無法正常運作，後來在 Hadoop 的 2.x 之後版本將有提供 High Availability 的 NameNode 服務進行設計 [14]，可透過兩個 NameNode 來維持 HDFS 的正常運作，DataNode 在 HDFS 中負責提供 Client 存取 Block 的程序，因此在多個 DataNode 的情況下，可以達到分散 Disk IO 的功能。

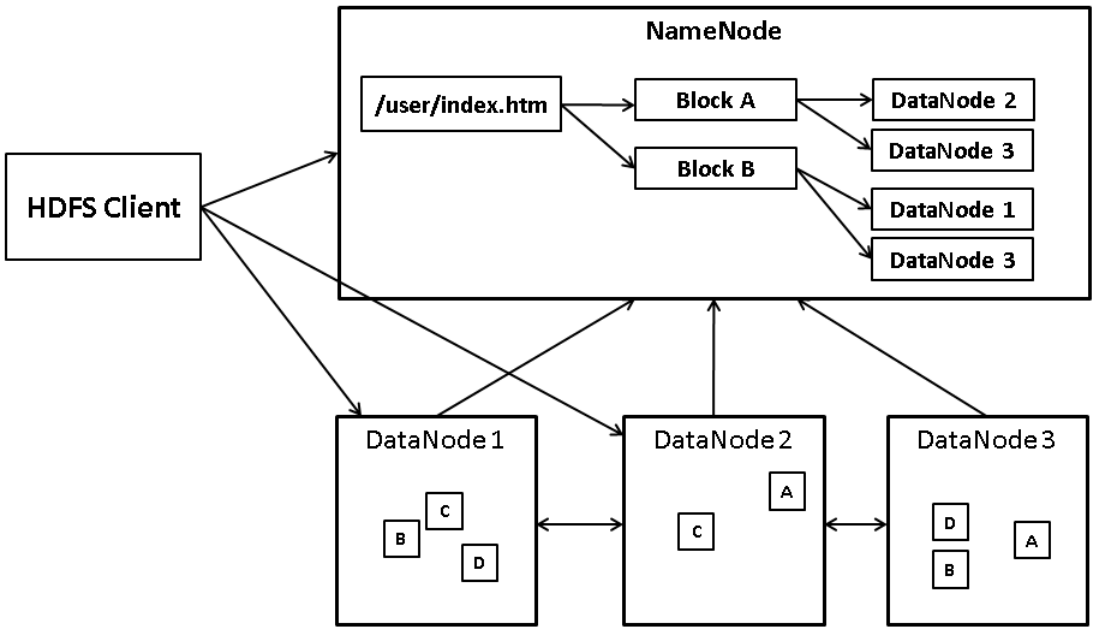

圖 2.3: HDFS 架構

2.2.2 Hadoop MapReduce

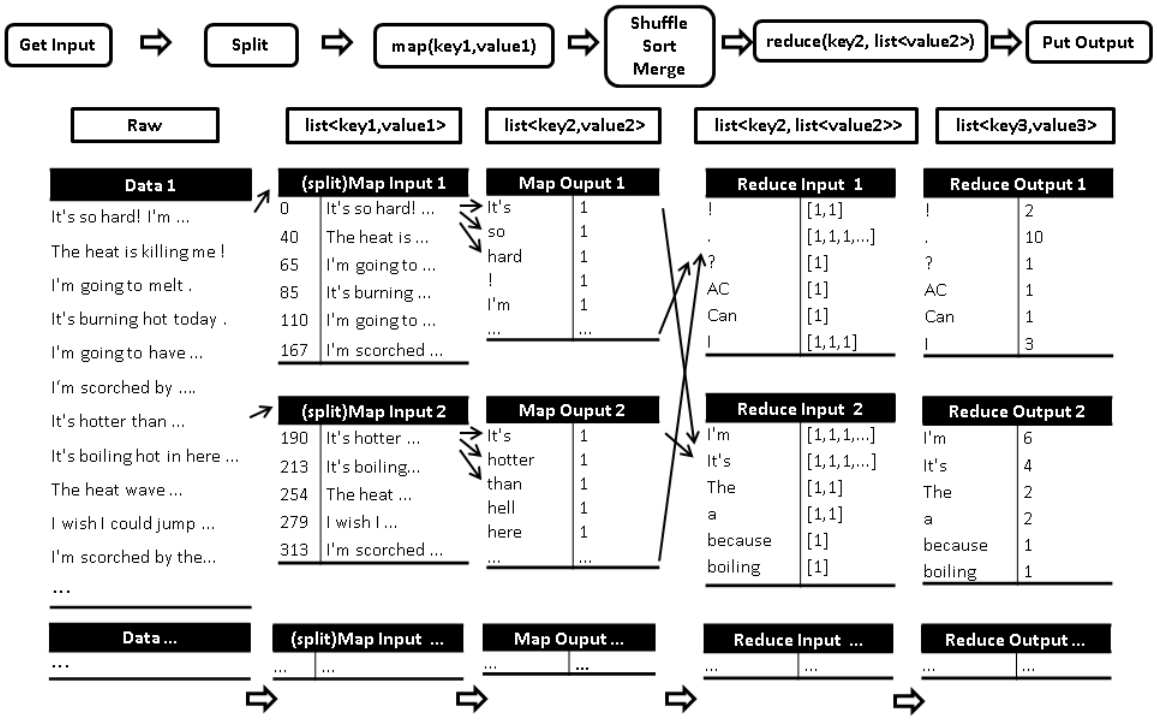

圖 2.4: MapReduce 流程 WordCount 範例

Hadoop MapReduce 是一種利用 key algorithm 所設計出來的程式架構（圖 2.4），主要目的是為了將海量的資料（大於 1TB 以上），能夠讓程式人員容易的撰寫分散式計算程式與執行，其概念是設計 Map 與 Reduce 兩函數來進行所欲進行的計算，Map 是一個 key1,value1 與 list<key2,value2> 的函數轉換，而 Reduce 則是一個 key2,list(value2) 與 list<key3,value3> 的函數轉換，其中 Hadoop

MapReduce 自動進行 Split、Shuffle、Sort、Merge 的處理，使得程式人員可以省去撰寫程式間資料切割與傳遞的部分。

使用者在執行 Hadoop 的 MapReduce 的程式時，主要利用 Hadoop MapReduce Job 程式進行（圖 2.5），在 Job 中可以定義 Mapper 與 Reducer 實作的方式，再透過 JobTracker 與 TaskTracker 進行程式的執行，JobTracker 是一個 Single Point of Failure 的服務，因此當 JobTracker 發生錯誤時，Job 則會無法正常執行，而 Cloudera 有對於 High Availability 的 JobTracker 服務進行設計 [15]，而 TaskTracker 在執行 Task 發生過多錯誤時，JobTracker 則會自動將 Task 重新轉交由其他 TaskTracker 執行，因此在執行海量資料 Job 時可防止有錯誤節點出現造成 Job 執行失敗，TaskTracker 負責從 JobTracker 取得 Task 執行並且定時回報狀態，Task 可分為 Map、Reduce 兩種類型，每個 TaskTracker 運作時會分別有固定 Task Capacity 數量，這是每個 TaskTracker 所可執行 Task 的數量上限。

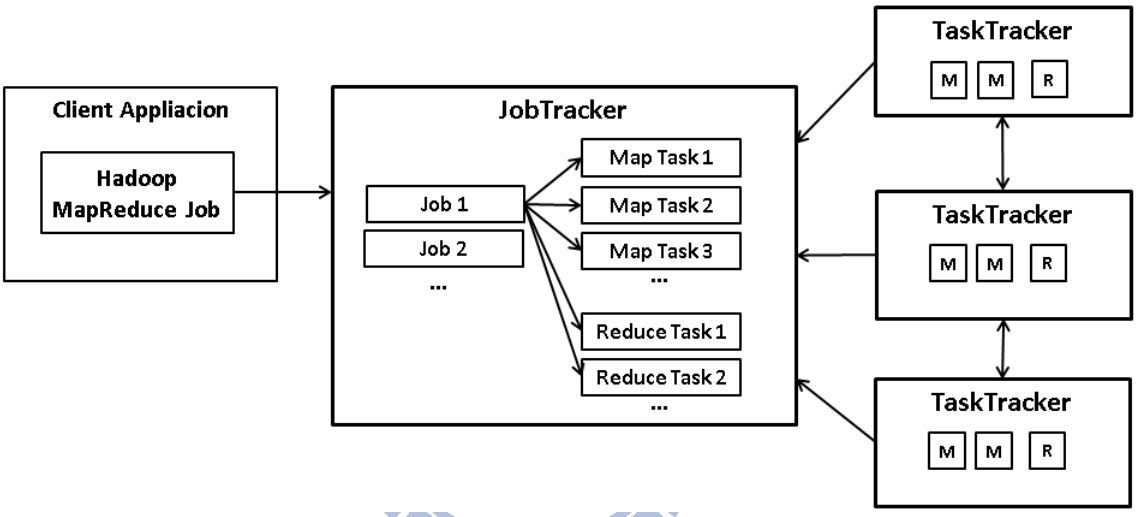

圖 2.5: Hadoop MapReduce 架構

Client 在交付 Job 前可以利用一個 Configuration 指定一些共用的參數（如:Input 與 Output 的 Path.. 等等），交付給 JobTracker 開始時，會先請求一個 JobID，Client 會利用 JobID 取得一特定的 SubmitJobDir(Shared Storage Path) 做為上傳 Job 使用，之後再利用 Configuration 中所指定的 InputFormat 實作類別（如:FileInputFormat）使用 getSplits 函數來產生 MapTask 所使用 InputSplit，此處通常會給定一個資料來源 Path，此 Path 可以利用不同格式來定義所使用的資料來源（如:hdfs://、s3://），透過此函數後將可計算出一個 List 的 SplitMetaInfo(如:FileSplit 類別包含一個檔案的 Path 與 Offset Length 與資料來源的主機名稱）再上傳到 SubmitJobDir 上，此時則可以取得 Map Task 的數量並設定到 Configuration 並上傳 Job 的副本，完成交付的動作，此時 JobTracker 就會開始進行 initTasks 的動作，從 SubmitJobDir 取得所有的 TaskSplitMetaInfo(若是使用 HDFS，此處每個 TaskSplitMetaInfo 都可以取得 Block 所存放的 DataNode 主機名稱)，進行分配每個 Map Task 給 TaskTracker 去執行（使用 HDFS 時會優先使用 Block 存放位置與 TaskTracker 的主機相同的機器）。

TaskTracker 在執行 MapTask 時會使用 InputSplit 的資訊讀取資料的來源（圖 2.6），再透過 InputFormat 實作類別轉換成可以給予 Map 實作類別的 Record，每一筆 Record 都是 Key/Value 的結構，因此使用者可以定義 Mapper 實作類別來進行 Map 的動作，若 Job 有指定 Reduce 時，Map

的輸出將會透過 MapOutputBuffer Spill 儲存到 Local disk(可透過 Configuration 定義 mapreduce.cluster.local.dir 指定 path，預設路徑是在 ${hadoop.tmp.dir}/mapred/local)，另外可以透過指定 mapred.output.compression.codec 參數（如 org.apache.hadoop.io.compress.GzipCodec)，來對輸出進行壓縮，將可減少本地的儲存空間與時間，再傳送給 ReduceTask 時也可減少傳送時間，Spill 時會透過 Partitioner 利用每筆 Record 的 Key 分配該由哪個 ReduceTask 處理，之後 ReduceTask 時會再透過 MapOutputServlet 取得相對應的 Record 進行 ReduceTask。

ReduceTask 在執行時會先透過 MapOutputCopier 從 MapOutputServlet 取得資料後，進行 Shuffle 將資料儲存在 Local Disk，將資料 Copy 完後，再從 Local Disk 讀取成 key/value 結構的 Segment 資料，進行 Sort 與 Merge 的動作後，再透過 RawKeyValueIterator 進行 ReduceContext 的建構，因此使用者可進行 Reducer 的實作進行 Reduce 動作，之後再透過 OutputFormat 的實作類別，將 Record 輸出到 Shared Storage 上。

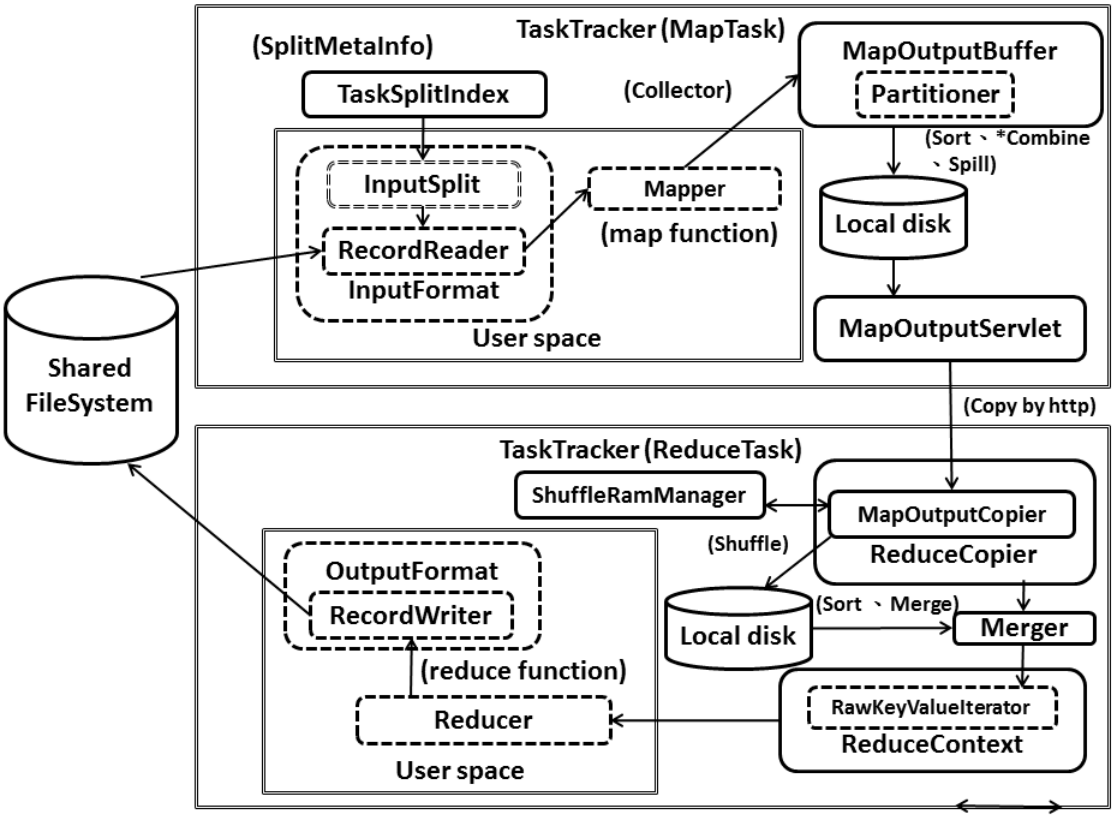

圖 2.6: Hadoop MapReduce TaskTracker 運作流程

2.3 Object Stroage RADOSGW(Ceph)

Ceph 是 Sage A. Weil 等人在 2006 年發表的一具有 Scalable,High Availability、High Performance 的分散式儲存系統 [17]，可提供一個 RADOS(reliable autonomic distributed object store) 的服務，RADOS 是一個 object-based 的儲存服務，可透過 Ceph 所提供的函式庫 (librados) 撰寫 Client 端的應用程式，將會透過 Ceph Storage Cluster Protocal 進行資料的存取與程式間的溝通。

Ceph 在使用上可以分為 Ceph Storage Cluster(RADOS) 與 Ceph Client 兩部分（圖 2.7)，Cluster

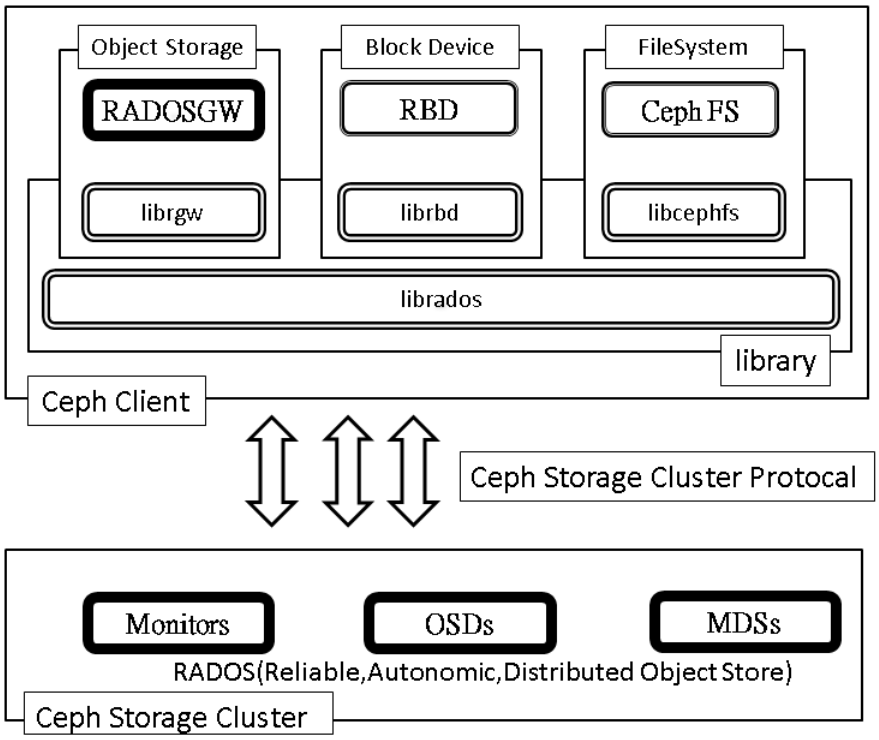

圖 2.7: Ceph 架構

中包含為 Monitors、OSD(Object Storage Daemon)、MDS(MetaData Server) 三種類型的服務，Cluster 在架設時可以透過大量的 OSD 與少量的 Monitor 去維持 RADOS 的服務，OSD 是扮演一個 Logical Disk 的程序，而 Client 部分可以透過 RADOSGW、RBD、CephFS 三種方式來使用 RADOS 的資源，CephFS 可以提供一個 POSIX(Portable Operating System Interface) 的檔案系統，其中 CephFS 在使用上還需要透過 MDS 來提供檔案的 MetaData，而 RBD(RADOS Block Device) 則可以提供 Resizable、Thin-Provisioned 的 Block 裝置，其裝置支援了 Snapshotting 與 Cloning 等操作，RBD 可透過 rbd kernel module 進行裝置的取得，或是透過 librbd 動態提供給 Qemu/KVM 的 Guest 使用，目前已整合 libvirt 可提供 hypervisors 使用，而 RADOSGW(RADOS Gateway) 則可以提供一個支援 S3 或 Swift API 的 Object Storage 服務。

在傳統架構中，Client 對一個 Cluster 服務溝通時，都會透過一個集中式的溝通方式的作為進入點 (如:gateway、broker, API.. 等)，這會使得服務效能以及可延展性受到限制，而且單一節點發生問題時可能容易導致整個服務也受到影響。因此 Ceph 為了避免集中式設計，讓 Client 本地計算該如何對 OSD 進行存取，OSD 接收資料後再與其他節點進行創建副本以確保 Data Safety 與 High Availabilty 的服務，當有 OSD 發生問題時，Monitor 會對 OSD 的存取方式進行調整，以確保 OSD 的 High Availability，Ceph 把這種設計的演算法稱為 CRUSH(Controlled Replication Under Scalable Hashing)[19]。

Client 可以透過 Ceph Stroage Cluster Protocal 進行 Object 的存取，存取 Object 時是利用”Pool” 作為一個邏輯分區的概念，Client 可指定 Pool 來存放 Object，每個 Pool 會有各別定義的副本數 (Replicas Size)、PG(Placement Group) 數量與 CRUSH 規則，儲存在 Pool 中的 Object

會通過 hash 函數（ 預設是 Robert J. Jenkins. Hash[21]）與 mask 被分配到 PG 上，在 CRUSH 規則可以定義階層式的 OSD 結構（此結構在此處被稱為 Bucket）以及挑選的規則，PG 的儲存則是透過 Pool 的副本數與 CRUSH 規則被計算出所要存放的 OSD。

Client 可以透過 Monitor 取得 Cluster Map，Cluster Map 包含 Monitor 、OSD 、MDS、CRUSH、PG Map 資訊，在 Monitor、OSD、MDS Map 中主要記錄所有的服務狀態與服務 IP 與 Port，OSD Map 則還包含了 Pool 設定，Crush Map 中則是記錄 OSD 的 Buckets 結構以及挑選規則，PG Map 則是記錄所有 PG 的狀態，Client 因此可透過 Cluster Map 本地計算出所要存取 Object 的 PG 與 OSD，而直接對 OSD 做溝通，因此要確保 Cluster Map 在 Cluster 中具有一致的資訊就相當重要，而 Monitor Cluster 是透過 Paxos part-time parliament 演算法 [20] 來確保 Cluster Map 在 Monitor Cluster 中具有 Strong Consistency 性質，因此當有 OSD 毀損時或是新加入時，可在幾秒內於透過 Monitors 進行 Cluster Map 的更新，來提供 High Availabilty 服務的。

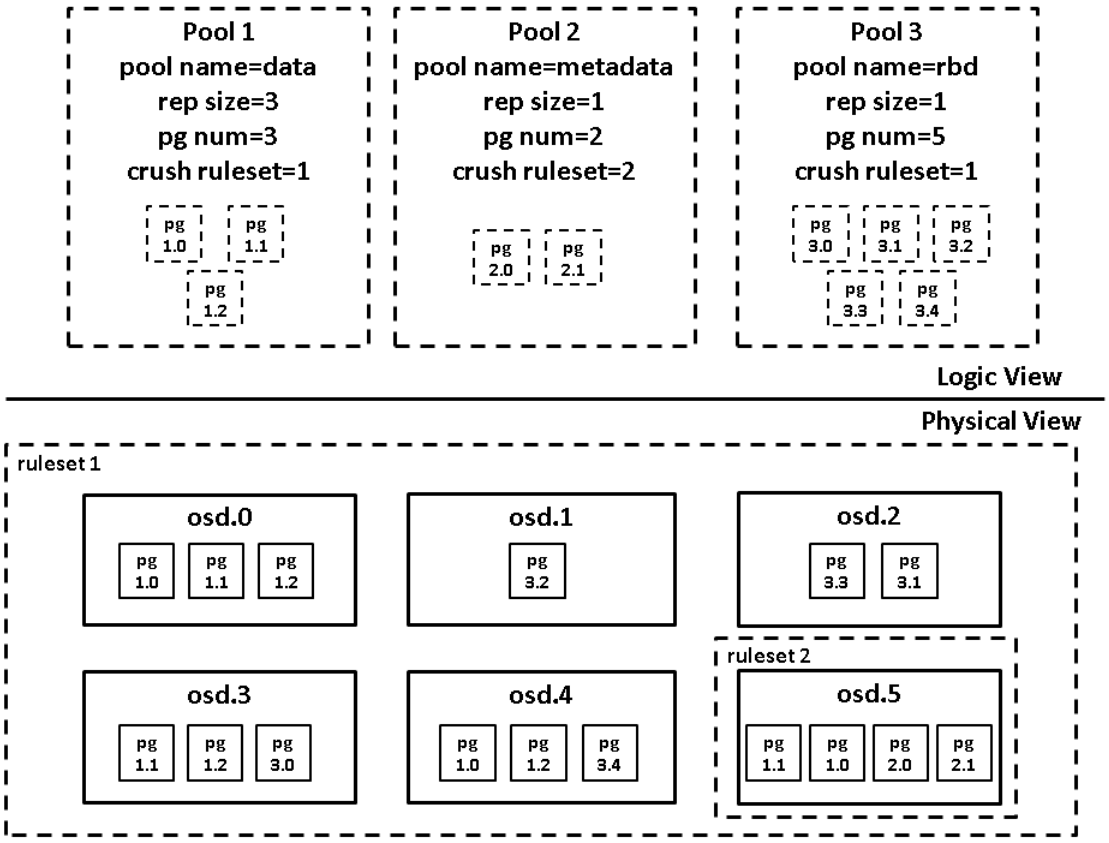

圖 2.8: Ceph Pool 概念圖

RADOSGW 是一個 Ceph 所提供的 FastCGI module，可透過 Apache HTTP Server 提供 Amazon S3 API 與 OpenStack Swift API 來進行 Object Storage 的存取，RADOSGW 中會使用”.rgw”、”.rgw.buckets”.. 等 Pool ，用來儲存 RADOSGW 所使用的帳號權限、bucket、object、log，因此在架設 RADOSGW 的機器上本身是不會儲存 S3/Swift 服務上的資料，在使用 S3/Swift API 所建立的 Bucket/Container 資訊會被儲存在”.rgw” 的 pool 上（圖 2.9），在.rgw 中的 object 會儲存 bucket 的相關資訊，其中也包含 bucket_id，而透過 S3/Swift 所儲存的 Object 會被儲存至”.rgw.buckets” 的 pool 上，該物件會被切割成多個 RADOS Object 所儲存（預設大小 4MB），在 Pool 中的 object name 是被透過”<bucket_id>_<object_key>” 的格式被命名，而在此 object 的”user.rgw.manifest” 的

屬性上會存放其餘切割的 object name(object name 是透過”<bucket_id>__shadow__<id>_<serial number>” 的格式被儲存)，因此當存取 Object 時，將會透過 CRUSH 所計算出來的 PG 分配來分散在 Pool 中所使用到的 OSD disk IO。

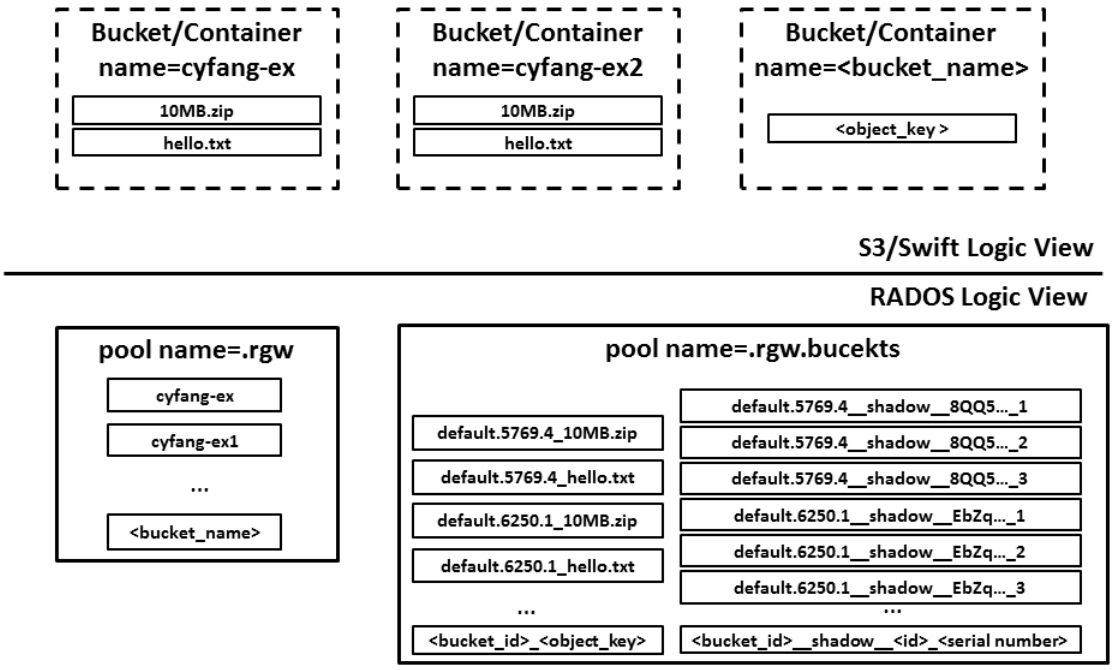

圖 2.9: RADOSGW 儲存結構概念圖

2.4 Amazon Elastic MapReduce (Amazon EMR)

Amazon EMR 是利用一個可利用 Hadoop 框架來進行分散式計算的服務，可以透過服務的使用，輕易的在 Amazon EC2 服務上建立一個調整數量與規格的 Hadoop Cluster 環境 [22]，可透過 Console(a web-based user interface) 、CLI(Command Line Interface)、Web Service API 來進行 Cluster 的管理與使用，其中 EMR 也整合了其他 AWS，包含 Amazon EC2, Amazon S3, Amazon DynamoDB, Amazon RDS... 等，可透過這些服務的使用來進行不同的操作，如: 可以透過 EMR 來分析在 Amazon S3 的資料，將輸出結果儲存到 Amazon RDS 或 Amazon DynamoDB。

Amazon EMR 可以被使用在許多不同的應用上，包含 log 分析、網站索引、資料倉儲、機器學習、金融分析 (financial analysis)、科學模擬 (scientific simulation)、DNA 序列分析 (bioinformatics)，Amazon EMR 上每年都有百萬個 Cluster 被使用者使用。

在使用 EMR 啟動 Instance 時，除了原本 EC2 費用外還需多付 EMR 使用的費用，而在 S3 服務費用上則分為 Data Request、Storage、Transfer 三種標準進行收費，Request 方面每 1000 個 PUT、COPY、POST、LIST 的請求進行 NT 0.15 元 (USD 0.005) 計算，每 1000 個 GET 或其他請求以 NT 0.12 元 (USD 0.004) 計算，DELETE 則不收費，Storage 方面以 GB-Month 為計算單位，在 1024GB-Month

以下時每 GB 以 NT 2.85 元 (USD 0.095) 計算，在 Data Transfer 方面上傳至 S3 的流量不進行收費，因此收集大量資料至 Amazon 只需付 Request 與 Storage 的費用，下載方面若是使用 S3 到 EC2 相同 Region 的情況下不收費，只收取 S3 到 Internet 的費用，前 1G 不收費，10 TB 流量以下時，每 GB 以 NT 3.6 元 (USD 0.12) 計算，Transfer、Request、Storage 的每單位收費價格則會依標準向下調整。

Standard On-Demand 規格								
Instances Type	vCPU	ECU	Memory	Storage	Network	EC2 Price	EMR Price	每小時花費
m1.small	1	1	1.7 GB	160 GB	Low	NT$1.80	NT$0.45	NT$2.25
m1.medium	1	2	2 GB	410 GB	Moderate	NT$3.60	NT$0.90	NT$4.50
m1.large	2	4	3.75 GB	840 GB	Moderate	NT$7.20	NT$1.80	NT$9
m1.xlarge	4	8	4 GB	1680GB	High	NT$14.40	NT$3.60	NT$18

表 2.1: EMR 服務價格表 (以 USD/NTD =1:30)

2.5 Cloud-Init

Cloud-Init 是一個以 Python 實作的 OpenSource 套件 [23]，主要功能是用於啟動 Cloud Instance 時進行客製化的設定，目前 Ubuntu Cloud Images 與 EC2 上 Ubuntu images 都已預設安裝，可以透過多種格式進行撰寫，如下列範例

```
#cloud-config
chpasswd:
  list: |
    root:password
  expire: False
ssh_pwauth: True
```

此範例可以透過啟動 Instance 時指定 -user-data 參數使用，將會使 Instance 啟動後，透過 Metadata Server 取得 User Data 後進行系統的配置，將 root 密碼變更為 password 並且允許 sshd 可透過 password 驗證，以下為可進行的動作:

- 系統時區的設定
- 修改系統主機名稱
- 插入檔案
- 套件庫更新
- 安裝套件
- 執行 Script

除此之外可在 launchpad[24] 上找到其他範例設定。

第 三 章　實作構想

3.1　實作目標

參考 AWS EMR 的 Launch a Streaming Cluster[25] 後，規劃出圖 3.1 的系統目標流程，希望能夠提供使用者下列幾種功能

1. Input Data: 利用 Object Storage 存放所需計算的資料，因此可先將資料蒐集到 Object Storage 上。
2. Launch Cluster: 利用 OpenStack 開啟所需 Hadoop Cluster，依照所需使用 Cluster 規模大小啟動所需的 Instance 規格與數量，並且自動完成佈署建置。
3. Run Hadoop Job: 可透過 Object Storage 取得資料來源進行 Hadoop 程式的計算。
4. Output Data: 最後 Hadoop Job 可將計算結果輸出到 Object Storage 中進行儲存。

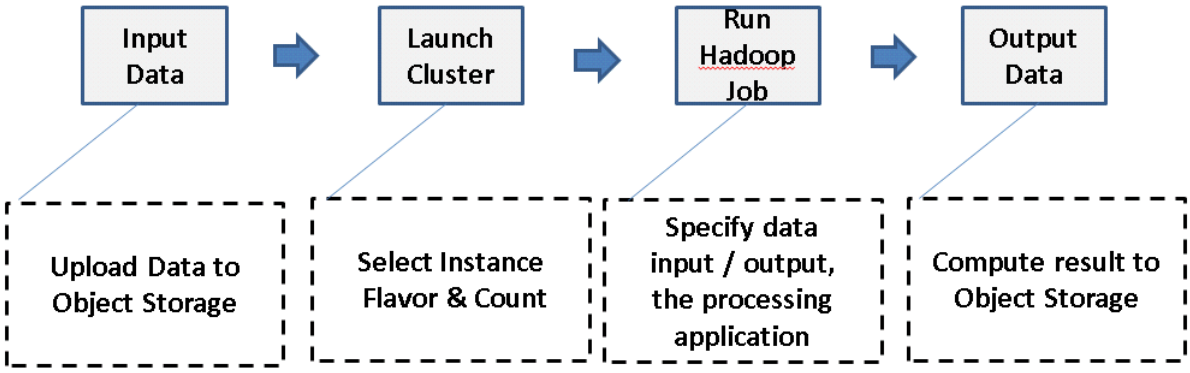

圖 3.1: 系統目標流程

參考 Hadoop 原有設計後希望透過此方式，將運作切割為三個層面 (圖:3.2):

- 控制層: 由於 Horizon 本身定位在一個 Client 工具，除了可由 OpesStack 提供者進行架設外，使用者也可自行架設使用，只需要指定 Keystone 的路徑，則可透過 API 取得 OpenStack 上服務的 endpoint 以進行使用，因此將 Horizon 作為控制層的一個介面。
- 儲存層: 利用 S3 與 Swift API 可以在 Object Storage 進行資料的儲存，因此可讓使用者透過 API 進行資料的收集 (如: 每日 Log 上傳)，之後可再透過 OpenStack 創建出來的 Instance 進行運算動作。

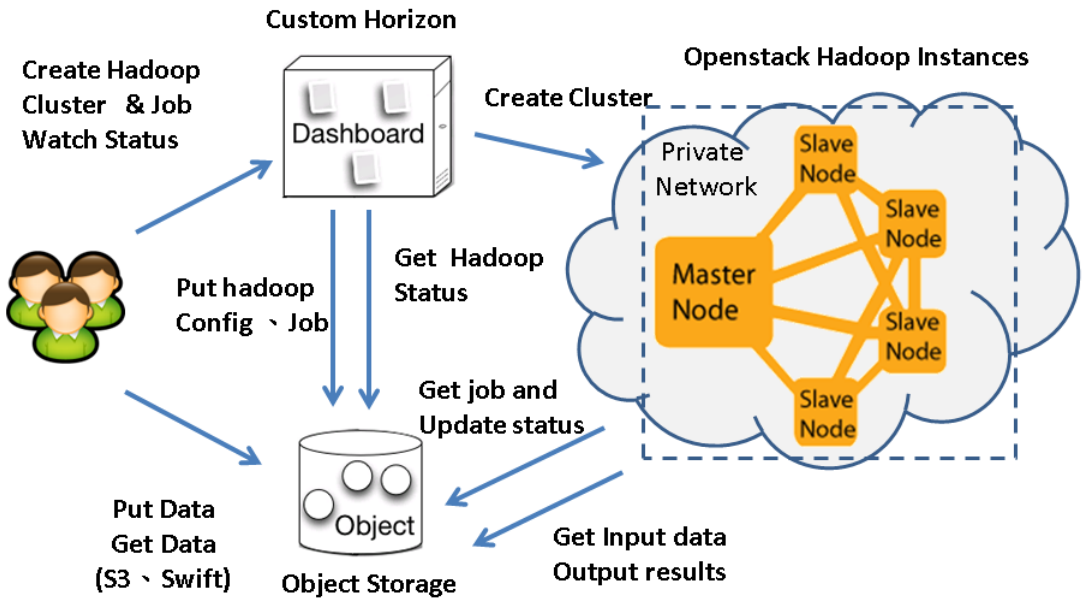

圖 3.2: 實作構想

- 計算層: 透過 Dashboard 進行 Hadoop Cluster 的創建，並且 Cluster 能夠從 Object Storage 取得資料進行 Hadoop Job 的執行

主要目的有下面幾點：

1. 無須額外的部署環境: 一般需要管理多台系統上的設定時，通常需要一台自動部屬 Server(如 Puppet) 來配置 Cluster 的設定，但需額外的架設服務來運行，因此使用者必須熟悉自動部屬工具才能夠進行環境建置，因為希望能夠透過 Horizon 程式，直接對 Cluster 進行部屬配置，並且能夠與 Cluster 溝通，將提供使用者能夠快速建置 Hadoop Cluster。

2. 可客製化的控制介面: 由於使用者所欲執行的 Hadoop Job 有所不同，若以固定的模式建立 Hadoop Cluster，則可能無法達到使用者所需使用的環境，因此為了改善此點，希望可提供使用者修改創建流程，因此將擴充原本 Horizon 程式，並將程式碼 OpenSource，讓使用者可取得 Custom Horizon 後，依程式需求修改使用，將 Dashboard 切割為用戶端，因此使用者也可自行架設 Dashboard 使用 (如: 圖 3.3)，而不會被限制只由服務端提供介面使用。

3. Hadoop Cluster 規模調整: 可讓使用者在 OpenStack 除了能夠建立多個 Hadoop Cluster 外，也希望在建構完 Hadoop Cluster 後，還能夠加入新的 Slave Node，來增加 Task Tracker 數量，擴充 Cluster 規模。

4. 額外的儲存環境: 由於 HDFS 是運行在 Instance 上，因此當將 Instance 移除時所儲存在 HDFS 上資料也會因此遺失，透過此設計可將資料儲存在 Object Stroage 上，因此可在有計算需求時啟動 Instance，計算完後可移除 Instance，避免持續占用 Openstack 的計算資源。

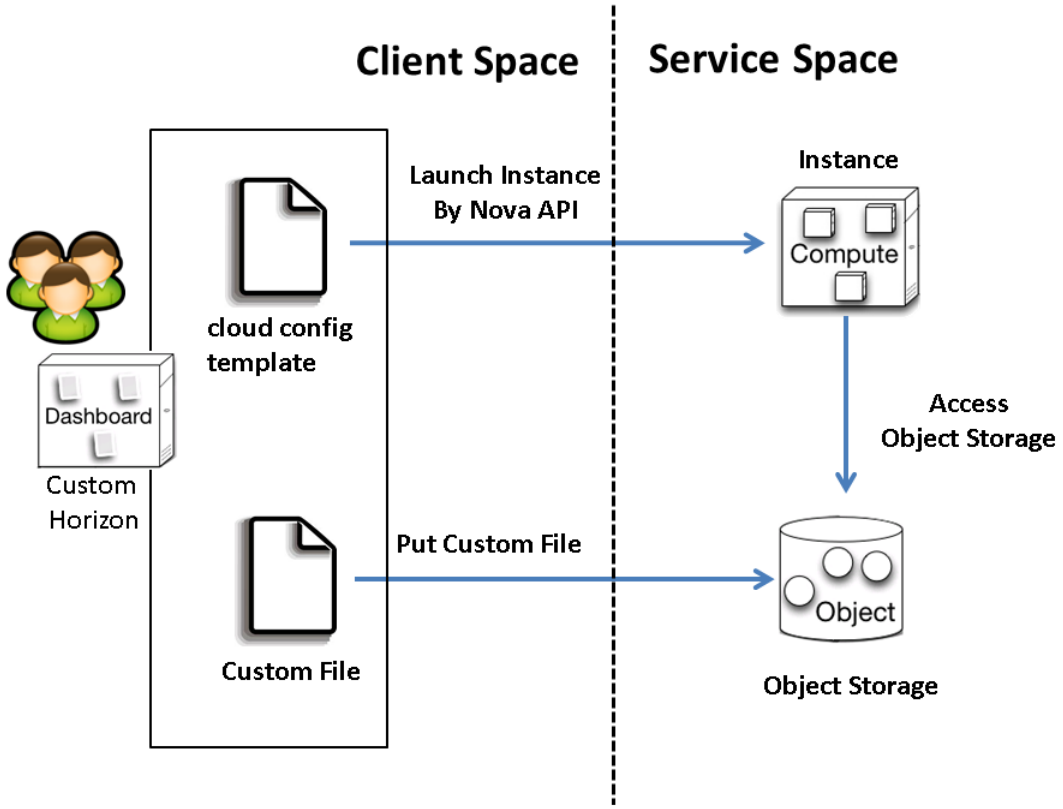

圖 3.3: Custom Dashboard 功能示意圖

3.2 面臨問題

由於 Horizon 的 OpenStack Dashboard 介面是通過 Service 的 API 呼叫設計而成的，因此依照 End User UI 的構想設計時，在實作上會遇到一些問題如下:

1. 客製化介面呈現: 一般動態網站要顯示一個列表時，通常是透過資料庫取得資料，在 Custom Dashboard 呈現 Hadoop Cluster 列表時，若透過服務端使用資料庫存放資料時，使用者在自行架設時將會導致須設定資料庫位置與權限，將會有權限問題存在。

2. Instance 的 Object Storage 存取權限: 為了讓 Hadoop Cluster 能夠從 Object Storage 取得資料運算，無事先在 Image 上設定權限的情況下，必須在 Instance 啟動後具有存取 Object Storage 的權限。

3. Private Instance 控制: 由於 OpenStack 是設計成可提供一個 Private Instance 的 Cloud，因此會無法透過 Horizon 直接對 Private Network 上的 Instance 進行連線，因此在 Private Instance 若無使用 Public float IP 的情況下，將無法直接進行控制，因此必須透過其他方式進行。

4. Cluster Instance List: 由於 Hadoop Cluster 的設定中，所有 Instance 都需要得知 Cluster 中所有主機的 IP 與主機名稱，彼此間透過 hostname 進行連線存取，但 IP 是在 Instance 啟動後所分配的，因無法在啟動前取得所有 Instance IP 列表，來對 Instance 進行設定，因此需要

透過 Private DNS 的服務或是其他方式取得主機清單與 IP，在 Instance 啟動後進行設定上，而目前由於 OpenStack 尚無完整實作 Private DNS 的服務，因此無服務可以正解 Instance 名稱而取得 IP，因此必須採取其他方式解決。

第 四 章　系統實作

4.1　系統架構

為了解決 3.2 節所提到的問題以及可以呈現客製化的介面，因此在設計上本論文透過 Object Storage 來存放資料，作為 NoSQL 的資料庫使用，在使用者登入同時，即可由 Keystone 取得 Token 後取得 Object Storage 的存取權限，而無須再自行架設額外的資料庫進行資料的儲存使用，為了能夠讓 Instance 可以在啟動後可以存取 Object Storage，本論文可以在 Horizon 撰寫 cloud-config 的 template，在啟動 Instance 的時候從 Keystone 取得能夠存取 Object Storage 的 S3 Key，並產生相對應的 cloud-config，使得 Instance 在啟動後進行 cloud-init 的時候可進行安裝，在 Private Instance 控制上，本論文採取 polling 方式，讓 Instance 主動連出取得控制的動作進行，因此必須要有一個能夠讓 Horizon 與 Instance 能夠共通存取以作為溝通的服務存在，由於啟動後本論文可以存取 Object Storage，因此可在特定路徑上定期的檢查，來做為控制的方式，另外在每台 Instance 啟動後，本論文也可在特定的路徑上，產生 Instance 的列表，使得其他 Instance 可以取得 Cluster Instance List，因此將系統架構設計為圖 4.1，來解決以上問題。

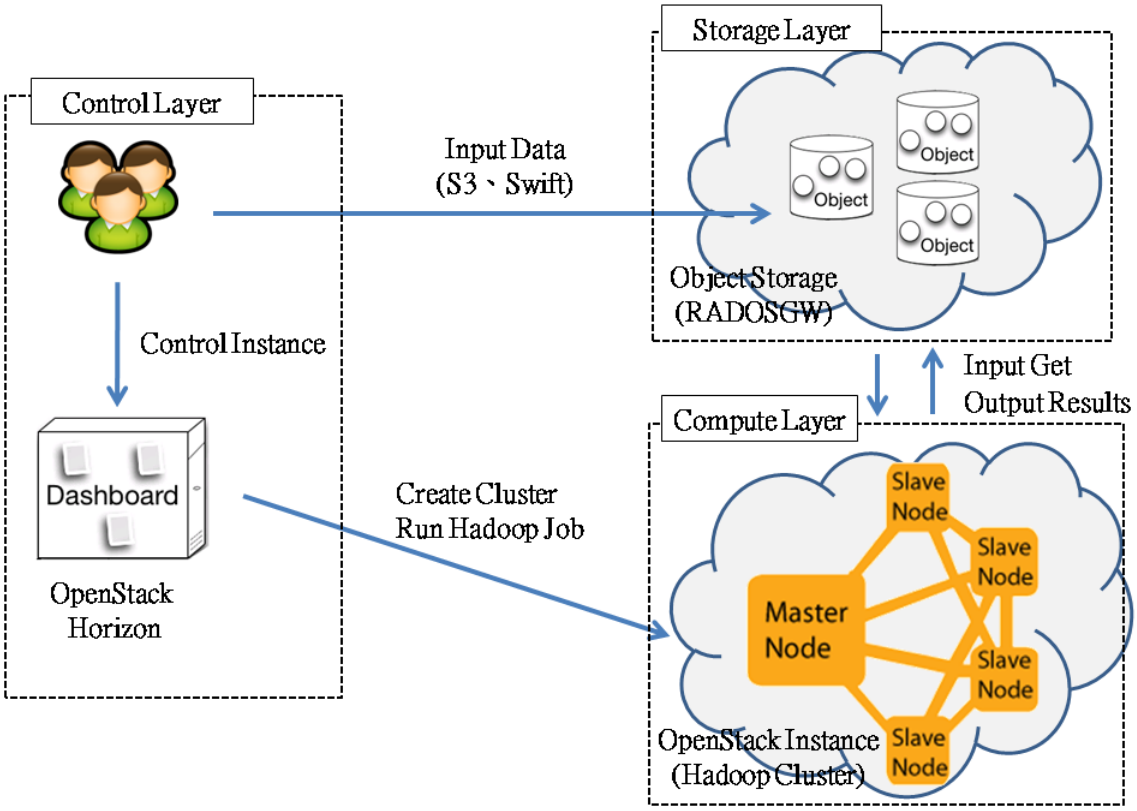

圖 4.1: 系統架構

4.2 Hadoop Cluster 啟動流程

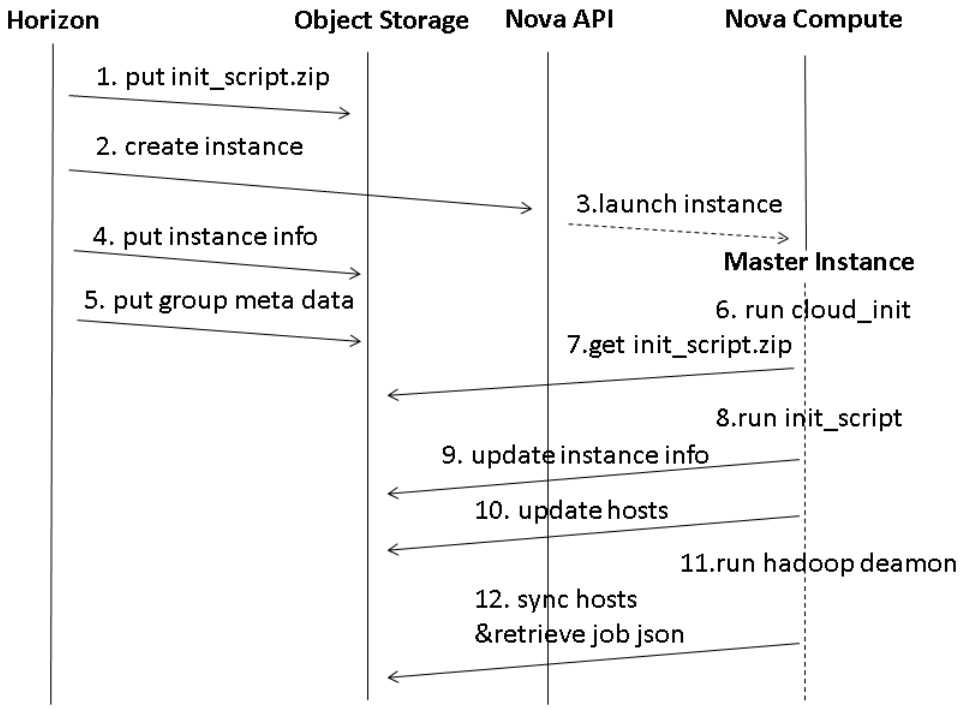

圖 4.2: Create Master Flow

在創建 Cluster 過程中，將分成 Master 與 Slave 兩個部分創建，在 Create Master Instance 流程將分成 12 個步驟進行，下列為步驟說明

1. Horizon 在接受使用者請求創建 Master Instance 後，將產生一 group-id 作為群組識別使用，並透過 Object Storage 在 custom-<project-id>/.hadoop/<group-id>/init_script.zip 檔案路徑上存放一個壓縮檔，此處將會利用 Horizon 所存放的 Custom 檔案進行打包，裡面包含 EC2 key 以及 init_script 以供 Instance 初始化使用。

2. Horizon 將會利用 group-id、ec2 key.. 等參數，產生相對應的 cloud-config，透過 Nova API 啟動 instance，使得步驟 6 可取得 cloud-config 進行初始化動作，同時也取回所建立的 instance-id。

3. Nova API 會建立一個 instance-id 回應 Horizon，再透過 Nova-Scheduler 在其中 Nova-Compute 上啟動 Instance。

4. 利用 instance-id 在 Object Storage 中 custom-<project-id>/.hadoop/<group-id>/instance/<instance-id> 的路徑上，儲存相關的 instance 資訊，以供 Horizon 顯示使用。

5. 在 custom-<project-id>/.hadoop/meta/<group-id> 路徑上，儲存此 group meta data，其中包含 network、cloud-config、keypair、root 密碼、ec2 key、Master ID 資訊，以供啟動 Slave 使用。

6. Instance 啟動後會進行 cloud-init 的程序，將透過 http://169.254.169.254/2009-04-04/user-data 網址，從 metadata server 上取得 Horizon 所產生的 cloud-config 檔案進行初始化，此時會進行 Boto、euca tools 的安裝與環境變數上的設定，並且進行修改 Hadoop 程式碼，同時也取得 Object Storage 的存取權限。

7. 從 Object Storage 取得 init_script.zip 解壓縮並開始執行從 Horizon 上所定義的 init_script。

8. 產生 Hadoop Configuration 的預設設定，會將 core-site.xml、hdfs-site.xml、mapred-site.xml 進行修改。

9. 更新 Instance 資訊，將提供給 Horizon 顯示使用。

10. 從 custom-<project-id>/.hadoop/<group-id>/instance/<instance-id> 中取得所有 instance 資訊並且轉成 hosts，上傳至 custom-<project-id>/.hadoop/<group-id>/hosts 固定路徑，以供同步 hosts 檔使用

11. 開始運行 Hadoop deamon，對 HDFS(NameNode 結構) 進行初始化動作，並且開始運行 JobTracker 與 NameNode。

12. 定期檢查 Object Storage 上的 hosts 大小是否與 Instance 上相同，若不一致將下載 hosts 進行更新，使得更新後 Hadoop deamon 可以透過 hostname 查詢到 IP，並且從 custom-<project-id>/.hadoop/job/<group-id>/queue/路徑中檢查，以 polling 方式檢查是否有 job 存在，使用者執行 Job 時，可透過 Horizon 介面的操作後，將會對此路徑進行存放 job 資訊，因此 instance 可取得資訊後進行 job 的執行。

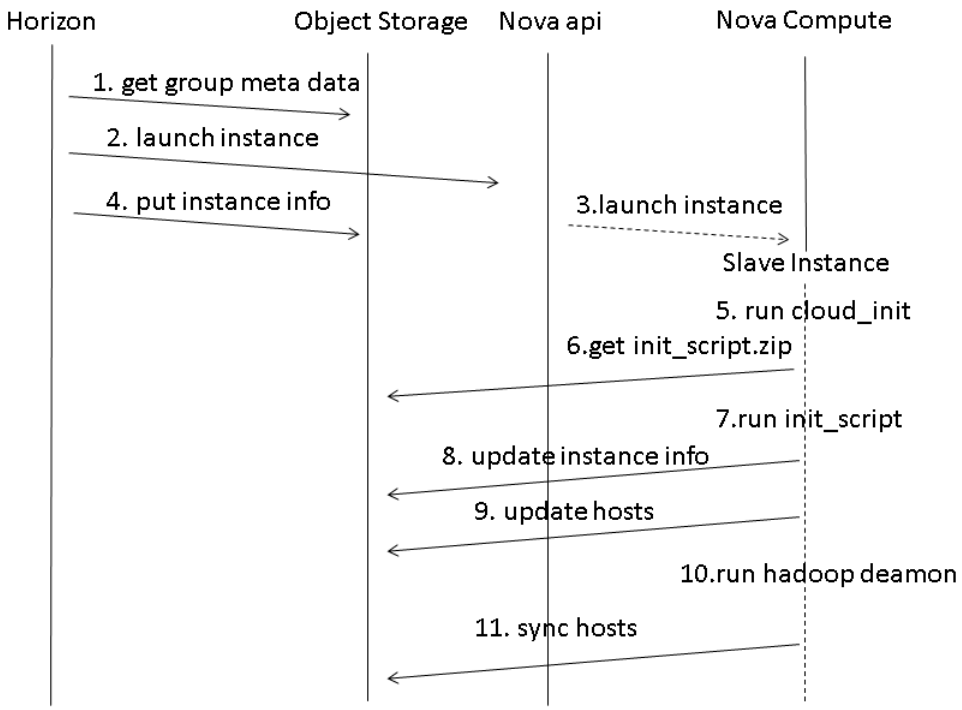

圖 4.3: Create Slave Flow

而在創建 Slave 過程中主要可以分成 11 個步驟，下列為步驟說明

1. Horizon 利用 group-id 從 Object Storage 取得 group meta data，已得知所需設定的參數。
2. 利用 group meta data 產生的 cloud-config，再透過 Nova API 啟動 Instance，使得步驟 5 可取得 cloud-config 進行初始化動作，同時也取回所建立的 instance-id。
3. Nova API 會建立一個 instance-id 回應給 Horizon，再透過 Nova-Scheduler 在其中 Nova-Compute 上啟動 Instance。
4. 利用 instance-id 在 Object Storage 中 custom-<project-id>/.hadoop/<group-id>/instance/<instance-id> 的路徑上，儲存相關的 instance 資訊，以供 Horizon 顯示使用。
5. Instance 啟動後會進行 cloud-init 的程序，將透過 http://169.254.169.254/2009-04-04/user-data 網址，從 metadata server 上取得 Horizon 所產生的 cloud-config 檔案進行初始化，此時會進行 Boto、euca tools 的安裝與環境變數上的設定，並且進行修改 Hadoop 程式碼，同時也取得 Object Storage 的存取權限。
6. 從 Object Storage 取得 init_script.zip 解壓縮並開始執行從 Horizon 上所定義的 init_script。
7. 產生 Hadoop Configuration 的預設設定，將 core-site.xml、hdfs-site.xml、mapred-site.xml 進行修改。
8. 更新 Instance 資訊，將提供給 Horizon 顯示使用。

9. 從 custom-<project-id>/.hadoop/<group-id>/instance/<instance-id> 中取的所有 instance 資訊並且轉成 hosts，上傳至 custom-<project-id>/.hadoop/<group-id>/hosts 固定路徑，以供同步 hosts 檔使用

10. 開始運行 Hadoop deamon，開始運行 DataNode 與 TaskTracker。

11. 定期檢查 Object Storage 上的 hosts 大小是否與 Instance 上相同，若不一致將下載 hosts 進行更新，使得更新後 Hadoop deamon 可以透過 hostname 查詢到 IP。

4.3 影像檔製作

由於 Hadoop Cluster Instance 是透過 cloud-init 來進行客製化動作，因此本論文採取 Ubuntu Cloud Image 作為 Instance 系統的使用，由於本身並無 Java 與 Hadoop 程式碼，為了省去啟動時安裝 Java 與 Hadoop 的時間，因此可以事先對 Image 進行安裝後上傳使用，而客製化 Ubuntu Cloud QCOW2 Image 的製作方法，可利用一台 Ubuntu Instance 上進行下列動作製作:

```
#使用root進行操作
$ sudo -i

#安裝所需套件
$ apt-get install qemu-utils

#下載 http://cloud-images.ubuntu.com/releases/ 上取得所需修改版本的image
$ wget http://cloud-images.ubuntu.com/releases/12.04.3/release/ubuntu-12.04-server-cloudimg-amd64-disk1.img

#載入Network block Device kernel module 使得可透過qemu-nbd來進行image file連接
$ modprobe nbd

#連接image檔到nbd裝置上
$ qemu-nbd -c /dev/nbd0 `readlink -f ./ubuntu-12.04-server-cloudimg-amd64-disk1.img`

#mount nbd裝置
$ mount /dev/nbd0p1 /mnt
$ mount -o bind /dev /mnt/dev
$ mount -t proc none /mnt/proc
$ mount -o bind /sys /mnt/sys
$ mount -o bind /tmp /mnt/tmp

#切換root到image上
chroot /mnt /bin/bash

#設定name server
$ mv /etc/resolv.conf /etc/resolv.conf.bak
$ echo " nameserver␣8.8.8.8" > /etc/resolv.conf

#開始安裝java與hadoop套件
$ add-apt-repository ppa:webupd8team/java
$ add-apt-repository ppa:hadoop-ubuntu/stable
```

```
$ apt-get update apt-get install oracle-java7-installer oracle-java7-set-default -y
$ apt-get install hadoop -y

#清除cache
$ rm -r /var/cache/oracle-jdk6-installer
$ rm -r /var/cache/apt/archives/*.deb

#還原nameserver 設定
$ rm /etc/resolv.conf
$ mv /etc/resolv.conf.bak /etc/resolv.conf

#返回原本root exit
$ umount image
$ umount /mnt/ qemu-nbd -d /dev/nbd0
```

經過此方式則可完成 Image 的操作，即可上傳到 OpenStack 使用。

4.4 Keystone 與 Object Storage(RADOSGW) 整合

目前若要使用 Swift API 來存取 RADOSGW，需透過設定來啟動與 keystone swift token 認證功能，只需要在/etc/ceph/ceph.conf 加入相對應的設定即可，此方式可在 Swift API 檢驗身分時，透過 Keystone 檢驗 Token 的權限並且 cache 在 RADOSGW 上。

```
[client.radosgw.gateway]
       rgw keystone url = {keystone server url:keystone server admin port}
       rgw keystone admin token = {keystone admin token}
       rgw keystone accepted roles = {accepted user roles}
       rgw keystone token cache size = {number of tokens to cache}
       rgw keystone revocation interval = {number of seconds before checking revoked tickets}
```

但 S3 部分，由於 Ceph 是在 v0.72 才新增的功能，因此若是在此之前的版本可以透過修改 Keystone 來建立 S3 key 的方式來達，在 Keystone 建立 credential 的同時利用 radosgw-admin 指令建立相對應 key，因此 RADOSGW 在做身分驗證時，即可取得相對應的使用者身分與 Bucket 權限，下列為 Keystone API 所需修改的 keystone/contrib/ec2/core.py 程式碼部分:

```
--- /usr/share/pyshared/keystone/contrib/ec2/core.py.orig 2013-07-18 13:03:39.603678655 +0800
+++ /usr/share/pyshared/keystone/contrib/ec2/core.py 2013-11-20 17:48:03.536391638 +0800
@@ -50,6 +50,35 @@

CONF = config.CONF

+import subprocess,json,os
+def _rgwadmin(cmd):
+ rgwadmin_cmd=' /usr/bin/sudo /usr/bin/radosgw-admin %s'  % (cmd)
+ try:
+ stdRed=subprocess.check_output(rgwadmin_cmd,shell=True,  env=os.environ)
+ return stdRed
+ except:
```

```
+ pass
+
+def rgw_key_create(cred_ref):
+ cred_ref[' project_id' ]=cred_ref[' tenant_id' ]
+ sub_cmd=" subuser create --subuser={project_id}:{user_id} --access=full" .format(**cred_ref)
+ _rgwadmin(sub_cmd)
+ sub_cmd=" key create --subuser={project_id}:{user_id} --key-type=s3 --access-key={blob[access]} --secret
    ={blob[secret]}" .format(**cred_ref)
+ _rgwadmin(sub_cmd)
+
+def rgw_key_delete(cred_ref):
+ cred_ref[' project_id' ]=cred_ref[' tenant_id' ]
+ sub_cmd=" user info --uid={project_id}" .format(**cred_ref)
+ info=json.loads(_rgwadmin(sub_cmd))
+ keys=info[" keys" ]
+ secret_key=cred_ref[" blob" ][" secret" ]
+ subuser=" {project_id}:{user_id}" .format(**cred_ref)
+ delete_keys = filter(lambda k:k[" user" ] == subuser
+ and k[" secret_key" ] == secret_key
+ ,keys)
+ for key in delete_keys:
+ sub_cmd=" key rm --subuser={user} --key-type=s3 --access-key={access_key} " .format(**key)
+ _rgwadmin(sub_cmd)

@dependency.provider(' ec2_api' )
class Manager(manager.Manager):
@@ -215,6 +244,8 @@
                  ' access' : uuid.uuid4().hex,
                  ' secret' : uuid.uuid4().hex}
        self.ec2_api.create_credential(context, cred_ref[' access' ], cred_ref)
+ cred_ref[' blob' ]={' access' : cred_ref[' access' ],' secret' :cred_ref[' secret' ]}
+ rgw_key_create(cred_ref) #rgw s3 key create
        return {' credential' : cred_ref}

    def get_credentials(self, context, user_id):
@@ -260,7 +291,9 @@
            self._assert_owner(context, user_id, credential_id)

        self._assert_valid_user_id(context, user_id)
- self._get_credentials(context, credential_id)
+ cred_ref=self._get_credentials(context, credential_id)
+ cred_ref[' blob' ]={' access' : cred_ref[' access' ],' secret' :cred_ref[' secret' ]}
+ rgw_key_delete(cred_ref) #rgw s3 key delete
        return self.ec2_api.delete_credential(context, credential_id)

    def _get_credentials(self, context, credential_id):
```

4.5 Hadoop 與 Object Storage 整合

為了能夠讓 Hadoop 的程式，可以透過特定的 Object Storage 服務存取，因此必須對 Hadoop Cluster 上進行一些設定，使得 Hadoop 進行 MapReduce 程式時，可存取 Object Storage，由於 Hadoop 程式中對檔案存取時，都是統一使用 org.apache.hadoop.fs.FileSystem 此抽象層的類別的來開啟檔案，而真正實作的 Class 是利用 Path 所給予的路徑型式，來切換要用何種 Class 處理，在表 4.1 與 4.2 中有不同形式介紹。

	Local FileSystem	HDFS
Path 範例	file:///user/	hdfs://<NameNode Host>:<port>/user/
Java Class	org.apache.hadoop.fs.LocalFileSystem	org.apache.hadoop.hdfs.DistributedFileSystem
說明	存取檔案是利用本地的檔案系統目錄，可在 JobTracker、TaskTracker 與 Client 固定路徑上掛載 Shared Storage(如 NFS) 代替 HDFS 使用，但此方式在越多台機器執行 MapReduce 時將無法達到分散 Disk IO。	Hadoop 程式所提供的檔案系統 (HDFS)，需指定 NameNodo 路徑，可透過 hdfs-site.xml 指定預設使用的 NameNode 路徑，存取檔案是透過 NameNode 取得 Block 索引，直接與 DataNode 存取 Block，因此可以達到分散 Disk IO。

表 4.1: Local 與 HDFS 比較

	S3 Native FileSystem	S3 Block FileSystem
Path 範例	s3://<Access Key>:<Secret Key>@<Bucket Name>/user/	s3bfs://<Access Key>:<Secret Key>@<Bucket Name>/user/
Java Class	org.apache.hadoop.fs.s3native.NativeS3FileSystem	org.apache.hadoop.fs.s3.S3FileSystem
說明	存取檔案是直接透過 S3 API 進行，此方式的優點是可以透過其他 S3 工具或函式庫來對檔案操作，但是缺點是無法儲存大於 5G 以上的檔案，因此無法完全代替 HDFS 使用。	利用 S3 API 以 Block 的方式存取的檔案系統，類似 HDFS 結構，每個檔案會被切割成 Block 的形式儲存在 Object Storage 上，這種方式在 rename 檔案時會較有效率 (由於 S3 搬移 Object 的方式是以 Copy 與 Delete 的 API 來完成)，此方式應使用一獨立的 Bucket 儲存，可支援超過 5G 的檔案存取，但是使用其他 S3 工具會無法直接讀取原始檔案。

表 4.2: S3 Native 與 S3 Block 比較

要使得 Hadoop 的 MapReduce 程序可利用 S3 Block FileSystem 或 S3 Native FileSystem 進行運算時需要 Access Key 與 Secret Key，可以定義預設參數在 core-site.xml，將可使 Hadoop 在路徑使用時省去輸入 Access Key 與 Secret Key 的輸入，另外由於使用 S3 FileSystem 寫入時，是先將檔案寫入本地目錄上後再上傳至 Object Storage，因此也可透過 fs.s3.buffer.dir 與 fs.local.block.size 的參數，也可修改暫存檔案的路徑與 Block 的大小，如下列設定:

```
<?xml version=" 1.0" ?>
<?xml-stylesheet type=" text/xsl"  href=" configuration.xsl" ?>
<configuration>
  ...
  <property><name>fs.s3.impl</name><value>org.apache.hadoop.fs.s3native.NativeS3FileSystem</value></property>
  <property><name>fs.s3.awsAccessKeyId</name><value>ID</value></property>
  <property><name>fs.s3.awsSecretAccessKey</name><value>SECRET</value></property>

  <property><name>fs.s3bfs.impl</name><value>org.apache.hadoop.fs.s3.S3FileSystem</value></property>
  <property><name>fs.s3bfs.awsAccessKeyId</name><value>ID</value></property>
  <property><name>fs.s3bfs.awsSecretAccessKey</name><value>SECRET</value></property>
```

```
  <property><name>hadoop.tmp.dir</name><value>/tmp/hadoop-${user.name}</value></property>
  <property><name>fs.s3.buffer.dir</name><value>${hadoop.tmp.dir}/s3</value></property>

  <property><name>fs.local.block.size</name><value>67108864</value></property>
</configuration>
```

另外由於設定上無法透過 Hadoop Configure 來修改使用 S3 服務的 endpoint，而在 Hadoop 中在對 S3 Native FileSystem 與 S3 Block FileSystem 時存取時是利用 jets3t 函式庫來進行操作，而有關的設定是在 s3service.s3-endpoint 參數上（預設是 s3.amazonaws.com），因此需要修改此參數，必須在 Hadoop 主要的程式碼 hadoop-core.jar 當中加入 jets3t.properties 檔案來進行設定。

```
cd /tmp
#產生jets3t.properties設定檔
tee jets3t.properties <<EOF
s3service.s3-endpoint=s3.nctu.edu.tw
s3service.https-only=false
s3service.s3-endpoint-http-port=80
s3service.s3-endpoint-https-port=443
s3service.disable-dns-buckets=true
EOF
#將設定植入hadoop-core.jar上進行修改
zip -r /usr/lib/hadoop/hadoop-core-*.jar . -i jets3t.properties
```

第 五 章　系統成果與測試

5.1 Hadoop Dashboard 介面

在實作過程中，本論文將原本的 Horizon 進行修改，並且透過 GitHub 進行 OpenSource[26]，由於 Horizon 框架的關係，本論文可以自由的新增模組，在 Horizon 中是以 Dashboard 作為模組載入，因此可在 local_settings.py 設定 INSTALLED_APPS 進行新增模組的動作，因此本論文保留了原本 Openstack 的 Dashboard 外，擴增了一個 Custom DashBoard，以提供管理 Hadoop Cluster 功能。

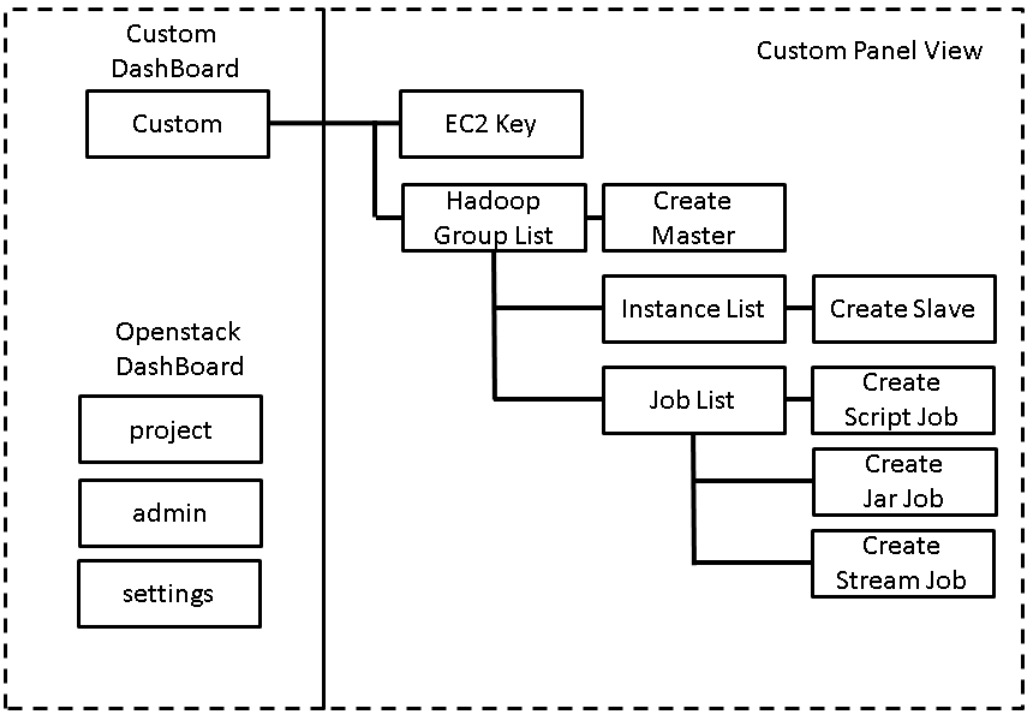

圖 5.1: Custom Horizon 架構

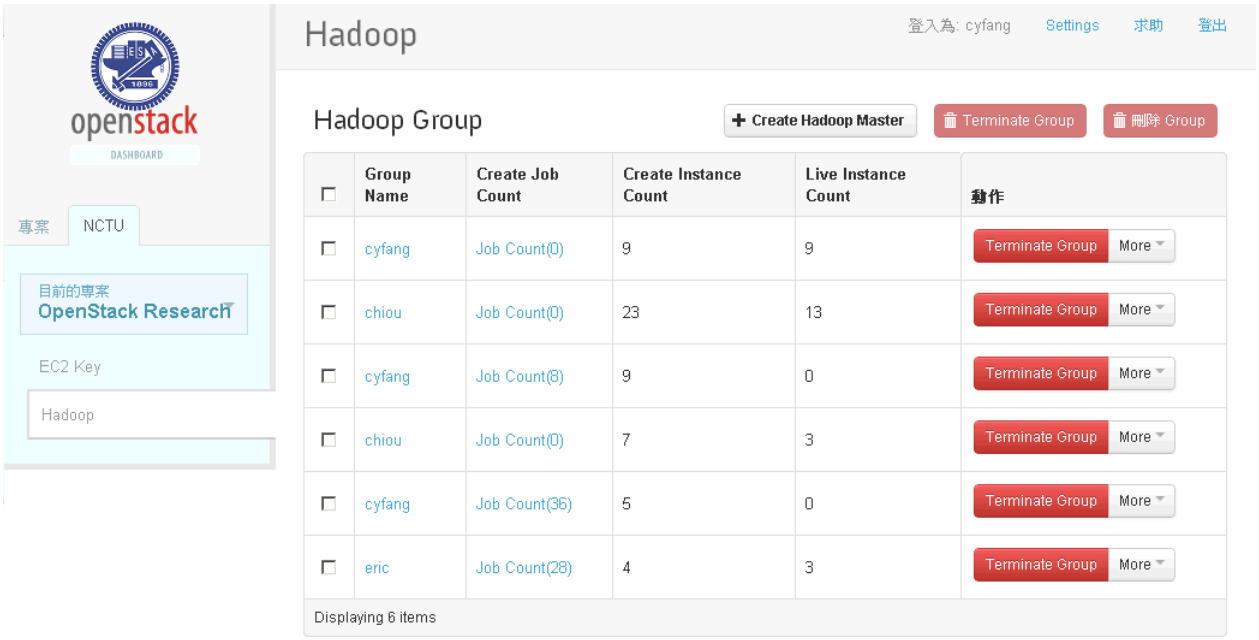

圖 5.2: Group List 頁面

頁面說明

- 此頁面可以瀏覽目前在此專案上，所有創建的 Hadoop Group，並且提供一個進入點，可以對 Group 進行管理

按鍵說明

- Create Hadoop Master: 建立一個 Group 的 Master Instance ，將會在其 JobTracer 與 NameNode 並執行 Job
- Terminate Group: 終止所有選擇 Group 中的 Instance
- Delete Group: 刪除並終止所有選擇的 Group 資料包含 Instance 與 Job

欄位說明

- Group Name: 建立 Master 所命名的名稱，點擊 Link 將會進到其 Group 的 Instance List 頁面
- Create Job Count: 此 Group 所執行過的 Job 數量，點擊 Link 後將會進到其 Group 的 Job List 頁面
- Create Instance Count: 此 Group 所建立的 Instance 數量
- Live Instance Count: 此 Group 目前尚未終止的 Instance 數量

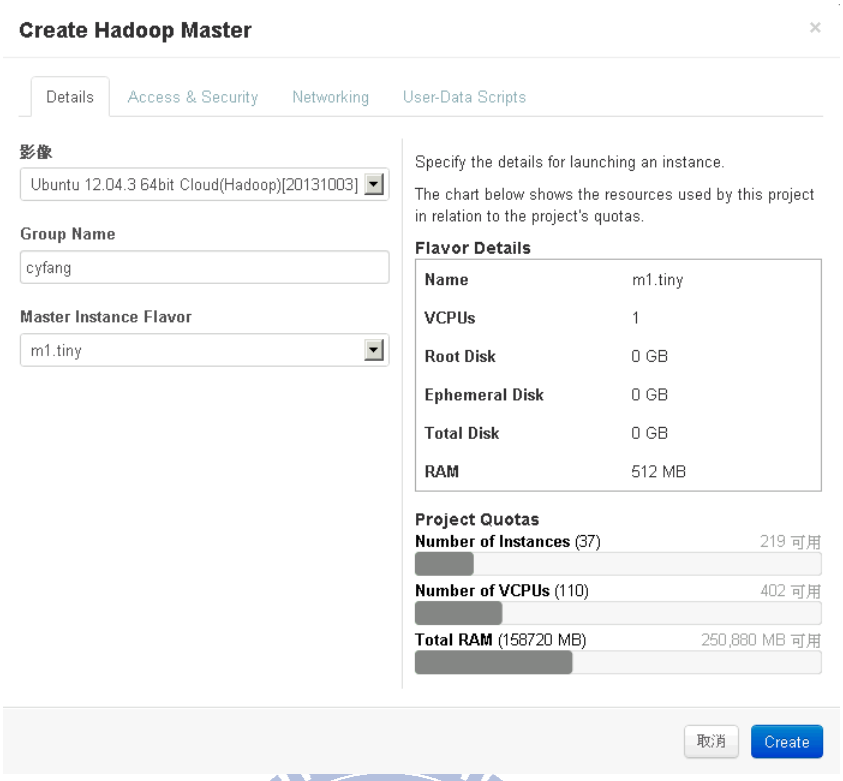

圖 5.3: Create Hadoop Master 頁面

頁面說明

- 此頁面用來創建與設定 Master Instance，創建後會創建 Group，可進入 Instance List 頁面創建 Slave

欄位說明

- Detail 設定
  - Group Name: 群組名稱，之後建立 Instance 將會以 <GroupName>-master 與 <GroupName>-<InstanceID> 此格式命名
  - Master Instance Flavor:Master 所建立的 Instance 規格
- Access&Security
  - EC2 Key:Group 中所有 Instance 所使用的 EC2 Key Keypair:Master 所使用的 Keypair (option)
  - Root Password: Master Instance 所使用 Root password，之後將可用 ssh 登入時可使用
  - Security Groups：Master Instance 所使用的 Security Group Rules
- Networking
  - Selected Networks: 選擇 Instance 所使用的 Network
- User-Data Scripts
  - User-Data Scripts: 設定客製化腳本，將會在 Instance 啟動後執行，可另外安裝所需套件使用

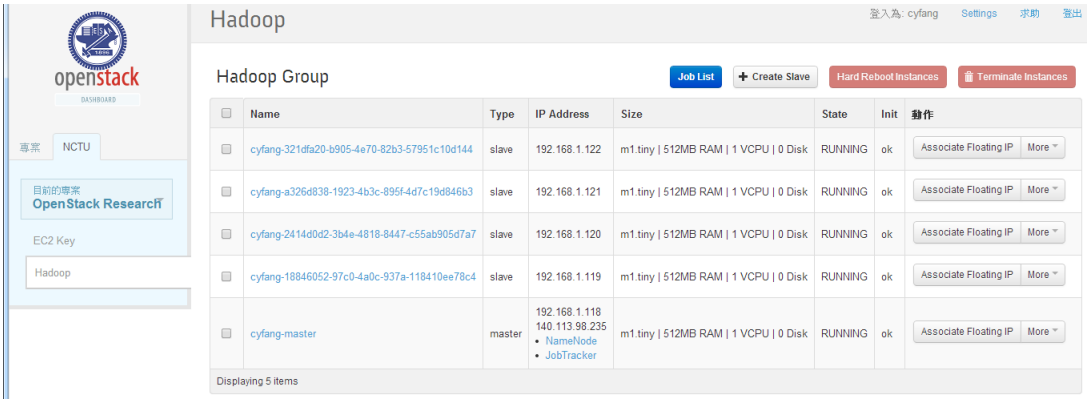

圖 5.4: Instance List 頁面

頁面說明

- 此頁面用來顯示與提供目前 Group 中所擁有的 Instance 狀態，也可對 Instance 進行操作與管理

按鍵說明

- Job List: 切換到 Job List 頁面
- Create Slave: 建立此 Group 的 Slave Instance ，將會在其 TaskTracer 與 DataNode
- Hard Reboot Instance: 重啟所選擇的 Instance
- Terminate Instances: 終止所選擇的 Instance
- Associate Floating IP: 可分配一浮動 IP，提供給 Instance 所使用

欄位說明

- Name: Instance 名稱
- IP Address: Instance 位址
- Size: Instance 規格
- State: Instance 運作狀況
- Init: Hadoop Instance 安裝狀況

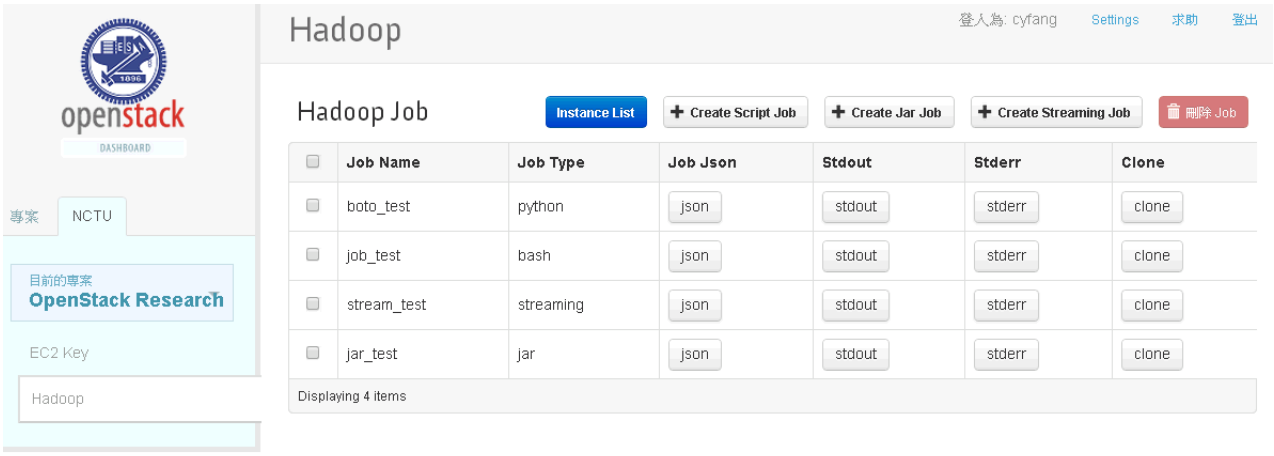

圖 5.5: Job List 頁面

頁面說明

- 顯示此 Group 所創建過的 Job 資訊，並且提供程式輸出結果的資訊

按鍵說明

- Instance List: 切換到 Instance 頁面
- Create Script Job: 創建一 Script Job
- Create Jar Job: 創建一 Hadoop Custom Jar Job
- Create Streaming: 創建一 Hadoop Streaming Job

欄位說明

- Job Name: 創建 Job 時所設定的名稱
- Job Type: 顯示執行 Job 的類型
- Job Json: 顯示創建時輸入的資料，以 Json 呈現
- Stdout: 顯示運行 Job 的 stdout 結果
- Stderr: 顯示運行 Job 的 stderr 結果
- Clone: 創建 Job，預設欄位將會填入同該 Job 資料

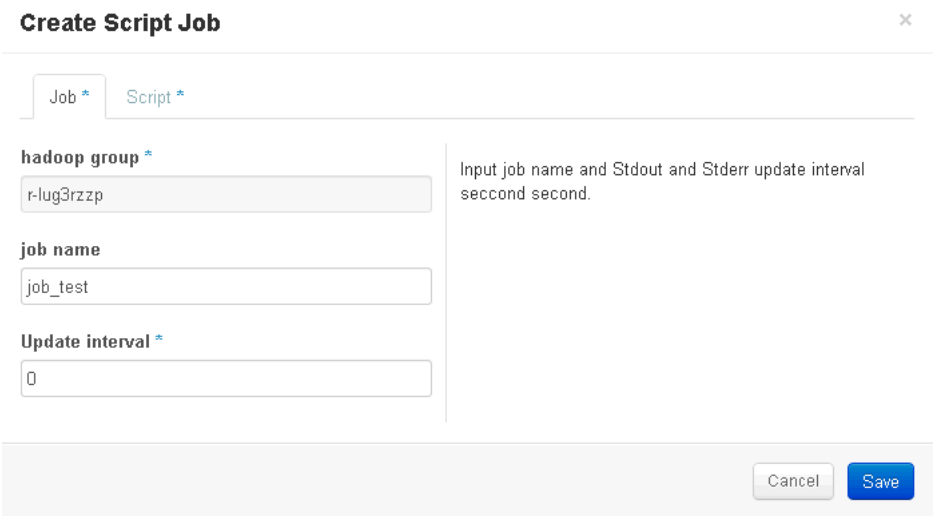

圖 5.6: Create Job 頁面

頁面說明

- 設定在 Job List 所顯示的名稱，以及上傳 stdout 與 stderr 的區隔時間

欄位說明

- Job name: job name 將會在 list 上顯示
- Update interval:job 執行中，上傳 stdout 與 stderr 的時間間隔，設定 0 為執行完後才上傳

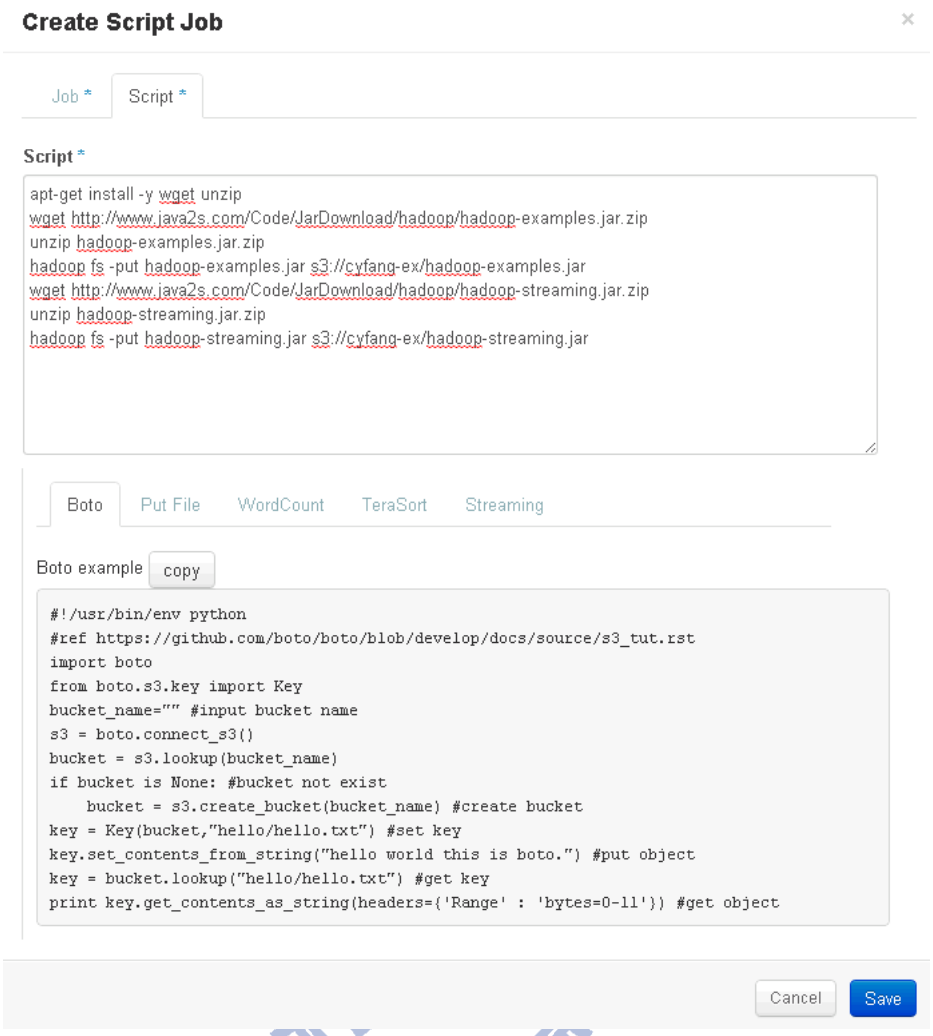

圖 5.7: Create Script Job 頁面

頁面說明

- 可以透過一般的 unix/linux Script 程式執行，如 bash、python、awk.. 等，將會在 Master Instance 上執行，可在第一行定義執行程式，若未給定時預設是以 bash 執行，可以直接在上頭撰寫 Hadoop 運行的 Script 執行，下方為簡單的範例程式可供參考使用

欄位說明

- Script:Master 執行 Job 時所會執行的 script 指令碼

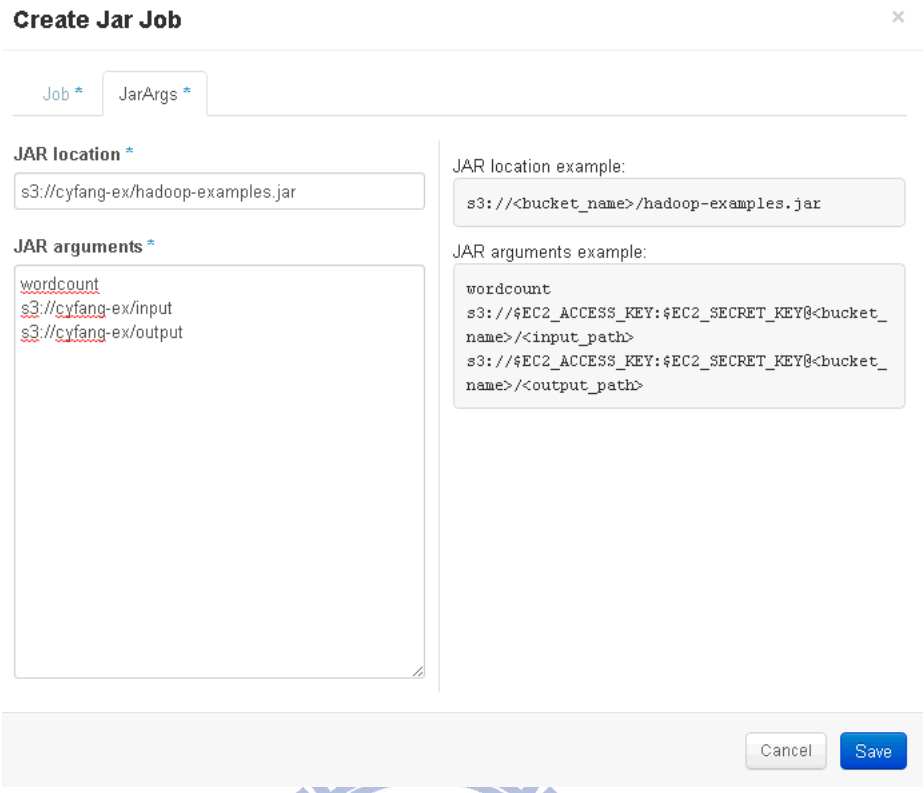

圖 5.8: Create Jar Job

頁面說明

- 可以透過 JAR location 取得程式碼後，進行 hadoop mapreduce 程序。

欄位說明

- JAR location: 指定所要運行的 Hadoop job jar 取得路徑，jar 要存在 bucket 中才可正常運行
- JAR argument: 指定運行時所使用的參數

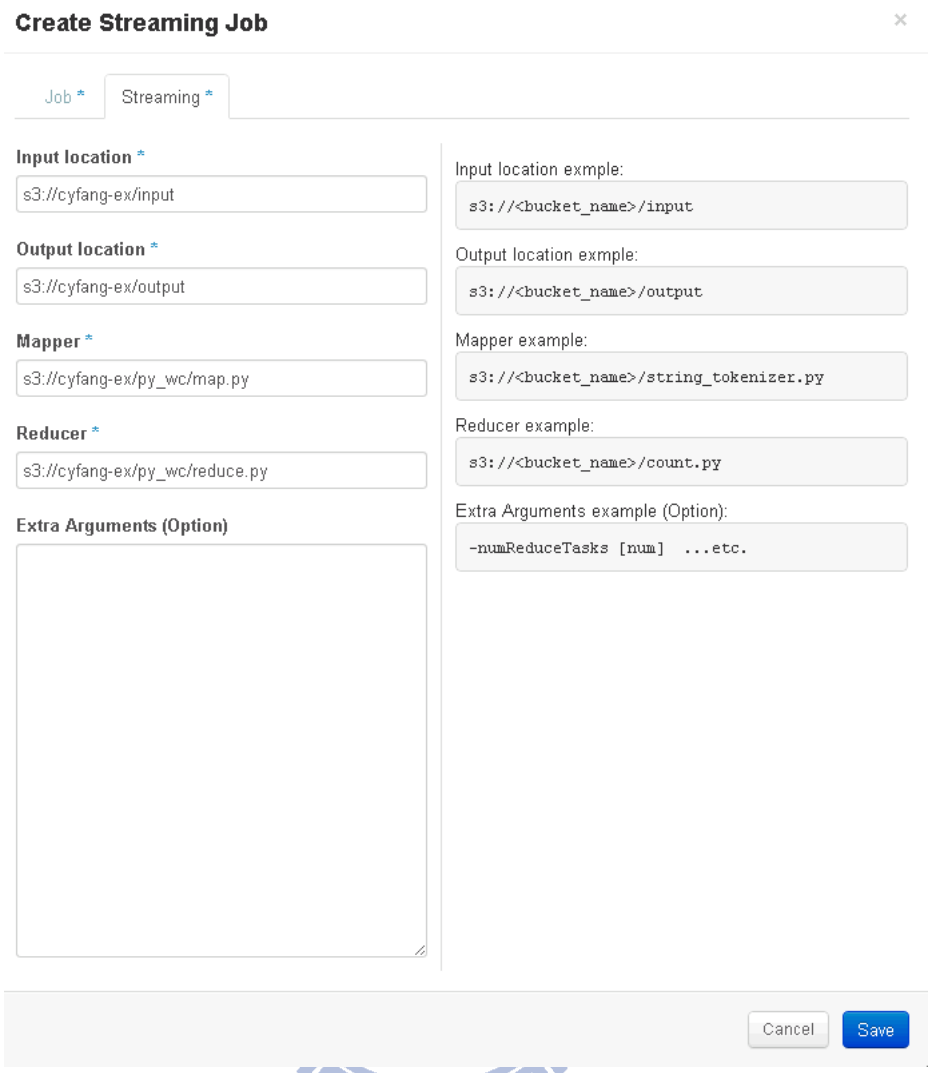

圖 5.9: Create Streaming Job

頁面說明

- 利用 Hadoop 所提供的 streaming 工具所進行的 job，會利用 Mapper 與 Reducer 欄位取得程式後，進行 Hadoop Stream Job

欄位說明

- Input location: 指定 streaming 所要運行的 input 路徑
- Output location: 指定 streaming 所要運行的 output 路徑
- Mapper: 指定 streaming 所要運行的 mapper 程序
- Reducer: 指定 streaming 所要運行的 reducer 程序
- Extra Arguments: 指令 streaming 所使用的額外參數

5.2 Hadoop Instance 啟動時間比較

由於 Hadoop Cluster 使用的規模，是依計算需求來決定需要啟動 Instance 的數量與規格，因此在這裡探討了在 OpenStack 上啟動不同規格的 Instance 情況，虛擬機使用 KVM 與自行安裝 Hadoop 版的 Ubuntu Cloud-Images 12.04.3 來進行比較，在 OpenStack 上透過 flavor 來定義 Root Disk

Size，用於調整 Instance 啟動時系統所存放的 Disk Size，當設定為 0 時，使用 Image 所建立 Disk Size 為基礎，可以觀察到當要啟動越大的 Root Disk 機器時，開機所花費的時間就越久，此處因為 cloud-init 在進行 boot 時會對 Root Disk 進行 extend 的動作導致，將會使得 Instance 在實體設備上所使用的空間逐漸增加，因此大多時間花費於此。

規格名稱	RAM 大小	VCPU 數量	Root Disk 大小 (GB)	啟動時間 (秒)	實際使用空間大小 (MB)
tiny	512MB	1	0(Image Size 2G)	81	23.55
small	2GB	1	20	116	338.81
medium	4GB	2	40	156	687.12
large	8GB	4	80	221	1351.37
xlarge	16GB	8	160	340	2682.06

表 5.1: Instance 啟動所需時間

5.3 Hadoop 執行效能比較

本論文利用學校現有設備進行建置 OpenStack 與 Ceph Cluster，其實體環境如圖 5.10，在 RADOS Cluster 當中，使用了 12 台 OSD 提供給 RADOSGW 服務使用，分別另外架設了 3 台 RADOSGW 作 Object Storage 服務，此外還透過 Varnish 軟體架設 Reverse Proxy 服務來作為 Load Balance，在 OpenStack Cluster 當中本論文利用 4 台相同規格的設備給 Instance 使用，每台設備具有 Xeon E5-2620 2.00GHz(24 Core) 、96G RAM 的主機進行實驗，每台 Instance 規格是以 4 個 vCPU 與 4G RAM 作為使用。

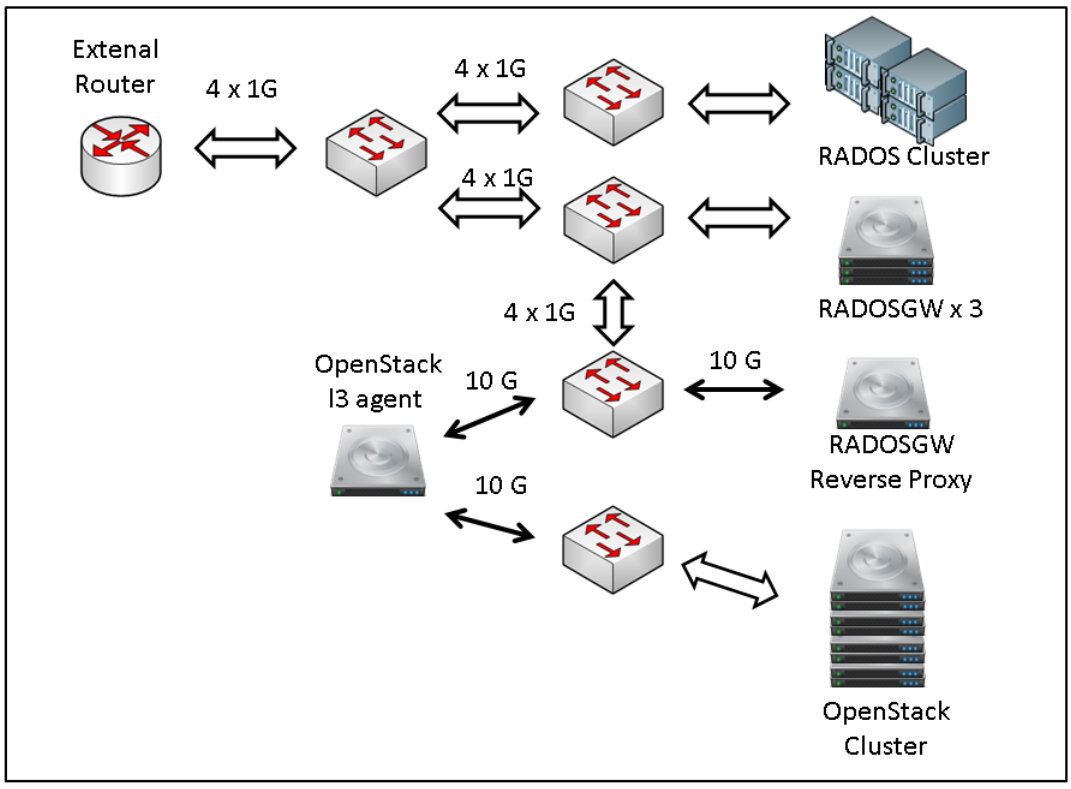

圖 5.10: NCTU OpenStack 與 Ceph 環境

在程式效能比較中，可以利用 Hadoop 中所提供的 Example 程式碼來進行，利用 TeraGen 與 TeraSort 來進行比較，本論文使用 HDFS 與 S3 Block FileSystem 來當作測試資料的儲存空間，透過 Script 的方式來進行 Teragen 與 Terasort 測試的動作 (如: 程式 5.1)，在 Hadoop Example 中，由於都使用 org.apache.hadoop.util.GenericOptionsParser 類別來進行剖析參數，因此本論文可以透過”-D” 參數來設定 Job Configure 的設定，在 TeraGen 當中本論文可以指定所要產生的資料大小，在第一個參數指定要產生的行數，每一行將產生 100 byte 大小的資料，而第二個參數指定輸出路徑，在 Job Configure 當中可以指定 mapred.map.tasks 參數，設定 Map Task 的數量進行產生資料，透過這種方式本論文可以測試在多個 TaskTracker 時，使用 Map Task 輸出到 Share Storage 的效率，而 TeraSort 則可以將對 TeraGen 所產生的資料，透過 TeraInputFormat 與 TeraOutputFormat 進行 MapReduce，此方式將會透過 Hadoop 的框架來完成 Sort 的動作，因此可對 Hadoop Cluster 進行相同資料量的輸入與輸出測試，來檢測 Hadoop 整體運算效率，此外可以指定 hadoop.job.history.user.location 參數來設定 Job History 的輸出路徑，可利用此輸出資料取得運算過程的情況以作為 Log 收集與分析。

程式 5.1 測試程式 Script

```
#!/bin/bash
for i in 1 2 4 8 16 32 64 128 256
do
export bucket=" cyfang-ex"
export jar_location=" s3://${bucket}/hadoop-examples.jar"
export jar=`basename ${jar_location}`
test -e ${jar} || hadoop fs -get ${jar_location} ${jar}

export size_g=" $i"
export size=" $((${size_g}*1024*1024*1024))"
export map_count=$((${size}/$((256*1024*1024))))
export reduce_count=" $((${size}/$((256*1024*1024))))"

export s3_bucket=" cyfang-s3"
export teragen_out=" s3bfs://${s3_bucket}/teragen_${size_g}g"
export terasort_out=" s3bfs://${s3_bucket}/terasort_${size_g}g"

export conf_opts=" -D fs.s3.block.size=268435456 -D dfs.block.size=268435456 -D fs.local.block.size=268435456 -
    D mapred.map.tasks.speculative.execution=false -D mapred.reduce.tasks.speculative.execution=false -D mapred
    .reduce.slowstart.completed.maps=1.0 -D mapred.task.timeout=600000"
export log_location=" -D hadoop.job.history.user.location=s3://${bucket}/logs/gen${size_g}g/b256m${map_count}/"

hadoop jar ${jar} teragen -D mapred.map.tasks=${map_count} ${log_location} ${conf_opts} $((${size}/100)) ${
    teragen_out}

export log_location=" -D hadoop.job.history.user.location=s3://${bucket}/logs/sort${size_g}g/b256m${map_count}r${
    reduce_count}/"
hadoop jar ${jar} terasort -D terasort.partitions.sample=100000 -D mapred.reduce.tasks=${reduce_count} ${
    log_location} ${conf_opts} ${teragen_out} ${terasort_out}
hadoop fs -rmr ${teragen_out}
hadoop fs -rmr ${terasort_out}
done
```

在實驗過程中本論文分別啟動 1 台 Master 與多台 Slave 進行測試，在 Master 中會運行 JobTracker 與 NameNode 程序，每台 Slave 會運行 TaskTracker 與 DataNode，可以分別運行 8 個 Map 與 3 個 Reduce Task，本論文分別針不同 Slave node 數量與不同資料量進行 TeraGen 與 TeraSort 的測試，以 256MB 作為 Block Size，設定成每個 Map 與 Reduce 處理的資料量約為 256MB，利用每個 Job 所花費時間與處理資料量計算出平均處理速率。

在使用 HDFS 當作 Share Storage Storage 時，可以觀察到在處理 16G 資料量以上時，速率將

會達到一個穩定的情況，而在 4 node 與 8 node 的速率曲線相似，其原因是 HDFS 的 DataNoe 是由 Slave instance 所架設的，因此 Slave instance 在實體設備運行需大量存取 Disk 行為時，會受限於實體 Disk IO 的效率，因此當 4 台實體設備只使用 Local Disk 的情況時，Hadoop Job 執行須大量存取 Disk IO 的運算時，4 node 與 8 node 的處理速率會呈現接近的曲線。

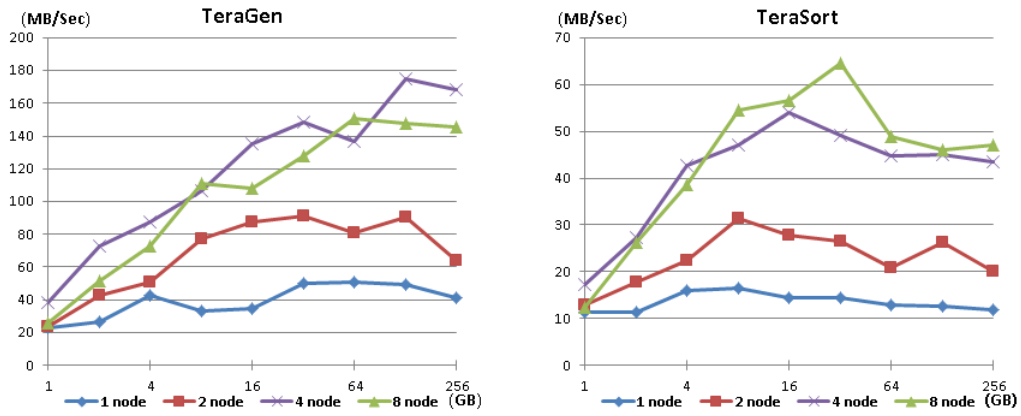

圖 5.11: HDFS 平均資料量處理速率

在進行 S3 Block FileSystem 當作 Share Storage Storage 時，在 TeraGen 的部分與 HDFS 的情況有明顯不同，可以在 S3 的部分觀察到每個數量 node 的曲線都是相近的，因此可以推斷出 Object Storage 的寫入效率仍顯不足，只利用 1 node 以 8 個 Map Task 進行輸出，就已達到 Object Storage 所可處理的寫入速率上限，而在 TeraSort 的情況，由於還包含了讀取與 Shuffe 的情況，因此在多節點的情況完成時間會較單節點時間快，但由於只使用 4 台實體設備，因此 4 node 與 8 node 曲線仍呈現相近狀況。

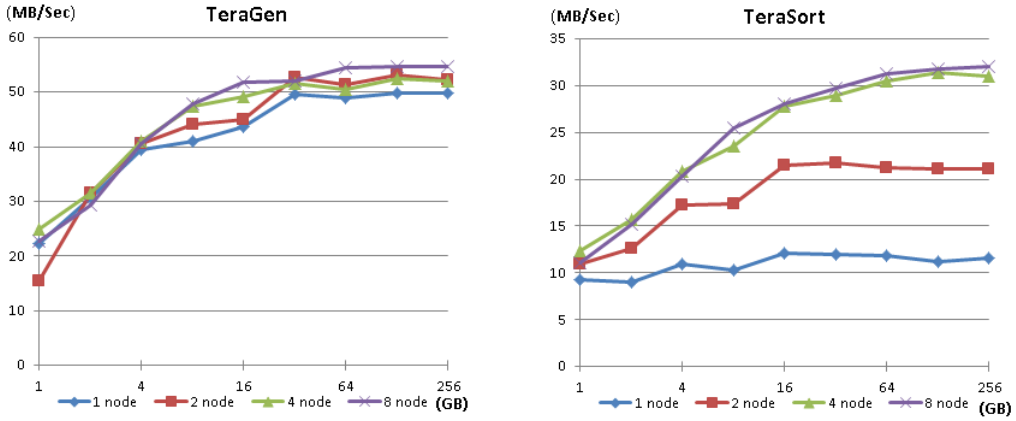

圖 5.12: S3 Block FileSystem 平均資料量處理速率

另外本論文也在 AWS 上進行在 4 node 情況時以相同的實驗進行測試，測試過程中本論文使用 1 台 Master Instance(含 JobTracker 與 NameNode) 與 4 台 Core Instance(含 TaskTracker 與 DataNode)，都採取 M1 Standard Extra Large (m1.xlarge) 擁有 4vCPU 與 4G Ram 的規格進行，映像檔本論文選擇 Amazon’s Hadoop distribution AMI 2.4.2(Hadoop 1.0.3) 進行使用，比較在 AWS 上使用 HDFS 與 S3 Block FileSystem 的差別，其中可以觀察到在 TeraGen aws s3 的曲線有明顯的振幅

出現，在 TeraSort 情況時也有類似的情形出現，因此可以推斷出在 AWS 上使用 S3 服務時，寫入的效率會因其他因素而有所不同，不過整體效率來看，自行架設的服務仍有改善空間。

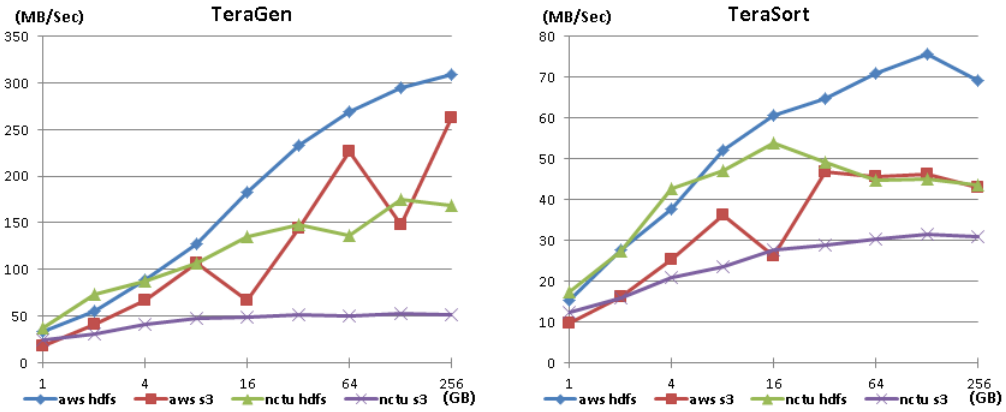

圖 5.13: AWS 與 NCTU 平均資料量處理速率

在使用 AWS EMR 測試期間，也另外紀錄了所花費的費用，過程中本論文進行了圖 5.13 上的測試實驗，透過使用 5 台 m1.xlarge instance 所進行 1~256G 資料量的 HDFS 與 S3 的 TeraGen 與 TeraSort 測試，期間總共花費了 24.39 美金，在 EC2 與 EMR 上總共花費了 40 單位小時為主要花費，分別須付費 EC2 19.2 與 EMR 4.8 美金，詳細明細在表 5.2，因此若是使用自行架設的服務進行小規模測試，將可節省不少支出。

Service	Details	Usage	Cost(US)	Cost(NT)
Elastic Compute Cloud	$0.480 per M1 Standard Extra Large (m1.xlarge) Linux/UNIX instance-hour (or partial hour)	40 Hrs	19.2	576
Elastic MapReduce	$0.12 per m1.xlarge instance hour or partial hour	40 Hrs	4.8	144
Amazon S3 Requests-Tier1	$0.005 per 1,000 PUT, COPY, POST, or LIST requests	32,368 Requests	0.16	4.8
Amazon S3 Requests-Tier2	$0.004 per 10,000 GET and all other requests	123,669 Requests	0.05	1.5
Amazon S3 TimedStorage-ByteHrs	$0.095 per GB - first 1 TB / month of storage used	1.913 GB-Mo	0.18	5.4
Total			24.39	731.7

表 5.2: AWS 收費明細

第 六 章　結論

本論文在 OpenStack 上設計一個類似 EMR 服務的功能，將可透過瀏覽器簡單的操作，隨即架設出 Hadoop Cluster 並且能夠運行程式碼，將可利用 Object Storage 進行資料的蒐集後，再透過 Hadoop Cluster 進行 Hadoop Job 運算，以達到整合的目的，目前在 OpenStack 上與 Private Instance 溝透機制仍有不足，因此本篇論文利用 Object Storage 的設計方式解決溝通上的問題，但目前是透過 instance polling 的方式進行，因此未來應在 OpenStack 上發展出類似 Amazon Simple Queue Service[28] 的服務出來，透過 API 的使用進行 Instance 溝通將會是另一種較好的解決方式，在本校的系統效能測試當中，可觀察到 OpenStack 中 Instance 與 Object Storage 若無特別的架構設計，在 IO 處理效率上仍有改善的空間，因此未來可對 OpenStack 與 Object Storage 上進行效能的改進研究，來提升服務的品質。

參考文獻

[1] Amazon EC2 Beta http://aws.typepad.com/aws/2006/08/amazon_ec2_beta.html

[2] Open source software for building private and public clouds. http://www.openstack.org

[3] Mell, P., Grance, T. The NIST definition of cloud computing. Version 15. National Institute of Standards and Technology, October 7, 2009.

[4] Django:The Web framework for perfectionists with deadlines https://www.djangoproject.com

[5] Paste Deployment http://pythonpaste.org/deploy/

[6] Eventlet Documentation http://eventlet.net/doc/index.html

[7] PEP 333 - Python Web Server Gateway Interface v1.0 http://www.python.org/dev/peps/pep-0333/

[8] Roy Thomas Fielding, ”Architectural styles and the design of network based software architectures.” PhD diss., University of California, 2000.

[9] AMQP and Nova http://docs.openstack.org/developer/nova/devref/rpc.html

[10] OpenStack Logical Architecture http://docs.openstack.org/admin-guide-cloud/content/ch_getting-started-with-openstack.html#logical-architecture

[11] CinderSupportMatrix https://wiki.openstack.org/wiki/CinderSupportMatrix

[12] Holmes, Alex ”What is Hadoop?” In Hadoop in Practice pp. 3-9

[13] Hadoop Wiki MapReduce http://wiki.apache.org/hadoop/MapReduce

[14] HDFS High Availability http://hadoop.apache.org/docs/current2/hadoop-yarn/hadoop-yarn-site/HDFSHighAvailabilityWithNFS.html

[15] Configuring High Availability for the JobTracker http://www.cloudera.com/content/cloudera-content/cloudera-docs/CM4Ent/4.6.2/Cloudera-Manager-Managing-Clusters/cmmc_JT_high_avail.html

[16] The Hadoop Distributed File System: Architecture and Design http://hadoop.apache.org/docs/r0.18.0/hdfs_design.pdf

[17] Weil, S. A., S. A. Brandt, et al. ”Ceph: A scalable, high-performance distributed file system.”, Operating Systems Design and Implementation (OSDI), pp. 307-320, 2006.

[18] Ceph Architecture http://ceph.com/docs/master/architecture

[19] Weil, S. A., S. A. Brandt, et al. ”CRUSH: Controlled, scalable, decentralized placement of replicated data”, In Proc. Supercomputing (SC), pp. 122, 2006.

[20] CEPH: RELIABLE, SCALABLE, AND HIGH-PERFORMANCE DISTRIBUTED STORAGE http://ceph.com/papers/weil-thesis.pdf

[21] Robert J. Jenkins Jr., 1995-1997 Hash functions for hash table lookup. http://burtle-burtle.net/bob/hash/evahash.html

[22] Plan an Amazon EMR Cluster https://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-plan-instances.html

[23] CloudInit Ubuntu Official Documentation https://help.ubuntu.com/community/CloudInit

[24] Init scripts for use on cloud images http://bazaar.launchpad.net/~cloud-init-dev/cloud-init/trunk/files/head:/doc/examples/

[25] Amazon Elastic MapReduce ,Launch a Streaming Cluster http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/CLI_CreateStreaming.html

[26] Custom Horizon https://github.com/fajoy/horizon

[27] Load balancing with Varnish https://www.varnish-cache.org/trac/wiki/LoadBalancing

[28] Amazon Simple Queue Service (Amazon SQS) http://aws.amazon.com/sqs/

附　錄

附錄 A　介面使用情形

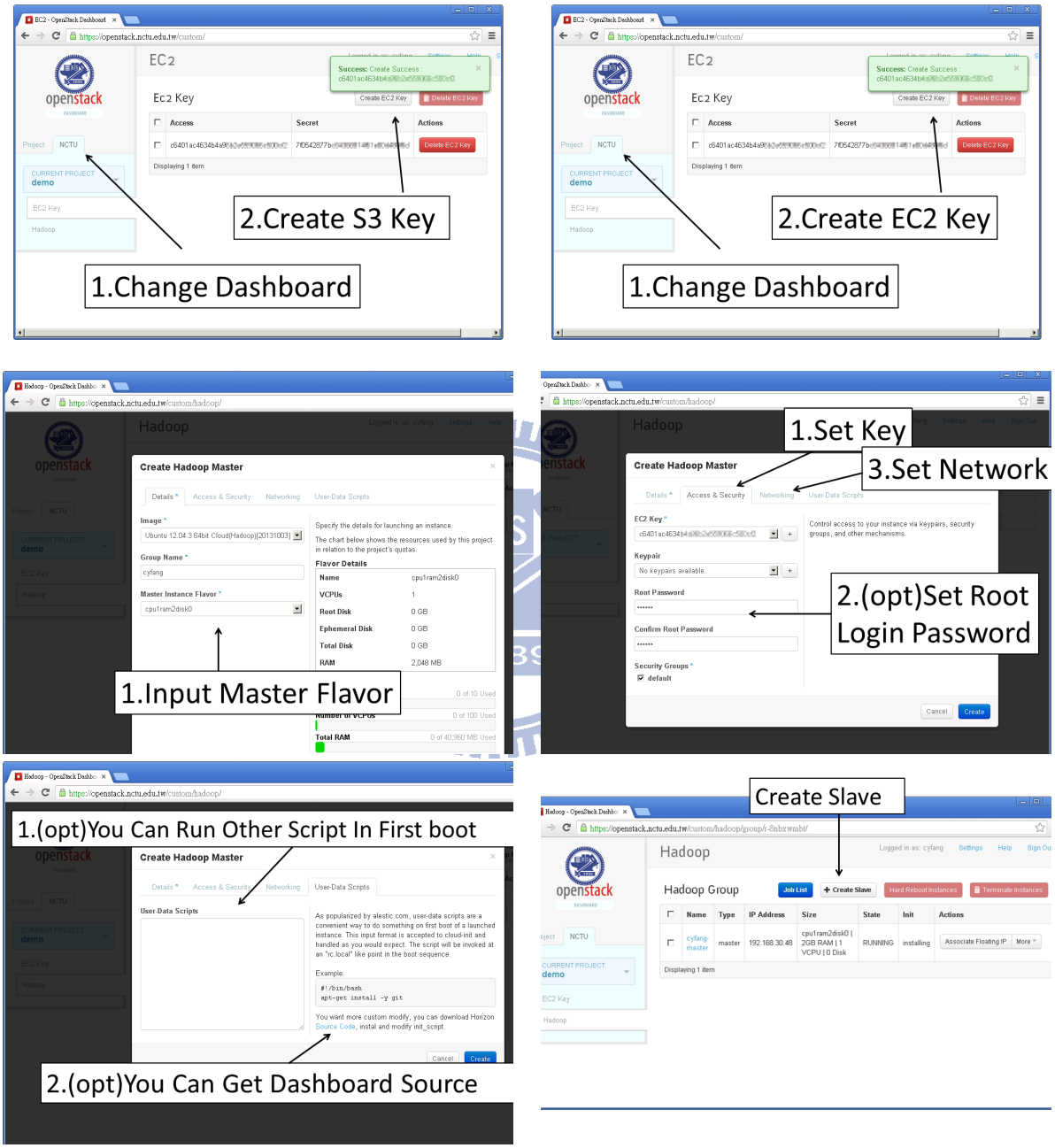

圖 A.1: Create Ec2 Key and Hadoop Cluster

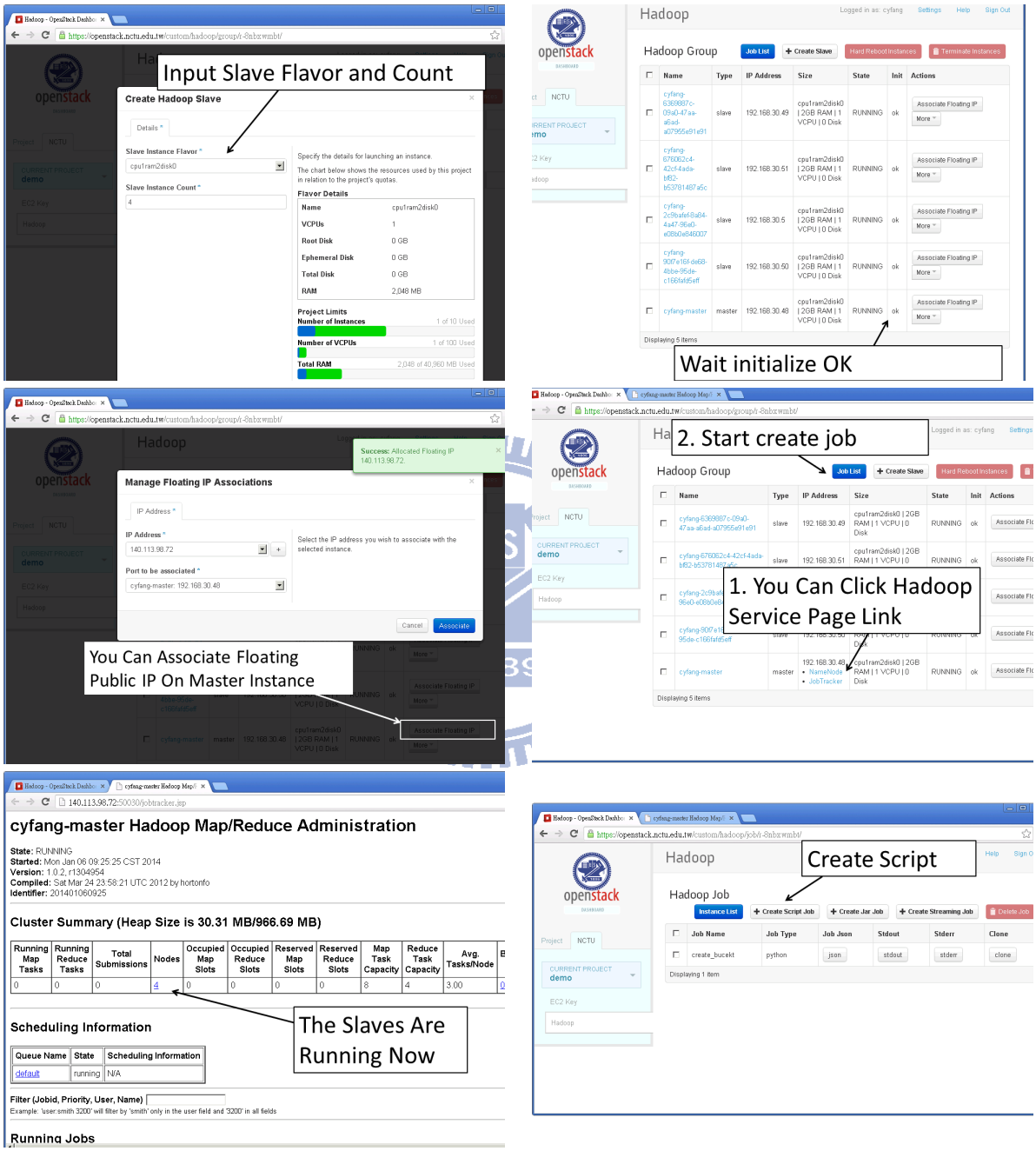

圖 A.2: Check Hadoop Cluster

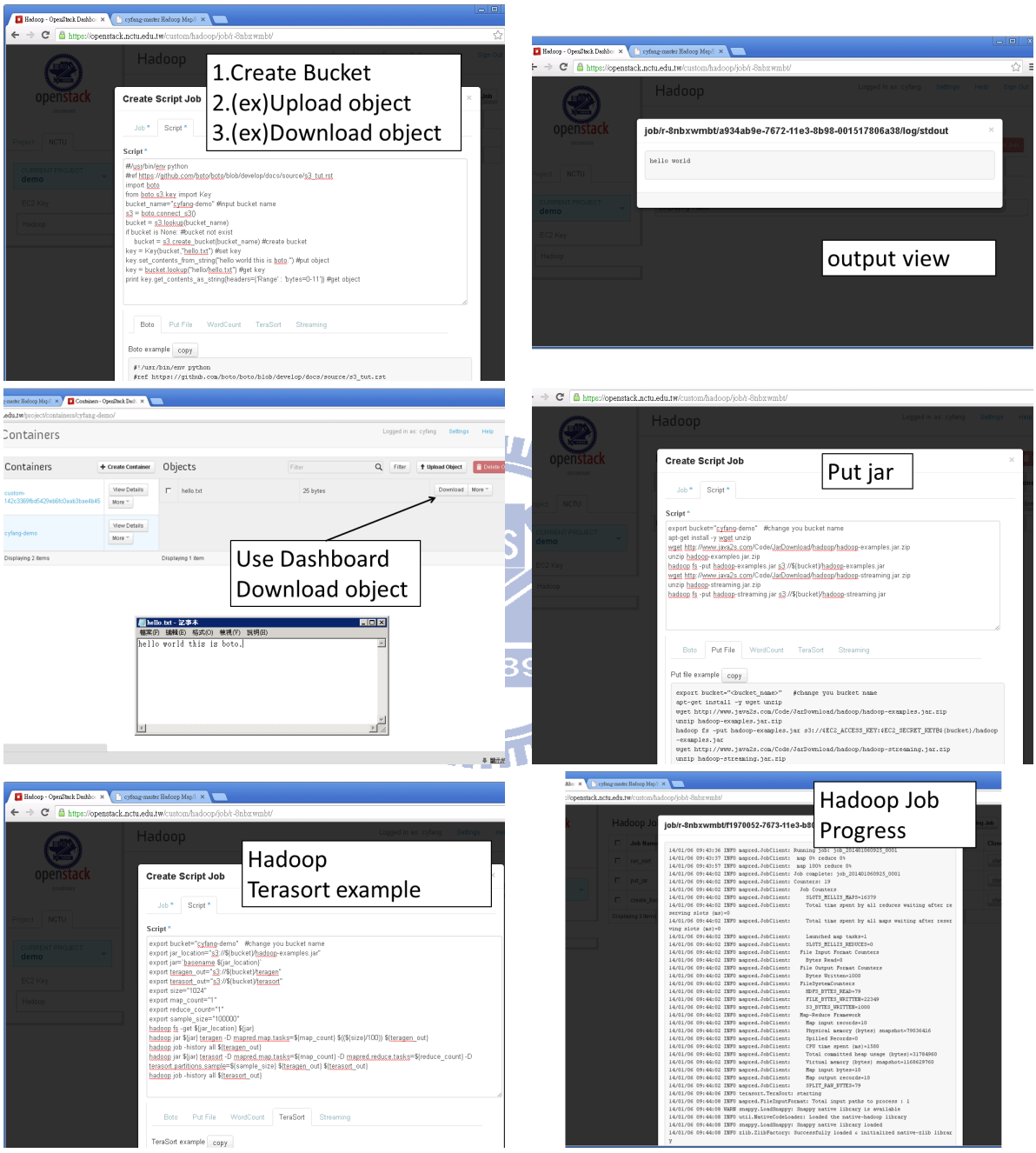

圖 A.3: Run hadoop job by dashboard

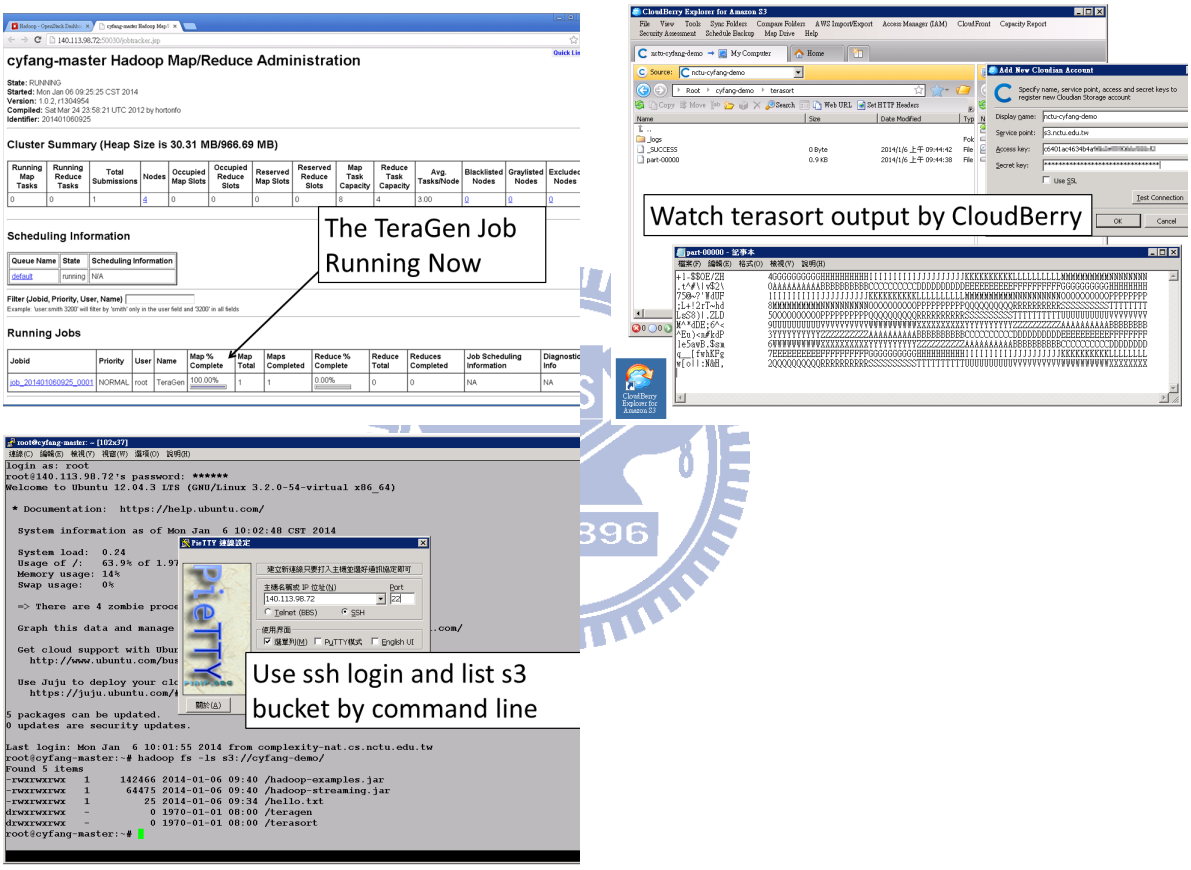

圖 A.4: Use other tools