

國立交通大學

資訊科學系

碩士論文

運用雲端運算在系統容易出現迫切需求：
用亞馬遜 Web 服務(AWS)

Applying Cloud Computing to Systems Prone to Pressing Demand:
Using Amazon Web Services (AWS)

研究生：黎高昆
指導教授：林寶樹 教授

中華民國一百零二年八月

運用雲端運算在系統容易出現迫切需求：
用亞馬遜 Web 服務(AWS)

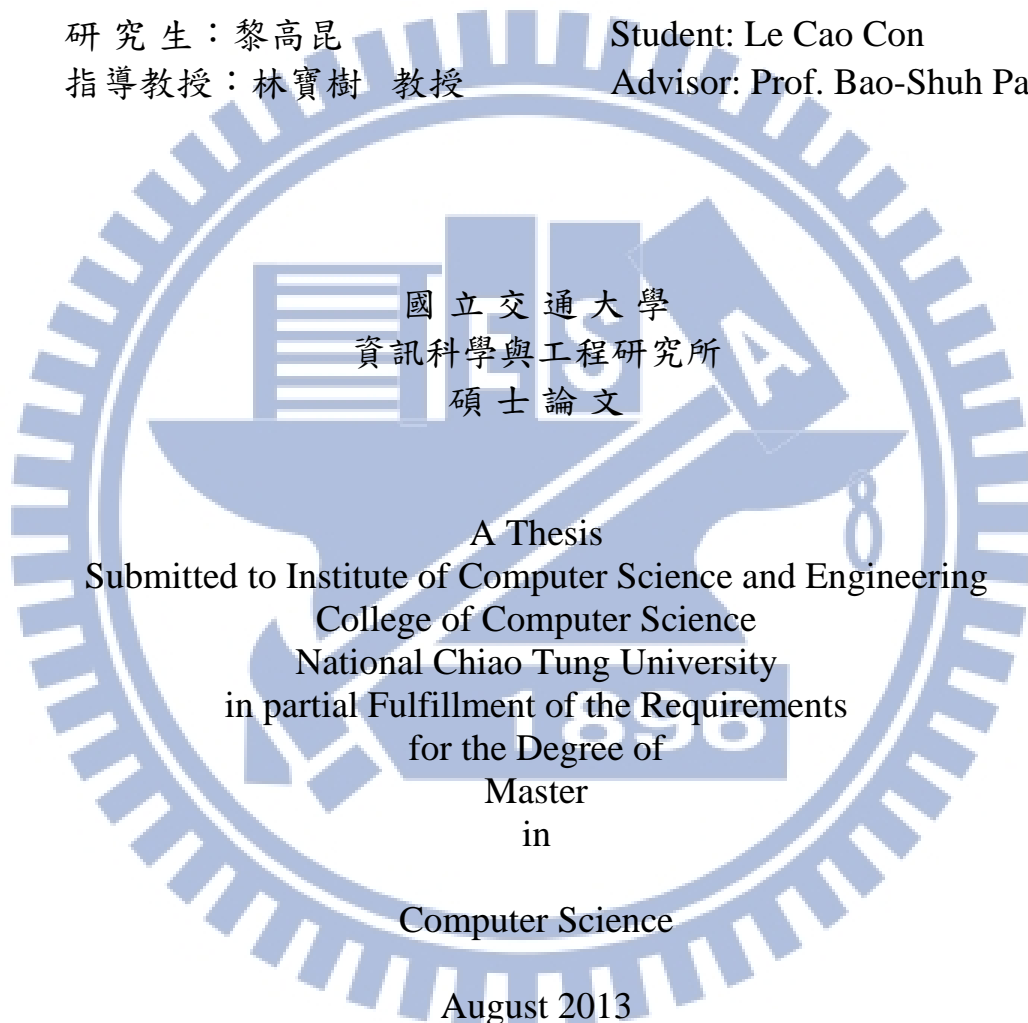
Applying Cloud Computing to Systems Prone to Pressing Demand:
Using Amazon Web Services (AWS)

研究生：黎高昆

Student: Le Cao Con

指導教授：林寶樹 教授

Advisor: Prof. Bao-Shuh Paul Lin



Hsinchu, Taiwan, Republic of China

中華民國一百零二年八月

運用雲端運算在系統容易出現迫切需求： 用亞馬遜 Web 服務(AWS)

研究生：黎高昆

指導教授：林寶樹 教授

國立交通大學
資訊科學與工程研究所 碩士班

摘要

雲端運算的使用，在技術及商業界已經司空見慣。雲端服務供應商的數量一直不斷增長，因此有一系列給遠端用戶的選擇。也有一些優秀的應用，如 Dropbox 的發明，檔案託管服務，提供雲端儲存以及文件同步等等。

藉由一些課程，我學習到與雲端運算相關的大量的知識，我參加了一些課程並做一個敏銳的觀察者，我意識到，這樣的系統如選課系統（例如，在交大的選課系統）或在我的國家，越南的鐵路銷售系統，像這樣的系統有時不太穩定。

當網站流量激增、流量過量時，將造成伺服器負載急劇增加而影響伺服器上的網路連接。比方說針對選課系統來說在學校讓學生選課時有時候會有超多學生同時一起來跟系統連結，這會造成伺服器當機或是反應很慢。另外一個系統像鐵路銷售系統在過年時有許多人為了工作而

離開家裡從北部移到南部，然後過年時要回家團圓，因為他們同時跟系統連結，所以這也會造成一個系統像上面提到一樣的。

針對類似上面提到的系統我們會有什麼辦法又簡單又便宜能夠解決短時間迫切需求之系統呢？

亞馬遜 **Web 服務(AWS)**是一家雲端運算供應商，他們提供許多服務又便宜又有大彈性。所以我認為這就是我們的選擇之一。在我的研究，我建好一個簡單的系統跟選課系統差不多，然後把這個系統移轉到遠端去，接下來是針對這兩個系統做評估看它們的效率如何。最後我建議在雲端可以建一個系統，它的架構包含一個域名系統 **Web 服務**、三個負載平衡器、三到十八台虛擬 **Web 伺服器**、和一台資料庫伺服器。**Web 伺服器**的數量會變動是因為靠使用者的數量同時供用讓整個系統做自動縮放。假設我們本來的系統頻寬大概 **1Gbps**，然後跟新的系統做一個比較，新的系統能力最大可以比本來的系統能力高三倍。而且新的系統也不算貴，每學期大概要花美金 **600 元**。

完成這個研究，我有一個結論是上面類似之系統可以改善它的情況如果有應用雲端運算。

Applying Cloud Computing to Systems Prone to Pressing Demand: Using Amazon Web Services (AWS)

Student: Le Cao Con

Advisors: Prof. Bao-Shuh Paul Lin

Institute of Computer Science and Engineering
National Chiao Tung University

ABSTRACT

Cloud computing has been a commonplace and widely used in both technological and business world. The number of cloud providers has been constantly growing and so has the set of options for end users. There have also been some outstanding applications such as the invention of Dropbox, a file hosting service that offers cloud storage and file synchronization.

With a great deal of knowledge on cloud computing from some courses that I took in the master program and as a keen observer, I realized that such systems as the course selection system (for example the one that operates at NCTU) or the railway ticket system back in my country, Vietnam, do not work well sometimes.

There are times that websites are overloaded to a point when their services are degraded or disrupted entirely. This web traffic overload happens when there is a large surge in traffic to a particular website causing a dramatic increase in server load and putting severe strain on the network links leading to the server. For example many students access the course selection system

at the time the school lets the students select their courses on it. Also, many people want to buy railway tickets on railway ticket system at the time closing to New Year vacation when they want to go home.

For those systems, is there any practical way to improve them with low cost but high degree of effectiveness?

Amazon Web Services (AWS), a cloud computing provider, provides a number of services with low cost and rapid elasticity. Therefore, this should be considered as a choice for us. In this research, I am going to conduct several experiments with an aim to migrate a sample system like a course selection system to AWS and evaluate the effects of the new system. Furthermore, I expect to propose a model for the new system on the cloud which includes one domain name system web services, three load balancers, three to eighteen web servers, and one database server. The number of web servers might be changed depending on how many users access the new system at the same time. It's assumed that the original system has the capacity bandwidth of 1Gbps. As well as that if we compare the new system with the original one, the capacity of the new system is approximately 3 times as big as that of the original system. Furthermore, the cost of new system is relatively low estimated at \$600 for every semester.

After doing this research, I have reached the conclusion that the overload system may be improved by applying cloud computing technology.

ACKNOWLEDGEMENTS

In my exploring of knowledge and in the course of completing my thesis, many individuals have assisted me. I would like to acknowledge wholeheartedly their assistance, cooperation and encouragement which all contributed in making this study possible. Without them, this study would not have been completed.

First, my appreciation goes to my advisor, Prof. Bao-Shuh Paul Lin. He has guided me through the completion of the master program and through this thesis. He discussed with me patiently, carefully and challenged me to think critically. He has constantly provided me with great sympathy, encouragement and support and most importantly trusted my capacity as a researcher. My sincere gratitude also goes to Dr. Li-Ping Tung who eagerly exchanged her ideas with mine and did not hesitate to share her resources. It is an honor for a student like me to have them as advisors during my research and role models for my forthcoming profession.

I also wish to thank other teachers and staff from the Institute of Computer Science and Engineering, NCTU, my friends, my classmates and my roommates back in the dormitory for creating an academic and friendly environment for me. Without their expectations, patience and cooperation, I may have struggled with this project.

Finally, a million thanks go to my families, in particular my wife and my adorable daughter who always take care of me and support me with understanding and kindness. Their love has given me enormous strength to overcome all the difficulties that a student has to face up with while studying abroad. As my appreciation, my heart goes with them forever.

TABLE OF CONTENTS

摘要	i
ABSTRACT	iii
ACKNOWLEDGEMENTS	v
TABLE OF CONTENTS	vi
LIST OF FIGURES	viii
1 Introduction	1
1.1 Motivation	1
1.2 Objective	1
1.3 Research Contribution	2
1.4 Thesis Organization	2
2 Theoretical Background	3
2.1 Cloud Computing	3
2.1.1 Essential Characteristics of Cloud Computing	4
2.1.2 Service models	5
2.1.3 Cloud Deployment Models	6
2.2 Amazon Web Services (AWS)	7
2.2.1 Why AWS	7
2.2.2 Services	8
3 Architecture	14
3.1 Website Overload Reasons	14
3.2 Website Overload Solutions	17

3.2.1	Expanded the Original System.....	17
3.2.2	Applying AWS to the Original System	20
3.3	Doing Migrations.....	22
3.3.1	Database Migration	24
3.3.2	Web server Migration	26
3.4	Security problems	28
4	Implementation	31
4.1	Experiments.....	31
4.1.1	Choice software for simulation.....	32
4.1.2	Test link speed 100Mbps	33
4.1.3	Test link speed 1Gbps	34
4.1.4	Test with Amazon Web Services	35
4.2	Monitoring System Scaling	38
4.2.1	Requirements.....	38
4.2.2	Structure	39
4.2.3	Result.....	42
4.3	Cost of using Amazon Web Services for Course Selection System ..	43
5	Conclusion and Future Works.....	45
6	REFERENCES.....	46
	Appendix 1: AWS Scaling.....	48
	Appendix 2: Database migration.....	51
	Appendix 3: Web migration.....	54

LIST OF FIGURES

Figure 2.1: Cloud computing logical diagram (Wikipedia.org)	3
Figure 2.2: Cloud computing types (Wikipedia.org)	6
Figure 3.1: General web farm architecture [15].....	14
Figure 3.2: Database Server overload.....	15
Figure 3.3: Web Servers overload.....	16
Figure 3.4: Load Balancer overload.....	16
Figure 3.5: Database Server overload solution.....	17
Figure 3.6: Web Servers overload solution.....	18
Figure 3.7: Load Balancer overload solution.....	19
Figure 3.8: Extended original system with the AWS	20
Figure 3.9: A normal website system.....	21
Figure 3.10: New system on AWS model.....	21
Figure 3.11: Equivalent of new system model.....	22
Figure 3.12: Migration from On-Premise to Off-Premise	23
Figure 3.13: modeling database migration.....	24
Figure 3.14: Modeling web server migration	26
Figure 3.15: AWS EC2 Key Pair	28
Figure 3.16: AWS EC2 Security Groups	29
Figure 3.17: DB Security Groups	30
Figure 3.18: AWS security credentials	30
Figure 4.1: Simulation webpage	31

Figure 4.2: Jmeter GUI	32
Figure 4.3: Simulation 100Mbps link	33
Figure 4.4: Simulation 1Gbps link.....	34
Figure 4.5: AWS experiment architecture	35
Figure 4.6: 02 AWS EC2 m1.large instances – 02 Elastic Load Balancers ...	35
Figure 4.7: 04 AWS EC2 m1.large instances – 02 Elastic Load Balancers ...	36
Figure 4.8: 06 AWS EC2 m1.large instances – 02 Elastic Load Balancers ...	36
Figure 4.9: 06 AWS EC2 m1.large instances – 03 Elastic Load Balancers ...	37
Figure 4.10: 12 AWS EC2 m1.large instances – 03 Elastic Load Balancers .	37
Figure 4.11: Running 18 AWS EC2 large instances and RDB db.m2.2xlarge	38
Figure 4.12: Running 18 AWS EC2 large instances and RDB db.m2.4xlarge	38
Figure 4.13: Replace Nameserver	39
Figure 4.14: AWS scale-in.....	40
Figure 4.15: AWS scale-out.....	41
Figure 4.16: AWS scaling	42
Figure 4.17: References of server pricing.....	44

Chapter 1:

Introduction

1.1 Motivation

Recently while selecting courses, I have always felt that NCTU's system do not respond very well. One reason might be that many students access the servers at the same time, which possibly makes the servers overload. In general, the course selection system just requires more power at the time the students choose their courses at the beginning and the end of every semester.

Another system is railway ticket selling system. It seems appears difficult for passengers to order tickets when the time is close to vacations, especially in New Year. A lot of people want to travel or go home, so the amount of access to train ticket system is suddenly increased. Sometimes such escalating demand shuts the system down completely, and then no one can order tickets. In my country, actually, numerous workers working in the South but want to head back by train to the North where their families and homes are before the Traditional New Year vacation, and then come back to work after the holiday is finished. However, they cannot order train tickets via the system and, as a result, they have to buy expensive tickets by some other ways such as through middlemen in the black market.

1.2 Objective

As a matter of fact, cloud computing is very popular nowadays. One of the features of cloud computing is that it is elastic. This means we can change any resource appropriately and quickly on demand. It is very easy to scale-in or scale-out while the cost is based on actual usage. Therefore, it occurs to me

that we can apply cloud computing to our course selection system or the system of selling railway tickets. When we need our system to be more powerful, we just scale-out our system. Otherwise, we scale-in or use the original system only. In that way a lot of money can be saved to buy new devices and maintain them. The basis is pay as we go.

My research purpose is to construct a tool which could automatically attach some virtual Web servers that are run on AWS, migrate Database server to Amazon cloud and scale it up or down if needed when the system responds too slowly. That enables the managers to manage and scale the system as they want. After that, synchronize the database back with the original database when we do not need to use AWS.

1.3 Research Contribution

This research hopes to identify the causes of and work towards solutions to the problem of congestion in websites and system with suddenly and rapidly increased access at given time.

1.4 Thesis Organization

The thesis is organized as follows. Chapter 2 presents an overview of Cloud Computing and Amazon Web Services. Chapter 3 illustrates why a website can be overloaded and proposes some solutions. Chapter 4 presents experiments with web traffic load. Finally, Chapter 5 states the conclusion and envisages future work.

Chapter 2:

Theoretical Background

2.1 Cloud Computing

Cloud computing is a recent computing concept that describe a lot of computers which are connected together through communication network. There exist various definitions of cloud computing. For example, as is put by National Institute of Standards and Technology Definition of Cloud Computing, “Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction” [6]. Also, according to webopedia.com, a famous online computer dictionary for computer and internet, “Cloud computing is a type of computing that relies on sharing computing resources rather than having local servers or personal devices to handle applications.” [10]

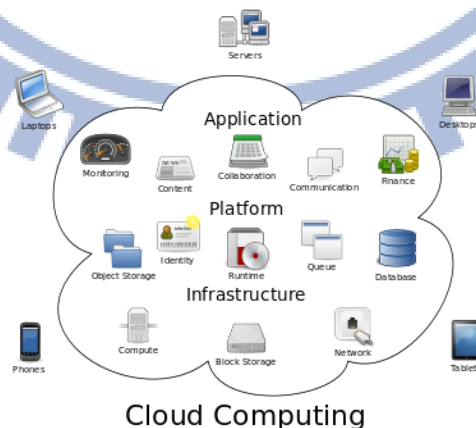


Figure 2.1: Cloud computing logical diagram (Wikipedia.org)

2.1.1 Essential Characteristics of Cloud Computing

Cloud computing has the following characteristics: [5] [6]:

✓ *On-demand self-service*

Consumer can use resources as much as they want. They can provision any amount of computers, network storages at anytime and anywhere without requiring human interaction.

✓ Broad network access

Consumer can use any kind of devices such as desktops, laptops, mobile phones, handhelds, etc. to access cloud computing services as long as they can access the Internet.

✓ Resource pooling

Computing resources are provided like a pool where consumers can use many kinds of resources such as virtual machine, storage, bandwidth, etc. However, they do not know where the resources exactly are; they just know the high level of abstraction (ex: country, region). In other words, the location of resources is invisible to end users.

✓ Rapid elasticity

Consumers can scale their application in or out very quick as the resources seem unlimited and can satisfy any requirement.

✓ Measured service

Cloud systems provide metering capability that helps consumers monitor their used resources such as CPU usage, data storage, network bandwidth, etc.)

2.1.2 Service models

Cloud computing services can be classified into several kinds of services such as Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS), Network as a Services (NaaS), Database as a Service (DaaS), and XaaS - anything as a service.

- ✓ Infrastructure as a service (IaaS)

In this kind of service, cloud providers offer a complete infrastructure such as machine, network, firewall, load balancer, disk image, etc. Therefore, consumers do not need to care about hardware layer nor do they have to manage low layer. However, when they want to deploy an application, they not only have to install operating systems on the cloud infrastructure by themselves, but also have to install the required environment. And the consumers have to manage and maintain their own operating systems as well as applications.

- ✓ Platform as a service (PaaS)

In this model, cloud providers supply an environment that includes operating systems, database server, web server, and programming environment. Consumers can deploy their application without worrying about underlying hardware and software layers. Providers have to take care of the underlying layers including keeping them up to date. Consumers will take care of their application only. In some cases, cloud providers can scale their resources automatically to match the consumers' demands.

- ✓ Software as a service (SaaS)

This is the highest level of cloud computing services where consumers can rent or buy application software directly. Cloud providers make sure that

these software are up – to - date. Consumers do not need to manage the cloud infrastructure and platform.

2.1.3 Cloud Deployment Models

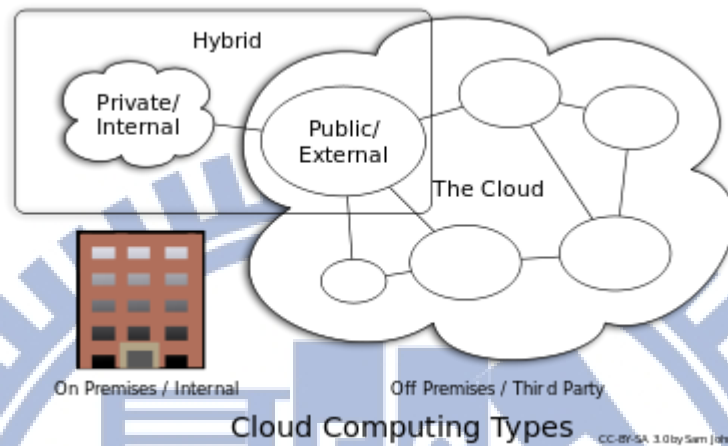


Figure 2.2: Cloud computing types (Wikipedia.org)

✓ Private cloud

Private cloud is only for a single organization. It is hosted at On-Premise (internal) or Off-Premise (external). Those who want to develop private cloud must be experts who have significant knowledge about system and system security. Every single security issue must be addressed to protect the organization from any potential vulnerabilities.

✓ Public cloud

Public cloud is open for public use. Therefore, it does not need high level of security as required by private cloud. However, the architecture of both private cloud and public cloud is the same.

✓ Community cloud

Community cloud is a cloud model that stands between private cloud and public cloud. This cloud model is made when some organizations share

infrastructure. Therefore, the number of users of this model are fewer than that of public cloud that is used all over the world and bigger than that of private cloud which is used within certain organization. That means community cloud does not help to save cost.

✓ Hybrid Cloud

Hybrid cloud is comprised of two or more of the aforementioned cloud models to take advantage of multiple deployment models to satisfy some temporary needs

2.2 Amazon Web Services (AWS)

Amazon Web Services offers a complete set of infrastructure and application services that allows everything from enterprise applications and big data projects to social games and mobile applications to be run virtually in the cloud [11].

2.2.1 Why AWS

Amazon Web Services provides a flexible, scalable, low-cost cloud computing platform for businesses of all sizes all around the world.

- *Pay as you go:* Consumers pay for exactly what they use. Except for AWS monthly free usage tier; there are no long-term contracts or up-front commitments.
- *Scalable:* With AWS, any application can be easily scaled in/out or up/down when needed as a result of a massive infrastructure that is provided by Amazon.
- *Flexible:* Every task from operating system to programming language, or those related to web application platform, software,

database server, etc. is performed flexibly. If an application can run On-Premise, it can run in the cloud.

- *Easy to use:* AWS can be started within few minutes. We have many ways to do what we need by using AWS Management Console, APIs, and Command Line Tools.

2.2.2 Services

AWS provides many following products and services:

- ❖ Compute
 - Amazon Elastic Compute Cloud (EC2)
 - Amazon Elastic MapReduce
- ❖ Auto Scaling
 - Elastic Load Balancing (ELB)
 - Content Delivery
 - Amazon CloudFront
- ❖ Database
 - Amazon Relational Database Service (RDS)
 - Amazon DynamoDB
 - Amazon ElastiCache
 - Amazon Redshift
- ❖ Deployment & Management
 - AWS Identity and Access Management (IAM)
 - Amazon CloudWatch
 - AWS Elastic Beanstalk
 - AWS CloudFormation
 - AWS Data Pipeline
 - AWS OpsWorks
 - AWS CloudHSM
- ❖ Application Services
 - Amazon CloudSearch
 - Amazon Simple Workflow Service (SWF)
 - Amazon Simple Queue Service (SQS)
 - Amazon Simple Notification Service (SNS)
 - Amazon Simple Email Service (SES)
 - Amazon Elastic Transcoder
- ❖ Software
 - AWS Marketplace

- ❖ Networking
 - Amazon Route 53
 - Amazon Virtual Private Cloud (VPC)
 - AWS Direct Connect
- ❖ Payments & Billing
 - Amazon Flexible Payments Service (FPS)
 - Amazon DevPay
- ❖ Storage
 - Amazon Simple Storage Service (S3)
 - Amazon Glacier
 - Amazon Elastic Block Store (EBS)
 - AWS Import/Export
 - AWS Storage Gateway
- ❖ Support
 - AWS Support
- ❖ Web Traffic
 - Alexa Web Information Service
 - Alexa Top Sites
- ❖ Workforce
 - Amazon Mechanical Turk
- **Amazon Elastic Compute Cloud (EC2)**

Amazon EC2 is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers.

Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing you to quickly scale capacity, both up and down, as your computing requirements change.

Amazon EC2 presents a true virtual computing environment, allowing you to launch instances with a variety of operating systems, load them with your custom application environment, manage your network's access permissions, and run your image using as many or few systems as you desire.

Service Highlights:

- ✓ **Elastic:** we can scale the number of instances within minutes. Furthermore, we can launch thousands of server instances simultaneously.
- ✓ **Completely Controlled:** We have complete control of our instances.
- ✓ **Flexible:** We have multiple choices of instance types, operating systems, and software packages.
- ✓ **Designed for use with other Amazon Web Services:** It is tightly integrated with other Amazon Web Services.
- ✓ **Reliable:** Amazon EC2 offers a highly reliable environment; the Service Level Agreement commitment is 99.95% availability for each Amazon EC2 Region.
- ✓ **Secure:** Amazon EC2 works in conjunction with Amazon Virtual Private Cloud (VPC) to provide security and robust networking functionality for our compute resources.
- ✓ **Inexpensive:** We pay a very low rate for the compute capacity we actually consume.
- ✓ **Easy to Start:** Amazon EC2 provides preconfigured software on Amazon Machine Images (AMIs), so we can quickly deploy this software to EC2 via 1-Click launch, EC2 console, and API functions.

- **Elastic Load Balancing**

Elastic Load Balancing automatically distributes incoming application traffic across multiple Amazon EC2 instances. It can detect unhealthy instances and automatically reroutes traffic to healthy instances until the unhealthy instances are detected becoming healthy instances.

Customers can enable Elastic Load Balancing within a single Availability Zone or across multiple zones for even more consistent

application performance. Elastic Load Balancing can also be used in an Amazon Virtual Private Cloud (“VPC”) to distribute traffic between application tiers.

- **Amazon Relational Database Service (Amazon RDS)**

Amazon Relational Database Service (Amazon RDS) is a web service that makes it easy to set up, operate, and scale a relational database in the cloud. It provides cost-efficient and resizable capacity while managing time-consuming database administration tasks, freeing you up to focus on your applications and business.

Amazon RDS gives you access to the capabilities of a familiar MySQL, Oracle or Microsoft SQL Server database engine. Amazon RDS automatically patches the database software and backs up your database, storing the backups for a user-defined retention period and enabling point-in-time recovery. You benefit from the flexibility of being able to scale the compute resources or storage capacity associated with your Database Instance (DB Instance) via a single API call.

Service Highlights:

- ✓ **Simple to Deploy:** Easy to create new database server by using API calls or AWS Management Console in minutes without any worry about underlying hardware or software level.
- ✓ **Managed:** Amazon RDS handles time-consuming database management tasks, such as backups, patch management, and replication, allowing you to pursue higher value application development or database refinements.

- ✓ **Compatible:** With Amazon RDS, you get native access to a relational database. This facilitates compatibility with your existing tools and applications.
 - ✓ **Fast, Predictable Performance:** Amazon RDS Provisioned IOPS is a high performance storage option designed to deliver fast, predictable, and consistent performance for I/O intensive transactional database workloads.
 - ✓ **Scalable:** You can easily scale your database to meet your application needs by using API function or the AWS Management Console.
 - ✓ **Reliable:** Many features such as automated backups, DB snapshots, automatic host replacement, and Multi-Available Zone, enhances the level of reliability for our database.
 - ✓ **Designed for use with other Amazon Web Services:** Amazon RDS is tightly integrated with other Amazon Web Services.
 - ✓ **Secure:** Amazon RDS provides a number of mechanisms to secure your DB Instances. It includes configure firewall settings that control network access to your database. And it also allows you to run your DB Instances in Amazon Virtual Private Cloud (Amazon VPC). Amazon VPC enables you to isolate your DB Instances by specifying the IP range you wish to use, and connect to your existing IT infrastructure through industry-standard encrypted IPsec VPN.
 - ✓ **Inexpensive:** You pay very low rates and only for the resources you actually consume.
- **Amazon Simple Queue Service (Amazon SQS)**

Amazon Simple Queue Service (SQS) is a fast, reliable, scalable, fully managed queue service. SQS makes it simple and cost-effective to decouple

the components of a cloud application. You can use SQS to transmit any volume of data, at any level of throughput, without losing messages or requiring other services to be always available.

- **Amazon CloudWatch**

Amazon CloudWatch provides monitoring for AWS cloud resources and the applications customers run on AWS. Developers and system administrators can use it to collect and track metrics, gain insight, and react immediately to keep their applications and businesses running smoothly. Amazon CloudWatch monitors AWS resources such as Amazon EC2 and Amazon RDS DB instances, and can also monitor custom metrics generated by a customer's applications and services.

Amazon CloudWatch lets you programmatically retrieve your monitoring data, view graphs, and set alarms to help you troubleshoot, spot trends, and take automated action based on the state of your cloud environment.

- **Amazon Route 53**

Amazon Route 53 is a highly available and scalable Domain Name System (DNS) web service. It is designed to give developers and businesses an extremely reliable and cost effective way to route end users to Internet applications by translating human readable names like www.example.com into the numeric IP addresses like 192.0.2.1 that computers use to connect to each other. Route 53 effectively connects user requests to infrastructure running in Amazon Web Services (AWS) – such as an Amazon EC2 instance, an Amazon Elastic Load Balancer, an Amazon CloudFront distribution, or an Amazon Simple Storage Service (Amazon S3) bucket – and can also be used to route users to infrastructure outside of AWS.

Chapter 3:

Architecture

For the websites of small and medium – sized agencies, investing in the hardware system is a difficult puzzle to solve because of several reasons. First and foremost, substantial investment for the initial hardware system will be an expensive and needless waste if actual demand is too low. On the other hand, a modest sum leads to the situation in which the system will not be able to cope well with a sudden surge in demand at certain points. This is even more challenging for websites with very low level of usual access but with occasional massive traffic. The question of how much to spend or invest poses a genuine dilemma for the designers and operators of such systems.

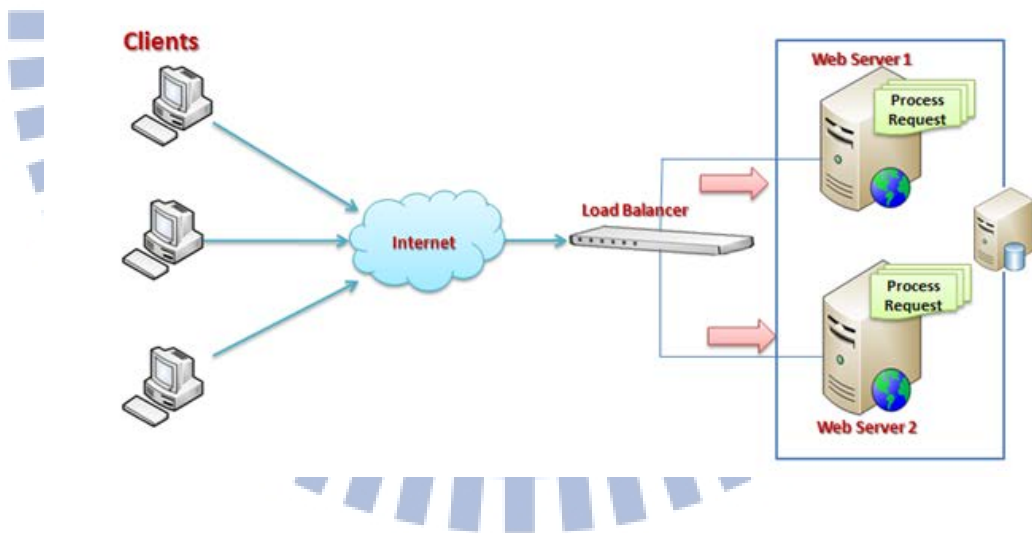


Figure 3.1: General web farm architecture [15]

3.1 Website Overload Reasons

General website models typically shown in figure 3.1 included some main components such as Load Balancer, Web Server, Database Server, network link, firewall, etc. Website is overload when we let the system do a

work exceeds its ability [3]. A website overloaded should be due to one of the three following reasons:

3.1.1 Database Server Overload

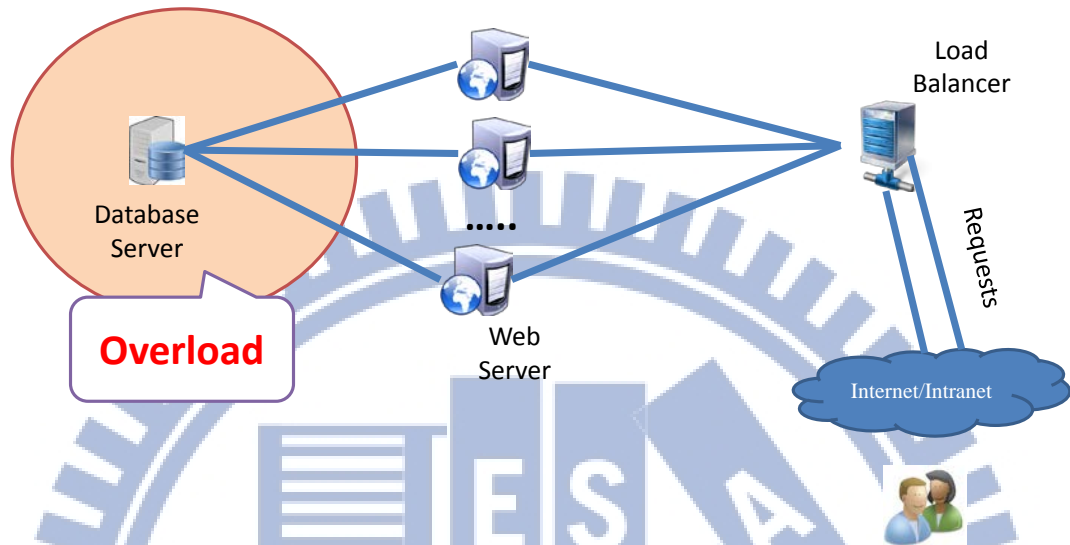


Figure 3.2: Database Server overload

If a system's Load Balancer and Web Servers are good enough, Load Balancer can satisfy massive data going through and if Web Servers have strong configuration, the reason for our website system overload maybe is our Database Server is overloaded. Of course, like Web Servers, Database Server is overloaded because its CPU is overloaded or memory overloaded or bandwidth overloaded, etc.

3.1.2 Web Servers Overload

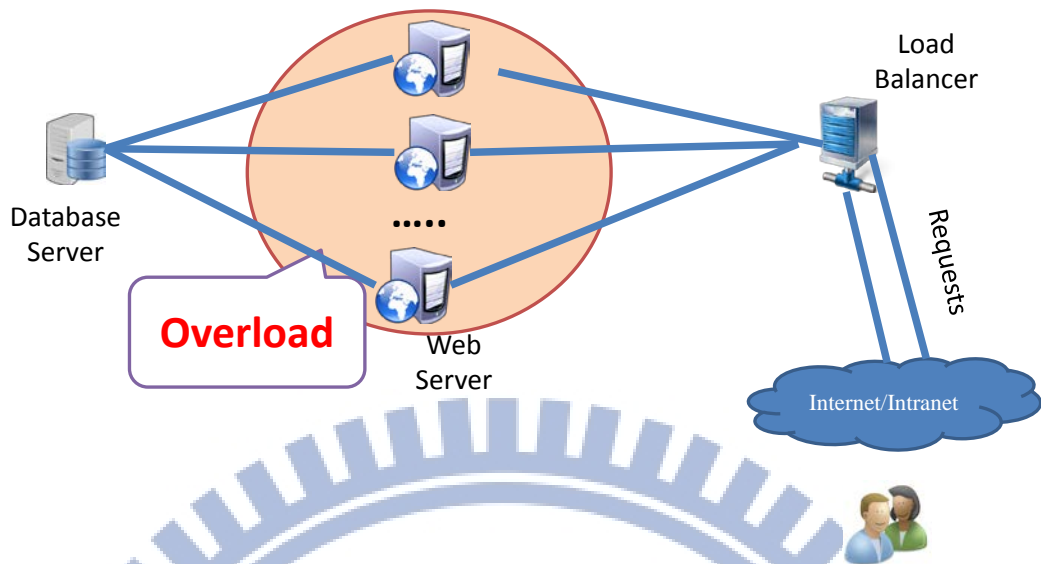


Figure 3.3: Web Servers overload

When capacity bandwidth of Load Balancer is large enough, it can handle all data transferred from Web Servers. The reason for website overload maybe is Web Servers are overloaded. Web Server has some kind of overloads like CPU overload, memory overload, bandwidth overload, etc.

3.1.3 Load Balancer Overload

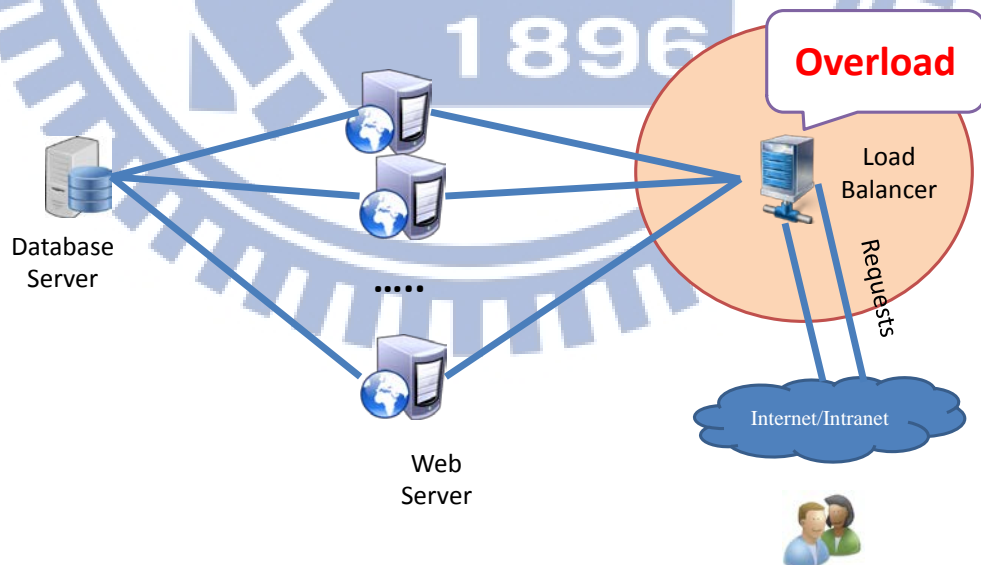


Figure 3.4: Load Balancer overload

The principle of Load Balance's operation is that every request is sent to Load Balancer, and then Load Balancer passes the requests to one of the Web Servers. After that, Web Server queries data from Database Server and prepares response content, and then the content go through Load Balancer before it is sent to end user. Obviously, data from every Web Server is transferred over Load Balancer, so Load Balance's bandwidth rate equals to Web Servers' entire bandwidth rate. Therefore, if users want their websites to work well even at times of a vast simultaneous user access, a Load Balance which has a large enough capacity bandwidth is needed. Otherwise, their website will be easily overloaded and cannot be expanded.

3.2 Website Overload Solutions

The solution depends on how adjustable a system is. If the system still can expand, several servers can be added into our original system. Otherwise, we can think about migrating our original system to one of cloud platforms such as Amazon Web Services (AWS).

3.2.1 Expanded the Original System

a) Database Server Overload

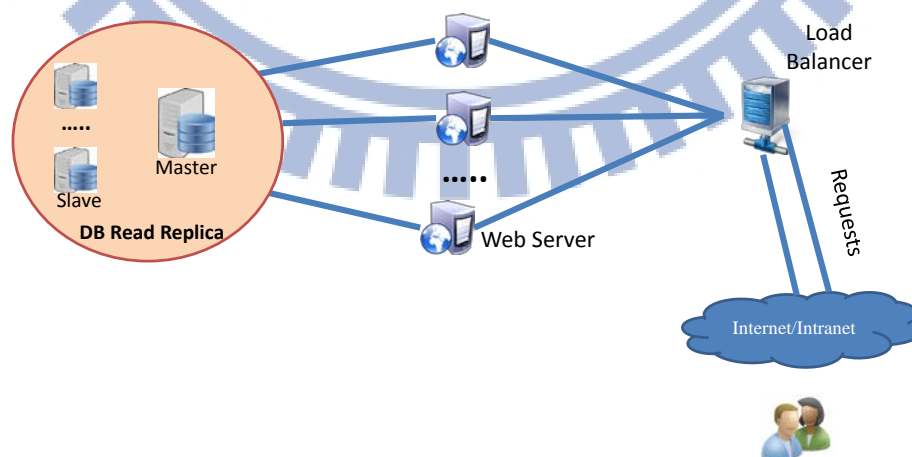


Figure 3.5: Database Server overload solution

The solution to this situation is using DB Read Replica technique, database replication [7][8][9]. It seems like a master – slave relationship between the original and the copies of database. The master can read/write database, but the slaves only can solely read

Every time the master writes new or updates several rows of database, it will also do synchronous database with the slaves

b) Web Servers Overload

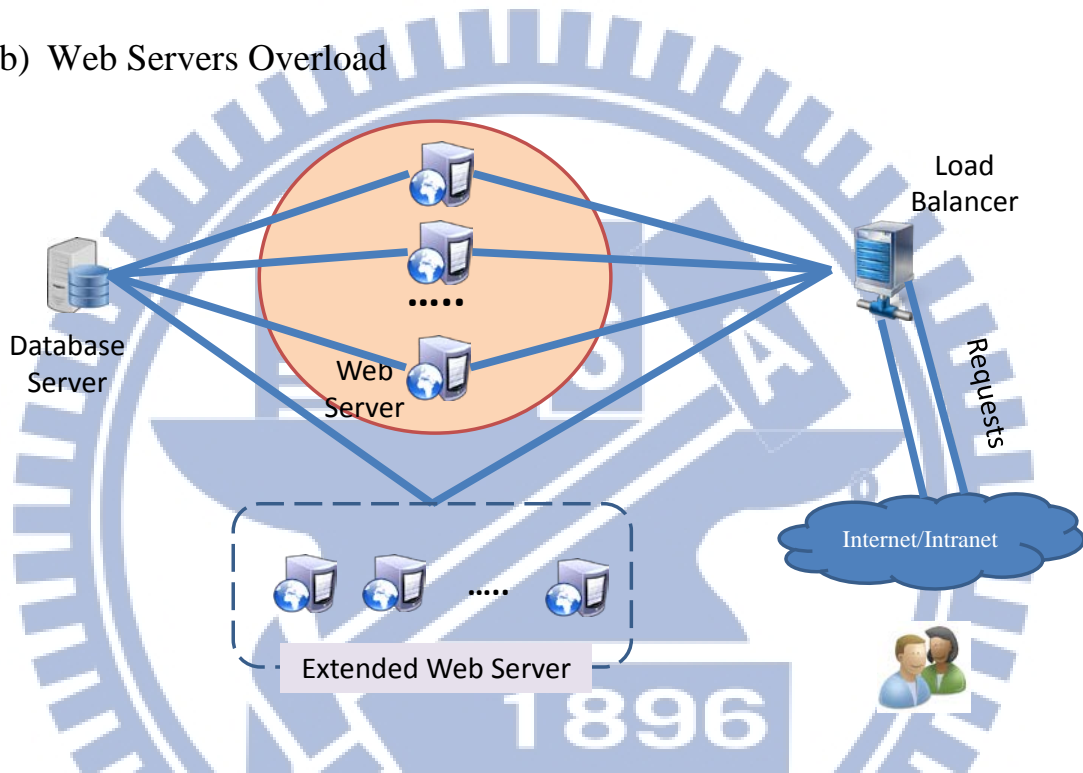


Figure 3.6: Web Servers overload solution

In this situation, we can upgrade the Web Servers or add some new Web Servers to the original system. However, machines should not be added in large number. Otherwise, the system may experience the previous problem (Database Server overload)

The numbers of Web Servers which can be added depend on the following aspects:

- Load Balance's capacity bandwidth: It is no sense if total Web Servers bandwidth, including added Web Servers, exceeds Load Balancer's capacity bandwidth.

Database server ability (CPU, memory, bandwidth, etc.): The Database server cannot serve too much Web Servers. If Database Server is strong enough and we just care about bandwidth, after doing some experiments, I have come to the conclusion that total Web Servers' bandwidth should approximately be 3 times as strong as Database server's bandwidth. That makes all of them work perfectly together.

c) Load Balancer Overload

To solve this situation, we can upgrade our Load Balancer, or add some new Load Balancers together with using Bin9 to route users' requests to one of Load Balancers.

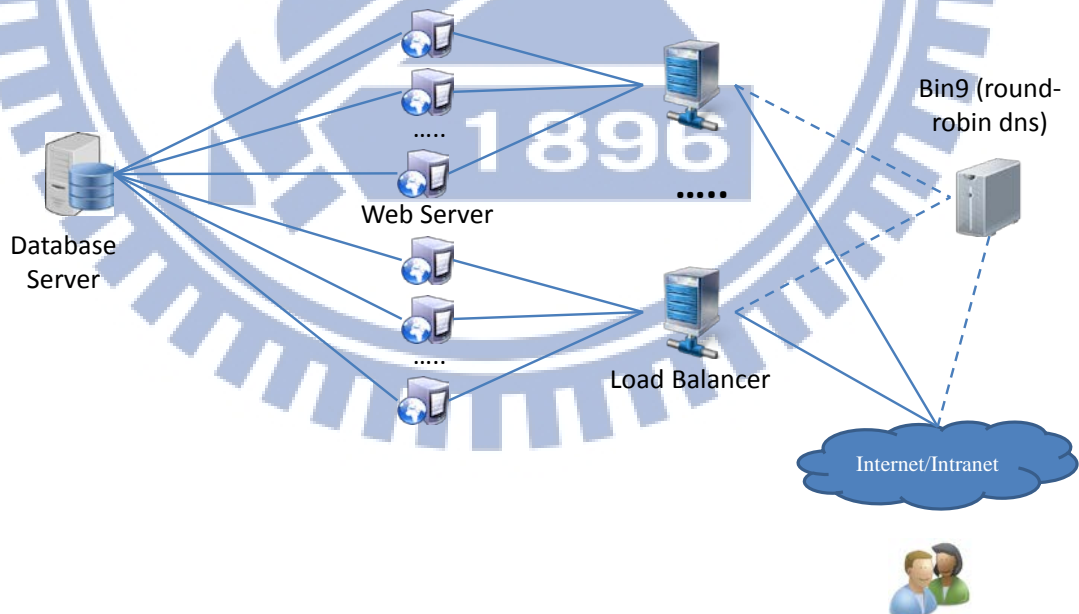


Figure 3.7: Load Balancer overload solution

For those aforesaid three solutions, before applying them, it's essential to analyze the strengths and the weaknesses:

- Strengths:
 - If we extend the original system, it will be used for long time
 - The whole system architecture does not change, so we do not need to worry about the system security.
- Weakness:
 - Extended system needs up-front cost.
 - Extended system also needs more power consumption and more management.

3.2.2 Applying AWS to the Original System

a) Extended original system with the AWS

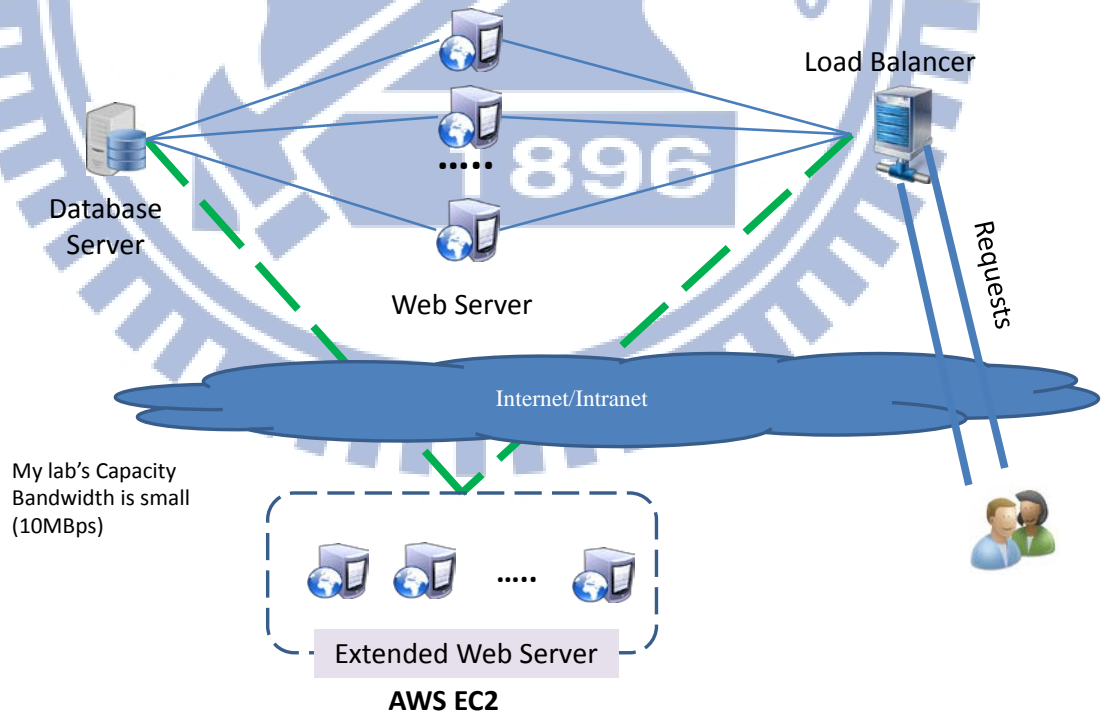


Figure 3.8: Extended original system with the AWS

The drawback of this solution is my lab's capacity bandwidth is small, so I could not see the effects. Furthermore, another reason is the data of database must be uploaded from local to AWS, but it is a slow speed link (uplink), I cannot do this kind of experiment.

b) Migration original to the AWS

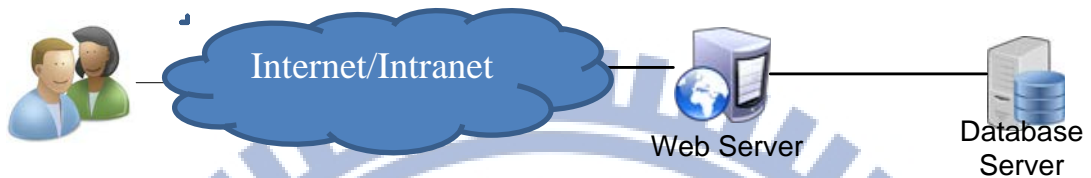


Figure 3.9: A normal website system

With the kind of system aforementioned, we could think about migrating our system to a cloud platform where we can scale our system in/out very quickly. I choose Amazon Cloud Services because it is cheap, supportive, and ease to develop. As a result, the real system, after we move our original system to cloud can be depicted in the figure bellow:

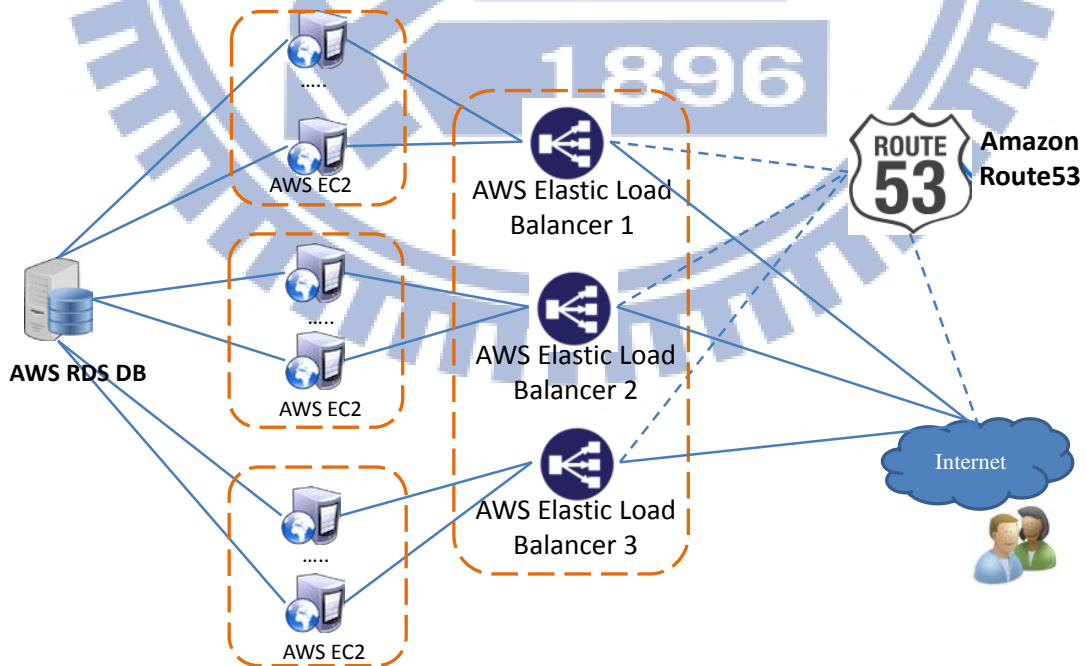


Figure 3.10: New system on AWS model

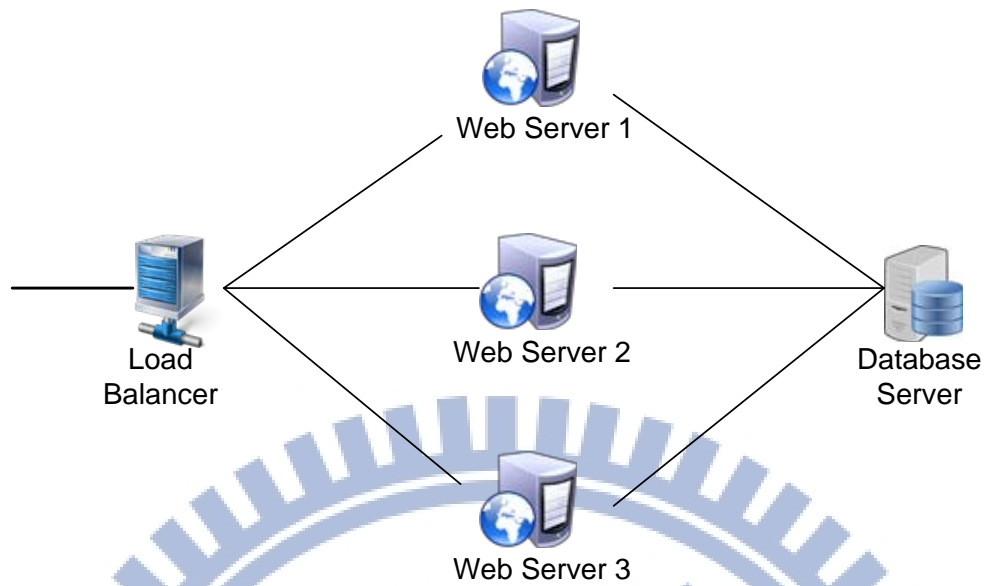


Figure 3.11: Equivalent of new system model

In this model, we need to use Amazon Route53 services that redirect user's requests to one of Amazon Elastic Load Balancers.

After migrating original system to AWS, if the capacity bandwidth of the link between user and Web Server is 1Gbps, we will have a new system's ability which is 3 times as efficient as the original system's ability. (The results of experiments are discussed later)

3.3 Doing Migrations

As a current trend, the concept of cloud computing has grown so popular, addressing multiple issues related to processing, calculating and sharing software, or developing enterprises. A lot of companies have thought about moving their services to cloud platform.

In the technologically business world, the use of on - premise software and off – premise software has been greatly assisting business activities. The former, which is often abbreviated as on-prem software, is installed and run on computers on the premises (in the building) of the person or an organization using the software, rather than at a remote place. The latter, off-

premises software, is commonly known as “software as service” or “computing in the cloud”

Until around 2005, the on-premises approach to deploying data was the most common. However, later, software running at a remote location became widely available and adopted. The use of new, alternative deployment method removed the need for the user to install any software on premises and brought about further benefits. Running software remotely can result in considerable cost savings because of reduced staffing, maintenance, power consumption, etc.

In other words, cloud computing has made it much easier to deploy data. With cloud, businesses can reduce up-front costs, management costs, pay for what they use, scale as they want, etc.



Figure 3.12: Migration from On-Premise to Off-Premise

To move a website from On-Premise to Off-Premise, we need to move Database and Web Servers.

3.3.1 Database Migration

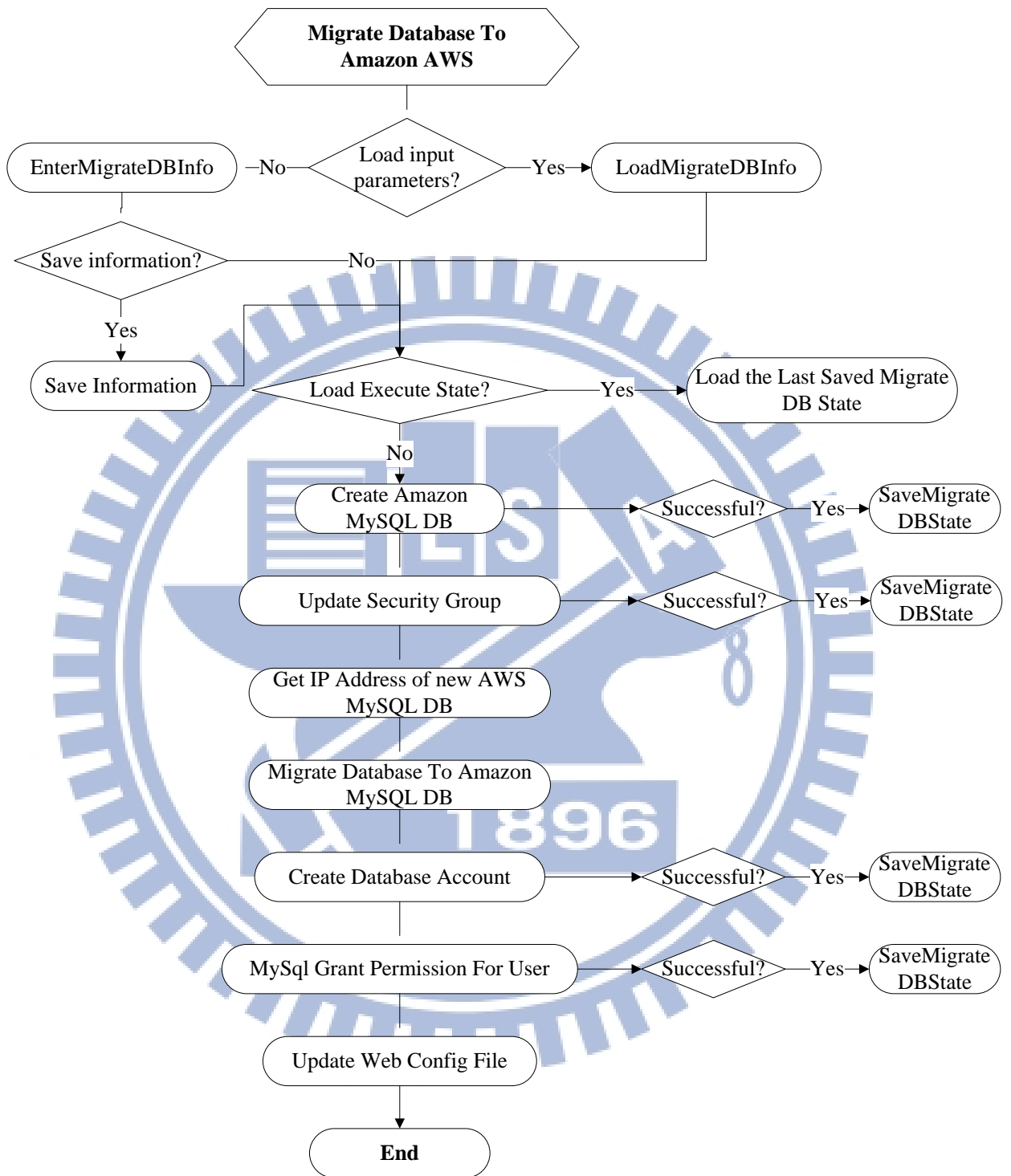


Figure 3.13: modeling database migration

* Simplified code file looks at appendix 2

The figure 3.7 shows the structure of doing database migration independently. When database migration starts, it asks whether the user wants to load the last saved input information or not. In situation 1, if the answer is yes, it will load the last saved information; otherwise, it will let us enter some requirement information for program running. After it gets all the requirement information, it will ask if the user wants to load the last saved program state or not. If the answer is yes, it will go to the last saved state of the program. Why is it necessary to save the state of the program? The answer is sometimes the program is crashed while it is running, so we do not want to let the program run at the beginning. We want the program to run from the last failed position. In contrast, in situation 2, if the answer is no, it will start to create a MySQL database on Amazon Relational Database Services (RDS). If this procedure is finished successfully, it will save this state and continue with the next step. The next step is to update security group of RDS which stores the list of the IPs with allowed access to database. Afterward, the program will get an IP address for the new database server and transfer database onto it. The program will accordingly create a new account and grand permission for that account to enable us to use the application. The last step of the program is to update web configuration file due to the change of database server IP address.

3.3.2 Web server Migration

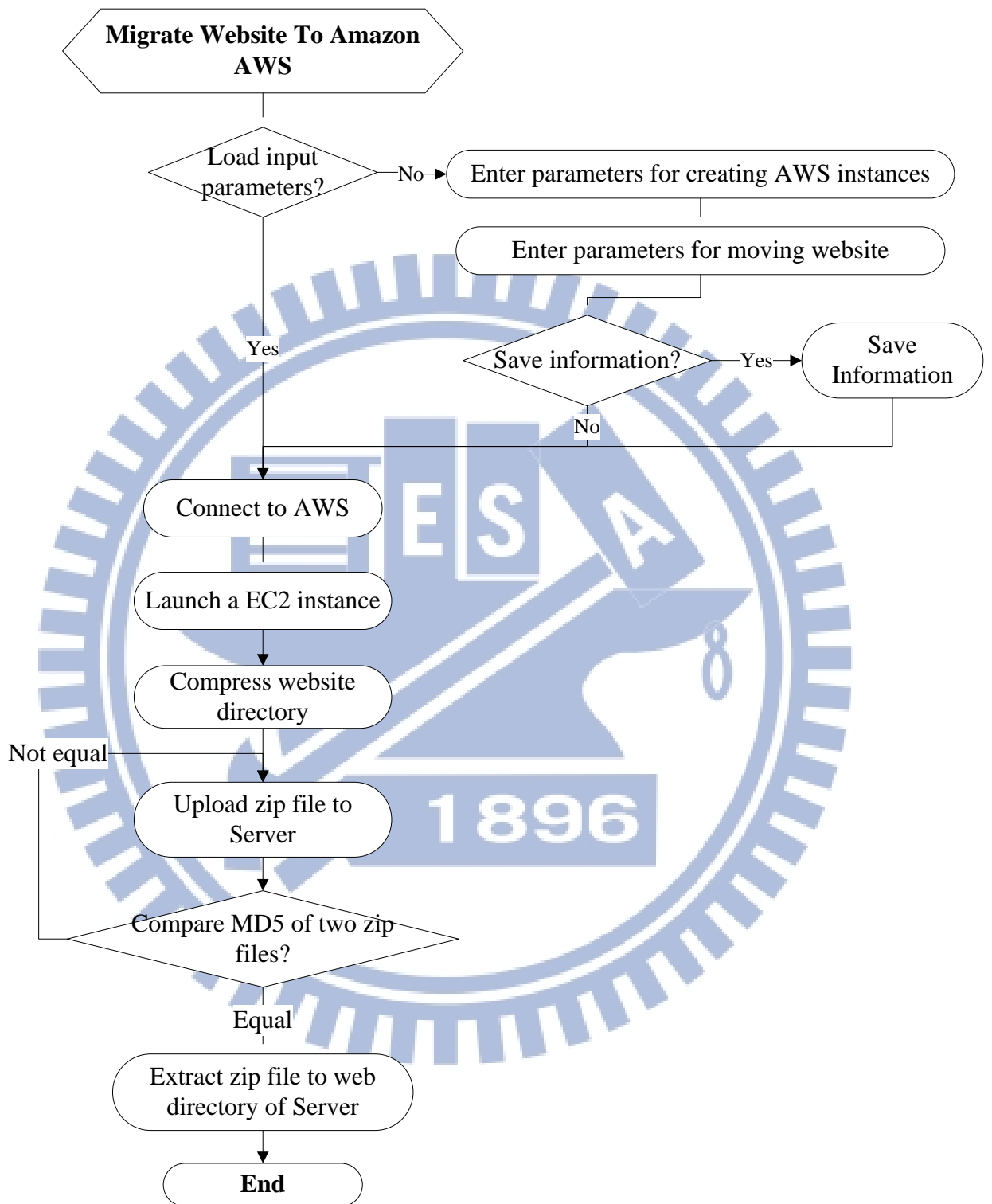


Figure 3.14: Modeling web server migration

* Simplified code file looks at appendix 3

The figure 3.8 shows the process of migrating web server independently. First of all, after started, the program asks the user if he/she wants to load requirement information. If the answer is no, it will call the input functions and let him or her enter requirement information. After that the program will ask another question which is whether the user wants to save the information or not. The user makes the decision and the program will proceed to the next step. In contrast, if the answer is no, the program will advance directly. In the next stage, the program will connect to AWS, launch an EC2 instance of AWS, and wait for the new instance to run until the instance's status is available. Afterward, the program will call compress website directory function – the function to make a zipped file including our whole website directory, and insert the new database configuration file into the zip file. This zip file later is uploaded to a new instance via SFTP (SSH File Transfer Protocol) function, a function of paramiko [14] that transfers files through SSHClient of paramiko between local machine and remote machine. At this point, the program will compare the MD5 of two zip files. If it is perfectly matched, the last step can be carried out. If not, it will try to upload the zip file again. In other words, the program will go to the last step when the MD5 of local zip file and remote zip file are matched and the last step will extract the zip file of new EC2 instance to the web directory of the instance. In the end, the new EC2 instance becomes a Web Server.

Actually, this function may be used just once. Why is that? This is because when we do the auto scaling – terminate or launch several instances – there are two ways to make an EC2 instance become a Web Server. In the first way, we create an Amazon Machine Image (AMI) that includes all of our requirement software and web directory which stores our website. As a result, we just have to update database configuration file at the time a new EC2 instance is created. In the second way, we compress our web directory and

upload it to Amazon Simple Storage Services (Amazon S3). When a new EC2 instance is created, it will download the compressed file from Amazon S3, extract the file to web directory, and then update database configuration file.

3.4 Security problems



Figure 3.15: AWS EC2 Key Pair

If we want to access one of EC2 instances, we need to show the Key Pair value. Every EC2 instance has one Key Pair.

Viewing: EC2 Security Groups 1 to 3 of 3 Items

	Group ID	Name	VPC ID	Description
<input checked="" type="checkbox"/>	sg-893432db	quick-start-1		quick-start-1
<input type="checkbox"/>	sg-d0fcc682	default		default group
<input type="checkbox"/>	sg-d63f2d84	research		limited using

1 Security Group selected

Security Group: quick-start-1

Details **Inbound**

Create a new rule: Custom TCP rule

Port range:

(e.g., 80 or 49152-65535)

Source:

(e.g., 192.168.2.0/24, sg-47ad482e, or 1234567890/default)

TCP Port (Service)	Source	Action
22 (SSH)	0.0.0.0/0	Delete
80 (HTTP)	0.0.0.0/0	Delete

Figure 3.16: AWS EC2 Security Groups

EC2 Security Groups acts as a firewall that lets us control our ports. Every EC2 instance belongs to one security group.

Look at the picture above, we just open two ports, port 22 for Linux remote control and port 80 for http connection.

Every connection from outside to other ports number is denied.

▼ Security Group Details

Connection Type	Details	Status	Actions
CIDR/IP	CIDR/IP: 140.113.128.159/32	authorized	<button>Remove</button>
CIDR/IP	CIDR/IP: 175.41.188.60/32	authorized	<button>Remove</button>
CIDR/IP	CIDR/IP: 192.168.0.110/32	authorized	<button>Remove</button>
CIDR/IP	CIDR/IP: 140.113.128.212/32	authorized	<button>Remove</button>
CIDR/IP	CIDR/IP: 192.168.0.109/32	authorized	<button>Remove</button>
CIDR/IP	CIDR/IP: 46.137.235.208/32	authorized	<button>Remove</button>
<input type="text" value="CIDR/IP"/>	CIDR: <input type="text"/> Our best estimate for the CIDR of your current machine is 140.113.128.159/32. However, if your machine is behind a proxy/firewall, this estimate may be inaccurate and you may need to contact your network administrator.		<button>Add</button>

[Refresh Security Groups](#)

Figure 3.17: DB Security Groups

DB Security Groups is for every Database server. It is also a firewall where we can set which IP address or range of IP address that can access our database.

Any device IP address which does not belong to the list of DB security group access to our database will be denied.

Your Security Credentials

Use this page to manage the credentials for your AWS account. To manage credentials for AWS Identity and Access Management (IAM) users, use the [IAM Console](#). To learn more about the types of AWS credentials and how they're used, see [AWS Security Credentials](#) in AWS General Reference.

Password

Multi-Factor Authentication (MFA)

Access Keys

Note: You can have a maximum of two access keys (active or inactive) at a time.

Created	Deleted	Access Key ID	Status	Actions
May 21st 2013		AKIAIWIYCN3POSJ2DCWA	Active	Make Inactive Delete
Jan 19th 2013	May 21st 2013	AKIAIXK2YLV06JAXOSMA	Deleted	
Sep 21st 2012	Mar 9th 2013	AKIAJOZYOH5QY3MJUCMA	Deleted	

Figure 3.18: AWS security credentials

For programmers who want to build an application interactive with AWS, they have to own one of security credentials.

Chapter 4:

Implementation

4.1 Experiments

For simulation, I have created a simple webpage that shows a list of course names. The table List Course has 43 rows. The webpage looks like a student who is making a query to Course Selection System. At the time the webpage is shown, it will insert a row into another table to mark one access is made successfully. After that we can count how many number of end users accessing the website at the same time.

交通大學 NCTU National Chiao Tung University
網·際·網·路·選·課·系·統

No: 0056137
Name: 黎高昆
Department: 資科工碩
Year: 102 學年度 第1 學期

Department: 資科工碩
No: 0056137
Name: 黎高昆
Year: 102 學年度 第1 學期

State	Course no	Course Name	Size limits	Registered numbers	Class Time/Room	credit	hours	Lecturers	Type
--	IOC5010	Independent Study	60	45	6AEC115	3	3	資科工所	選修
--	IOC5029	Algorithms	60	45	6AEC115	3	3	資科工所	選修
--	IOC5040	Artificial Intelligence	60	45	6AEC115	3	3	資科工所	選修
--	IOC5042	Evolutionary Computation	60	45	6AEC115	3	3	資科工所	選修
--	IOC5057	Operating System	60	45	6AEC115	3	3	資科工所	選修
--	IOC5076	Graph Theory	60	45	6AEC115	3	3	資科工所	選修
--	IOC5081	Data Mining	60	45	6AEC115	3	3	資科工所	選修
--	IOC5087	Secure Programming	60	45	6AEC115	3	3	資科工所	選修
--	IOC5090	Virtual Biomedical Instrumentation	60	45	6AEC115	3	3	資科工所	選修
--	IOC5091	Computer Architecture	60	45	6AEC115	3	3	資科工所	選修
--	IOC5100	Formal Languages and Theory of Computation	60	45	6AEC115	3	3	資科工所	選修

Figure 4.1: Simulation webpage

For running code file, we need to prepare our environment. We need to install some following tools:

- Boto [12]
- Amazon Relational Database Service Command Line [13]

- Paramiko [14]

4.1.1 Choice software for simulation

We have a lot of software which could help us to generate vast of users to simulate a real system with many users at the same time. They include httpert, curl-loader, loadUI, jmeter, etc. [16] But, the easiest, the most effective software is Jmeter - a java program. Jmeter can run on both Windows platform and Linux platform. And it is an application can be run with both command line and GUI.

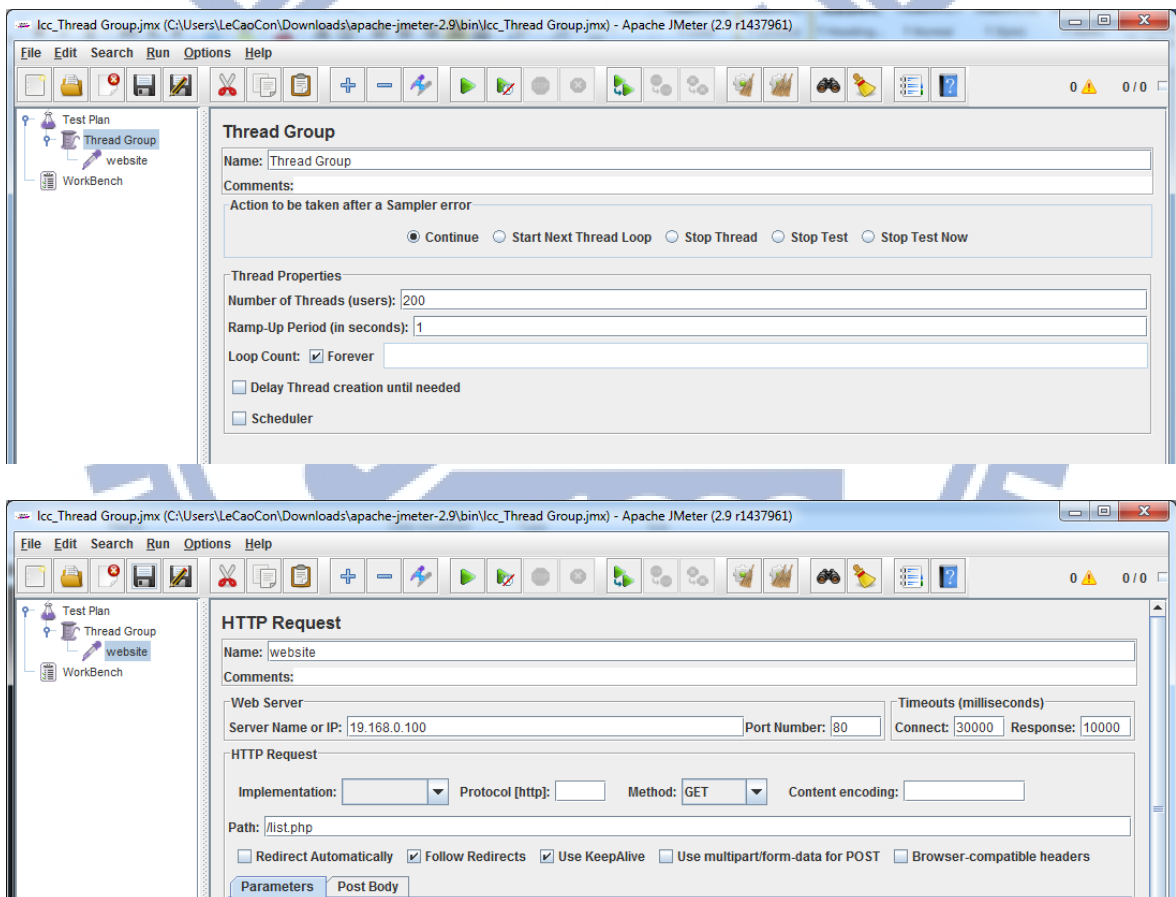


Figure 4.2: Jmeter GUI

If we want to run the application from command line, we need to prepare a jxm file. To create jxm file, we have two ways. First, we create a project from GUI, and then set all appropriate parameter values. After that

we will save the project into a new jxm file. Now we can use this file for running the application from command line. Second, we can use a jxm format file and insert some values into appropriate position such as host-ip, path, etc.

The command line format looks like:

```
java -jar ApacheJMeter.jar -n -t [jxm_filename]
```

4.1.2 Test link speed 100Mbps

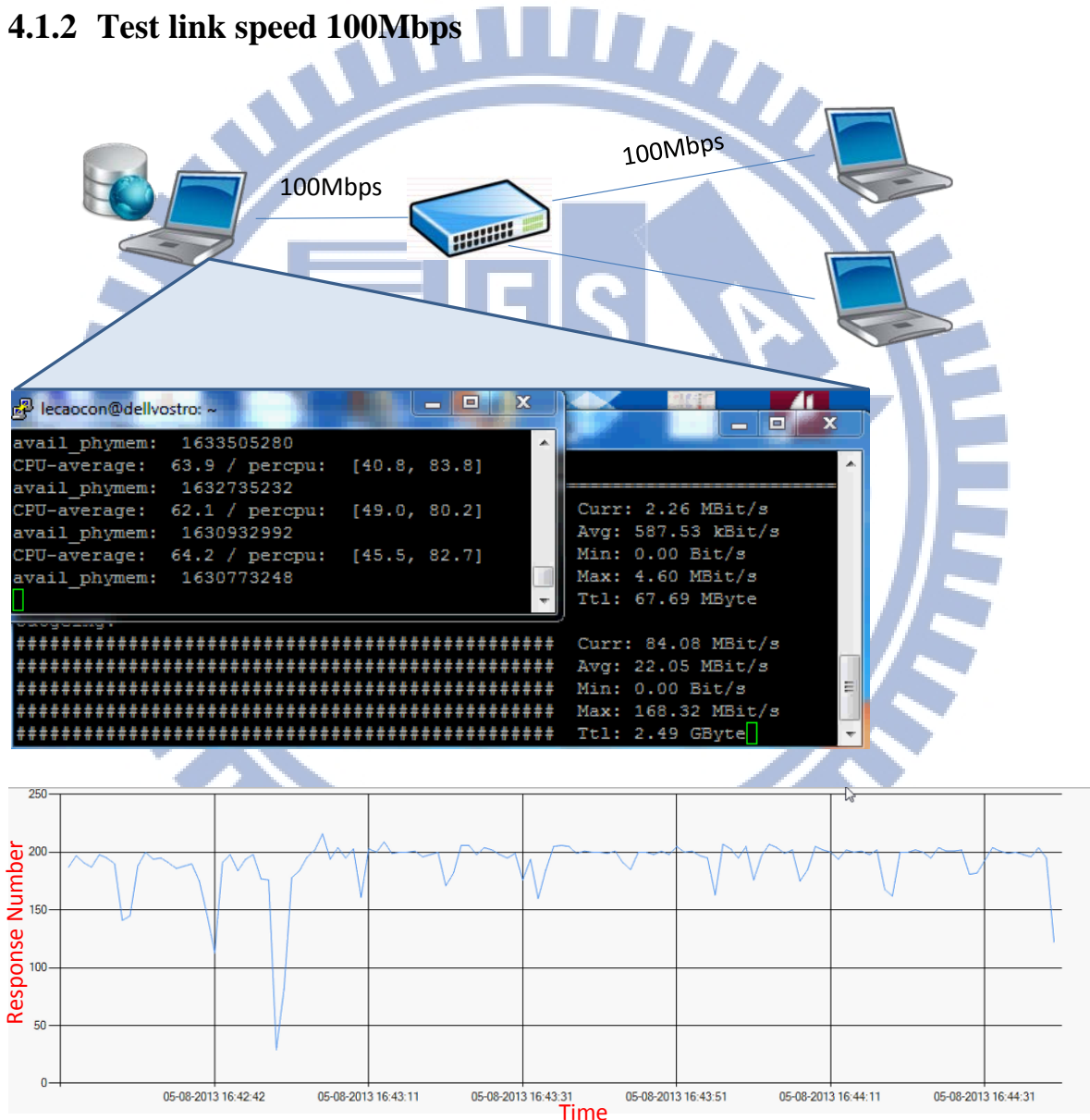


Figure 4.3: Simulation 100Mbps link

With the link connecting between end users and web server running at 100Mbps, we can see that the outgoing of system is approximate 80Mbps, and the number of user -access at the same time- is about 200 users.

4.1.3 Test link speed 1Gbps

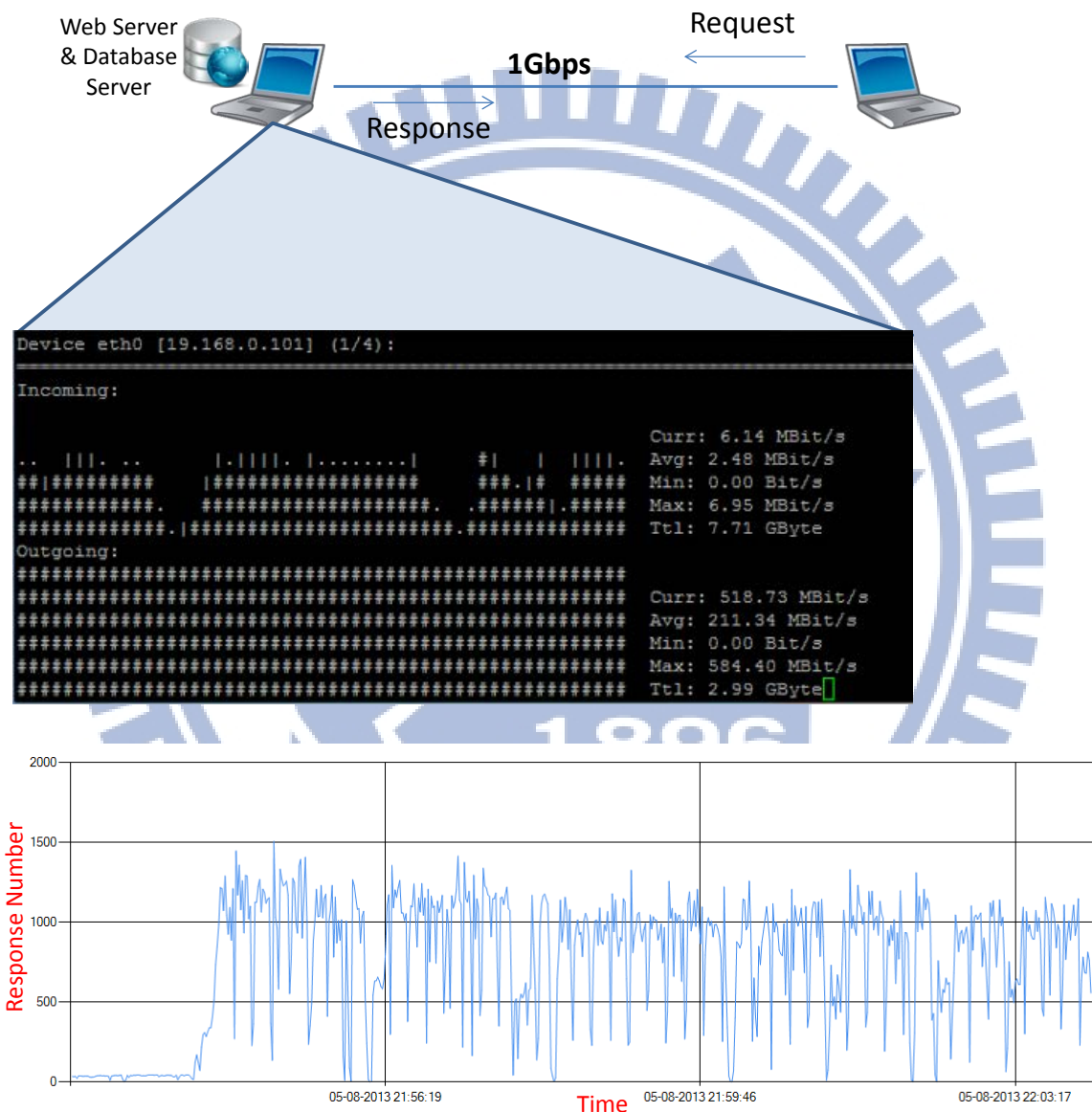


Figure 4.4: Simulation 1Gbps link

With this kind of link between end users and web server, we can see that the outgoing of system is approximate 500Mbps, and the number of user -access at the same time- is about 1000 users.

4.1.4 Test with Amazon Web Services

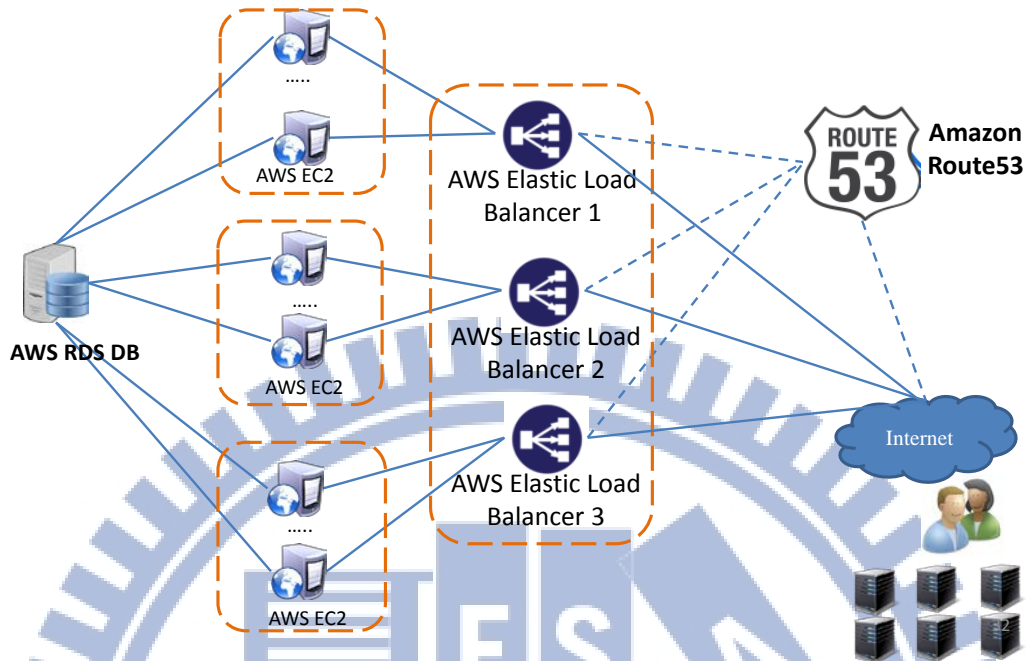


Figure 4.5: AWS experiment architecture

In these experiments, I create some virtual machines at another cloud platform as these machines are supposed to help me to generate big amount of requests.

a) 02 AWS EC2 instances

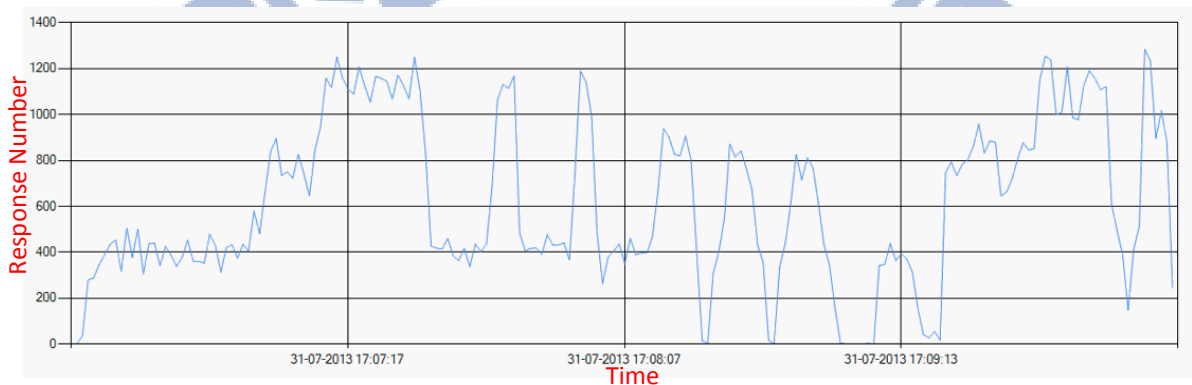


Figure 4.6: 02 AWS EC2 m1.large instances – 02 Elastic Load Balancers

When we run 02 EC2 instances, the number of user access at the same time is about 1100 per second. However, as illustrated in the diagram,

sometimes the number of user access decreases because the CPU of one or both of EC2 instances is overloaded. Therefore, these EC2 instances are marked unhealthy and ELB does not route traffic to these instances.

b) 04 AWS EC2 instances

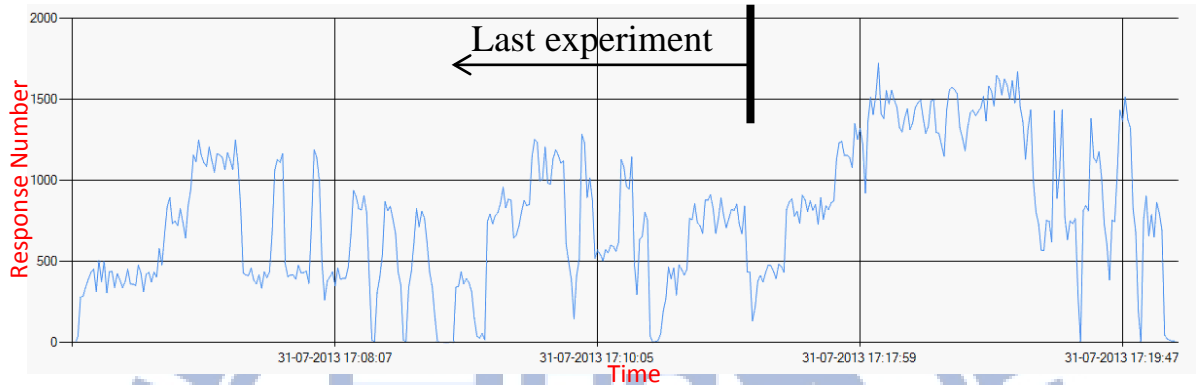


Figure 4.7: 04 AWS EC2 m1.large instances – 02 Elastic Load Balancers

With this kind of model, the number of access can reach 1500 per second. Nevertheless, in some cases, one of the EC2 instances is overloaded, so the number of access may fall.

c) 06 AWS EC2 instances

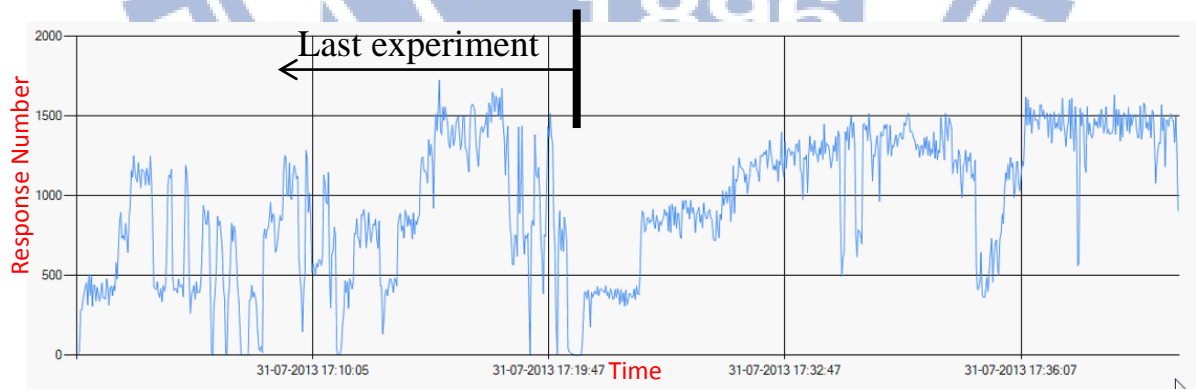


Figure 4.8: 06 AWS EC2 m1.large instances – 02 Elastic Load Balancers

With this kind of model, the number of access can reach 1500 per second. Nevertheless, in some cases, one of the EC2 instances is overloaded, so the number of access may fall.

d) 06 AWS EC2 instances with 03 Elastic Load Balancer

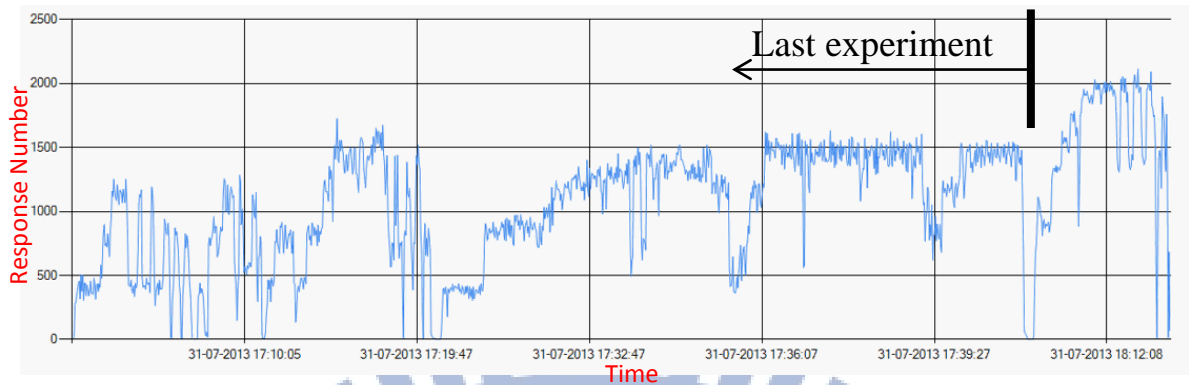


Figure 4.9: 06 AWS EC2 m1.large instances – 03 Elastic Load Balancers

In this experiment, the number of access can reach 2000 per second, but the drawback is that the system is not really stable.

e) 12 AWS EC2 instances

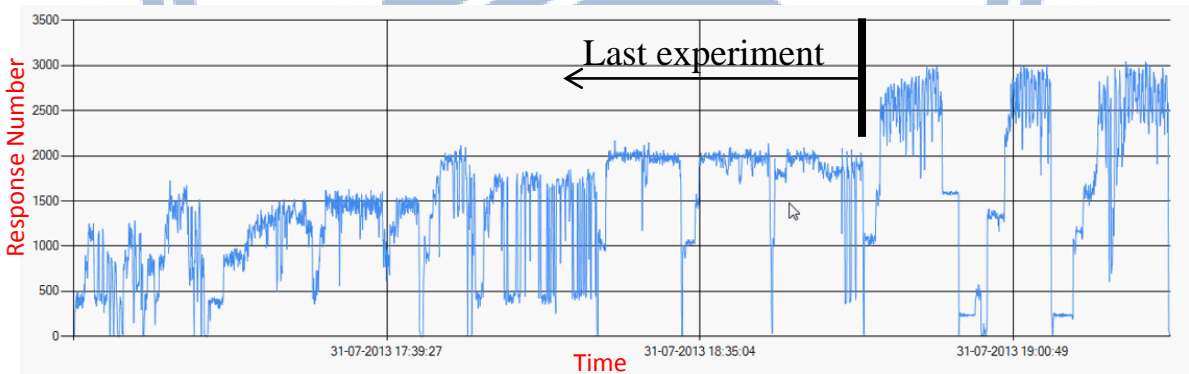


Figure 4.10: 12 AWS EC2 m1.large instances – 03 Elastic Load Balancers

In this experiment, the number of access almost reaches 3000 per second while the whole system remains quite stable.

f) 18 AWS EC2 instances

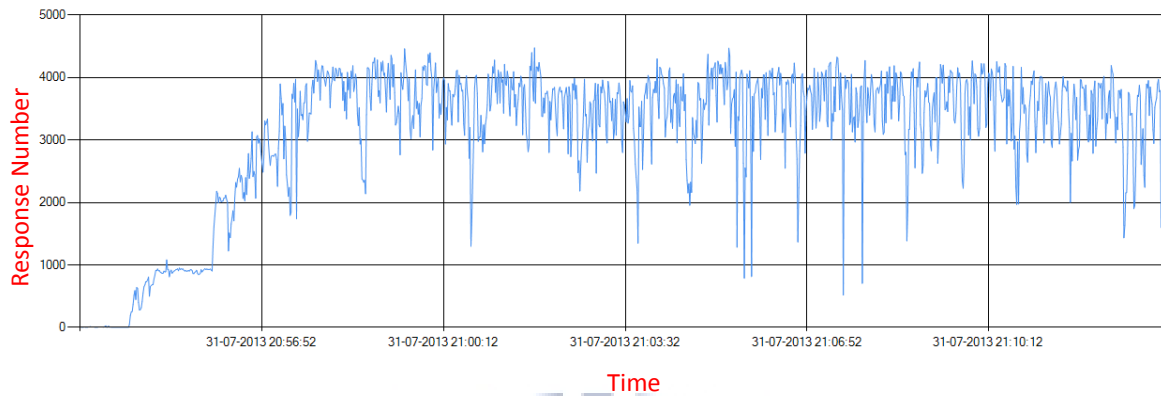


Figure 4.11: Running 18 AWS EC2 large instances and RDB db.m2.2xlarge

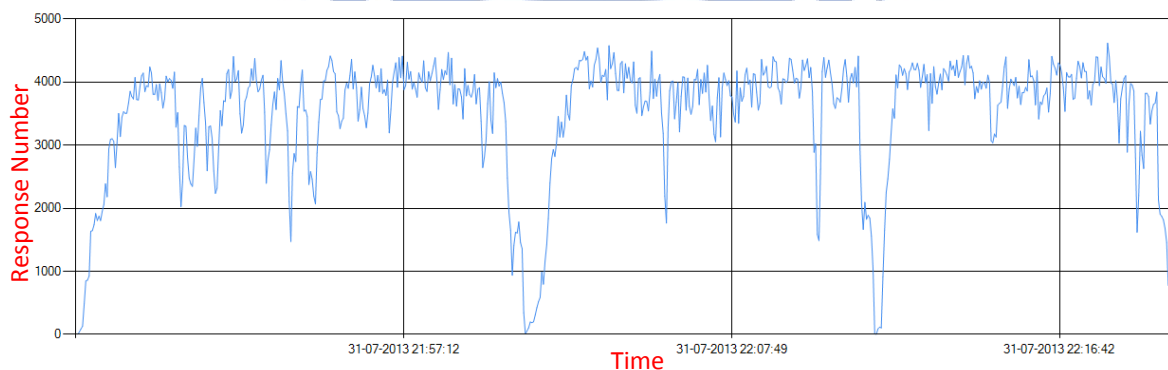


Figure 4.12: Running 18 AWS EC2 large instances and RDB db.m2.4xlarge

When we migrate our system to Amazon Web Services, our new system can respond up to 4000 user's requests at the same time. The new system is 3 to 4 times as efficient as our original system.

And the new system running with RDB db.m2.4xlarge gives a little bit better results than the others.

4.2 Monitoring System Scaling

4.2.1 Requirements

- Replace Nameserver of our Domain Name by Nameserver of Amazon Route53

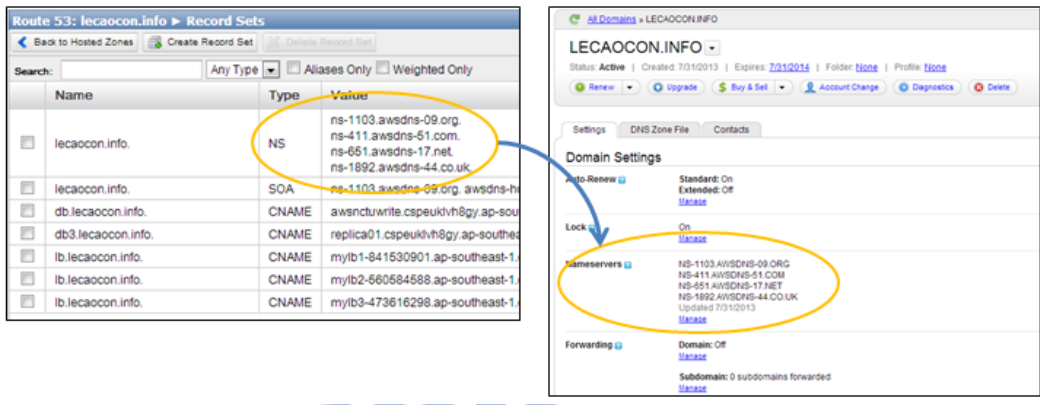


Figure 4.13: Replace Nameserver

- Prepare an Amazon Machine Image that includes all of the software we need such as Apache, php5, php5-mysql, OpenSSH-Server, etc.
- Our web application can be stored in AMI or uploaded to Amazon S3 or new EC2 instances (web server) have to send a message to a machine that will upload web directory to the new EC2 instances.

4.2.2 Structure

We set 1 condition that is CPU usage under 20 percent within 10 minutes for scale-in

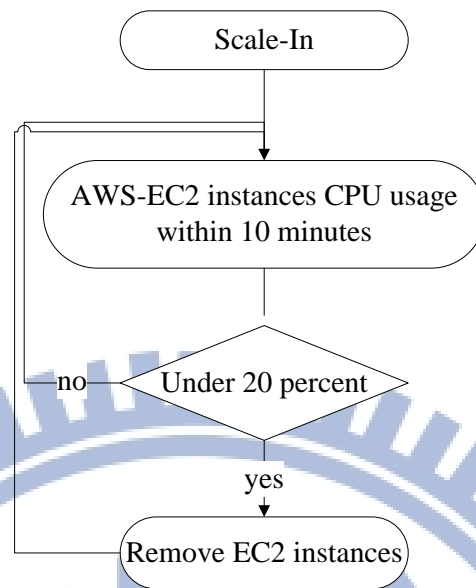


Figure 4.14: AWS scale-in

However, it's essential we set several conditions for scale-out operations. If the CPU usage is over 50 percent, we will launch 01 EC2 instance. If the CPU usage is increased to over 70 percent, we will launch 02 EC2 instances. As the system is subject to pressing demand, the system capacity needs to be enhanced more quickly.

We also set another condition for scale-out operation. The condition is when two or more EC2 instances are unhealthy instances, we will launch 01 EC2 instances because those instances will be back on track soon after they are detected as healthy instances.

With several policies, we can see that when our system have to response too much requests, some of instances maybe overload for few minutes and the whole system's CPU usage increase more quickly. At that time, we better launch several EC2 instances instead of launch one by one; it gives us a more effective system. [4]

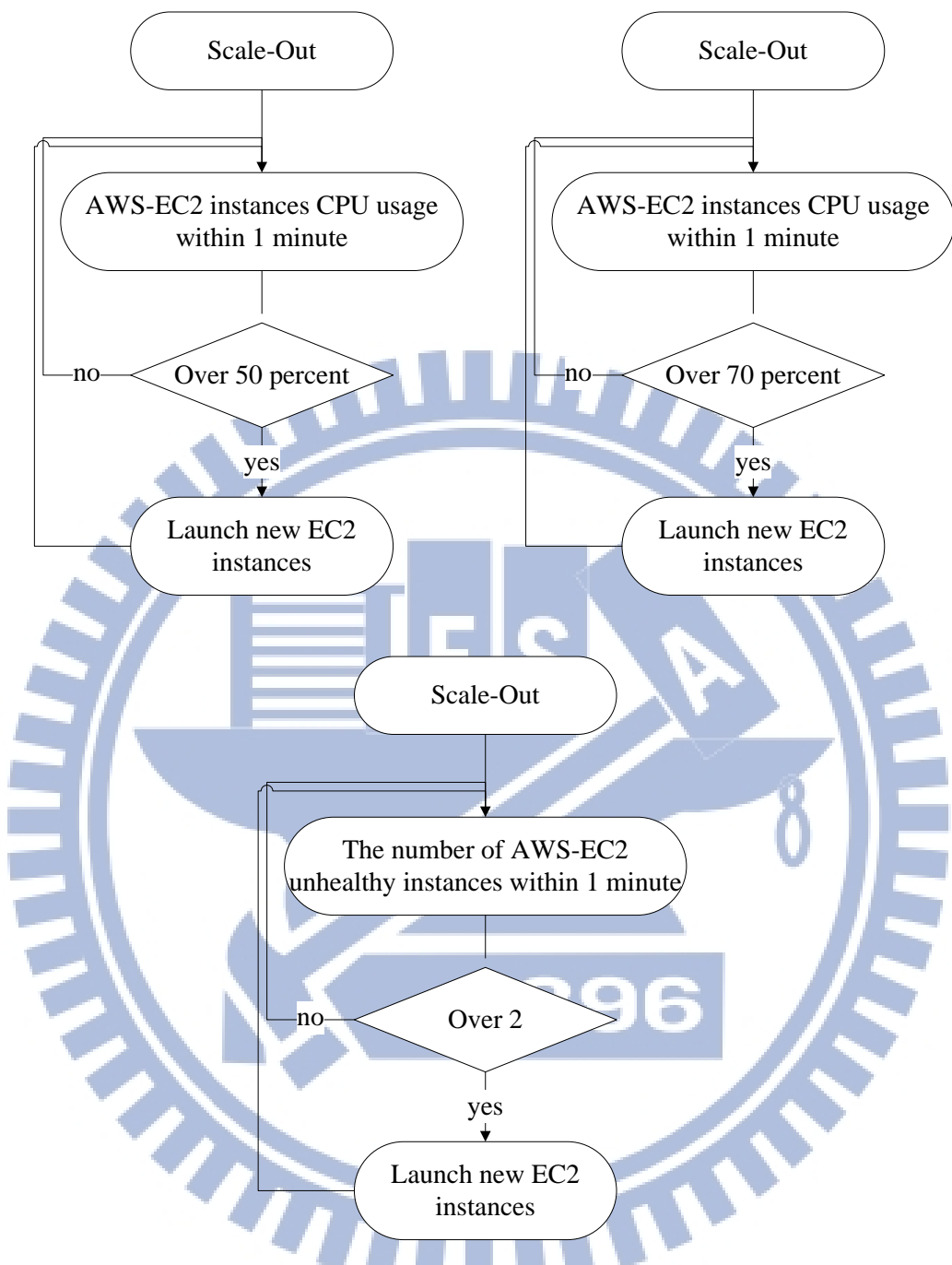
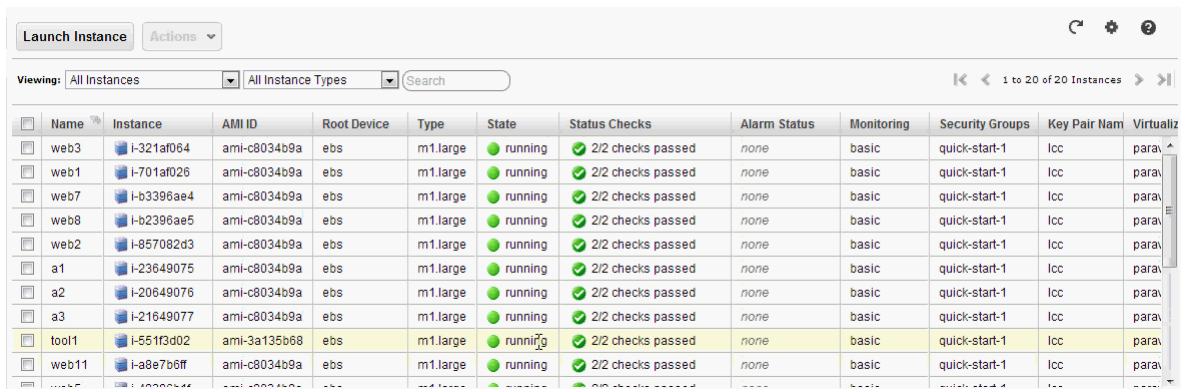


Figure 4.15: AWS scale-out

* Simplified code file looks at appendix 1

4.2.3 Result



Name	Instance	AMI ID	Root Device	Type	State	Status Checks	Alarm Status	Monitoring	Security Groups	Key Pair Name	Virtualization
web3	i-321af064	ami-c8034b9a	ebs	m1.large	running	2/2 checks passed	none	basic	quick-start-1	lcc	para
web1	i-701af026	ami-c8034b9a	ebs	m1.large	running	2/2 checks passed	none	basic	quick-start-1	lcc	para
web7	i-b3396ae4	ami-c8034b9a	ebs	m1.large	running	2/2 checks passed	none	basic	quick-start-1	lcc	para
web8	i-b2396ae5	ami-c8034b9a	ebs	m1.large	running	2/2 checks passed	none	basic	quick-start-1	lcc	para
web2	i-857082d3	ami-c8034b9a	ebs	m1.large	running	2/2 checks passed	none	basic	quick-start-1	lcc	para
a1	i-23649075	ami-c8034b9a	ebs	m1.large	running	2/2 checks passed	none	basic	quick-start-1	lcc	para
a2	i-20649076	ami-c8034b9a	ebs	m1.large	running	2/2 checks passed	none	basic	quick-start-1	lcc	para
a3	i-21649077	ami-c8034b9a	ebs	m1.large	running	2/2 checks passed	none	basic	quick-start-1	lcc	para
tool1	i-551f3d02	ami-3a135b68	ebs	m1.large	running	2/2 checks passed	none	basic	quick-start-1	lcc	para
web11	i-a8e7b6ff	ami-c8034b9a	ebs	m1.large	running	2/2 checks passed	none	basic	quick-start-1	lcc	para

Figure 4.16: AWS scaling

By default, our account is limited to a maximum of 20 instances per EC2 region. If we want to run more than 20 instances, we have to get approval directly from Amazon. Actually, RDS Database bandwidth is limited, so we do not need to launch too much EC2 instances. We will create three Elastic Load Balancers that make an optimized system for us; the number of EC2 instances of every Load Balancer is scaled from 1 to 6 instances depends on the system work load.

4.3 Cost of using Amazon Web Services for Course Selection System

EC2 and RDS					
	Cost per Hour	Number of instances	Hours per Day	Days	Cost
EC2 m1.large	\$0.32	3	15	5	\$72.00
EC2 m1.large	\$0.32	15	15	2	\$144.00
RDS m2.2xlarge	\$1.28	1	15	5	\$96.00
					\$312.00
Other services					
	Cost per GB	Using (GB)			Cost
Elastic Load Balancing \$0.008 per GB Data Processed	\$0.008	1000			\$8.00
AWS Data Transfer \$0.190 per GB - first 10 TB / month data transfer out	\$0.190	1000			\$190.00
Other services					\$90.00
					\$288.00
Total					\$600.00

The content of the table above is estimated for Selection Course System.

A Course Selection System is often opened from 9:00 AM to 12:00 AM every weekday of two weeks. Normally, in the first week, the system still works well with original system. This means we do not need to change our original system.

In the second week, actually, the system needs to be more powerful at the end of the week. Assuming that the last two days is the time of vast number of students using the system. Therefore, the first three days of the second week, we just launch three AWS EC2s for running web server. After that, for the last two days of the second week, the system maybe

automatically scaled out to its highest level that includes 18 EC2 instances running giving us a really powerful system..

If we apply this model of cloud computing for our Course Selection System, the estimated cost is \$600 per semester.

Some references of server pricing:

The image shows two screenshots of the KUSOBOX shopping website, which is a PChome affiliate. Both screenshots display a 'Hot! 熱賣商品' (Hot! Best Selling Products) section with various server models and their prices.

Top Screenshot (ASUS Servers):

- 華碩 ASUS RS300-E 8-PS4 熱抽插式伺服器 [Intel Xeon E3-1230 v3 3.3G / 4G 記憶體 / DVD-RW]**: \$35,720
- 華碩 ASUS RS100-E 8-P12 插架式1U伺服器 [Intel Xeon E3-1230 v3 3.3G / 4G 記憶體 / DVD-RW]**: \$28,250
- 華碩 ASUS RS300-E 7 / RS4 單CPU Hot-Swap熱抽1U插架式伺服器 [Intel Xeon E3 1230 v2 3.2G / 4G DDR3 / RAID 0,1,10]**: \$44,690
- 華碩 RS300-E7/PS4 1U 熱抽插式伺服器 [Intel Xeon E3-1230 v2 3.2G / 4G DD R3 / 支援RAID-0,1,10]**: \$37,990
- 華碩 ASUS RS100-E 7 非熱抽插式伺服器 [Intel Xeon E3-1230 v2 3.2G / 4G 記憶體 / DVD-RW]**: \$34,890

Bottom Screenshot (IBM Servers):

- IBM X3200 M3 (7 328-C2V) Hot-Swap 熱抽SATA/SAS 伺服器 [Intel X3430 QC 2.4G / 1G x2 記憶體 / RAID-0,1]**: \$33,890
- IBM X3300 M4 (7 382-C2V) Hot-Swap 熱抽SATA/SAS 單立式伺服器 [Intel E5-2420 6C 1.9G / 4GBx1 / H1110(Raid-0,1)]**: \$57,150
- IBM X3100 M4 (2 582-IRA) 非熱抽(Simple-Swap)SATA 單CPU單立式伺服器 [Intel E3-1230v2 4C 3.3G / 2GB / 支援Raid-0/1/10]**: \$23,790
- 500GB硬碟x1 + IBM X3100 M4 (25 82-IRA) 非熱抽SATA 單CPU單立式伺服器 [Intel E3-1230 v2 4C 3.3G / 2GB / 支援Raid-0/1/10]**: \$28,000
- IBM X3200 M3 (7 328-42V) Hot-Swap 熱抽SATA/SAS 伺服器 [Intel X3440 QC 2.53G / 1G x 2記憶體 / RAID-0,1]**: \$49,000

Figure 4.17: References of server pricing

Chapter 5:

Conclusion and Future Works

As a matter of fact, some kinds of websites, such as NCTU's Course Selection System or the railway ticket selling system in my country, always meet traffic overload situation a certain time. After doing this research, I have reached the conclusion that the system overload may be improved by applying cloud computing technology. It is certain that it depends on how well a system performs. If we compare 1Gbps-link system and AWS new system, we can see that the power of AWS new system is approximately 3 times as big as that of the 1Gbps-link system. We should migrate the system before congestion and set reasonable conditions for scaling to make our system highly available and redundant.

With Amazon Web Services, Database server bandwidth is limited, so we cannot make a more powerful system. But AWS has provided Database Replicas function. Unfortunately, it does not seem to work very well.

I may consider changing testing method— insert one row into the database at every loading webpage time, creating the ratio of read/write is 50:50 while this ratio should be 66 percent reads [1] or 60:40 [2]

I believe that the system will be more powerful when we combine EC2 Auto Scaling and Database Replicas functions.

I will also do research migrate systems based on Windows platform.

REFERENCES

- [1] Microsoft TechNet “*Understanding Database and Log Performance Factors*”, April 2013
- [2] White paper of EMC Corporation, “*Deploying Oracle Database on EMC VNX Unified Storage*”, May 2011
- [3] Bianca Schroeder, Mor Harchol-Balter, “*Web servers under overload: How scheduling can help*”, May 2002
- [4] Marshall, P.; Tufo, H.; Keahey, K. “*Provisioning Policies for Elastic Computing Environments*”, Parallel and Distributed Processing Symposium Workshops & PhD Forum (IPDPSW), 2012 IEEE 26th International, On page(s): 1085 - 1094
- [5] Wikipedia - Cloud Computing
http://en.wikipedia.org/wiki/Cloud_computing
- [6] Peter Mell, Timothy Grance, “*The NIST Definition of Cloud Computing*”, September 2011
- [7] Database Replication
[http://en.wikipedia.org/wiki/Replication_\(computing\)](http://en.wikipedia.org/wiki/Replication_(computing))
- [8] Database Journal, “*Very Large Databases and High Availability Evaluating Replication Options*”, Nov 8, 2006
- [9] A. Sousa L. Soares A. Correia Jr. F. Moura R. Oliveira, “*Development and evaluation of database replication in ESCADA*”
<http://citeseerx.ist.psu.edu/viewdoc/similar?doi=10.1.1.160.3801&type=sc>
- [10] Webopedia – Cloud Computing

http://www.webopedia.com/TERM/C/cloud_computing.html

[11] Amazon Documentation

<http://aws.amazon.com/documentation/>

[12] AWS SDK for Python

<http://aws.amazon.com/sdkforpython/>

[13] Amazon Relational Database Service Command Line

<http://docs.aws.amazon.com/AmazonRDS/latest/CommandLineReference/StartCLI.html>

[14] Paramiko

<http://docs.paramiko.org/>

[15] Abhijit's World of .Net

<http://abhijitjana.net/2010/10/01/what-is-the-difference-between-web-farm-and-web-garden/>

[16] Performance test tools

<http://www.opensourcetesting.org/performance.php>

APPENDICES

Appendix 1: AWS Scaling

```
def AutoScaling(ELB):
    AutoScalingGroup_name='my-group-' + ELB
    scale_up_policy_name= 'scale-up-' + ELB
    scale_down_policy_name = 'scale-down-' + ELB
    scale_up_alarm_avarageCPU_name = 'scale-up-on-cpu-' + ELB
    scale_up_alarm_Unhealthy_name = 'scale-up-on-unhealthy-' + ELB
    scale_down_alarm_name = 'scale-down-on-cpu-' + ELB
    try:
        filename = '/home/lecaocon/masterproj/src/AWS_Userdata.py'
        f = open (filename, 'r')
        user_data= f.read()
        f.close()

        regions = autoscale.regions()
        region = regions[0]
        conn = AutoScaleConnection(aws_access_key_id=AWS_ACCESS_KEY_ID,
aws_secret_access_key=AWS_SECRET_ACCESS_KEY, region=region)
        print conn
        #autoscale = boto.ec2.autoscale.connect_to_region(region_name=HomeRegion)

        #Launch Config
        print 'Launch Config'
        lc = LaunchConfiguration(name='my-launch-config',
            image_id=image_id,
            key_name=key_name,
            security_groups=[security_groups],
            instance_type=instance_type,
            user_data=user_data)
        try:
            conn.create_launch_configuration(lc)
        except Exception,e:
            "#print e

        #Auto Scaling Group
        print 'Auto Scaling Group'
        ag = AutoScalingGroup(group_name=AutoScalingGroup_name, load_balancers=[ELB],
            availability_zones=['ap-southeast-1a', 'ap-southeast-1b'],
            desired_capacity=1,
            launch_config=lc, min_size=1, max_size=6,
            connection=conn)
        conn.create_auto_scaling_group(ag)

        #Scale policy
        print 'Scale policy'
        print 'scale up'
        scale_up_policy = ScalingPolicy(
            name=scale_up_policy_name, adjustment_type='ChangeInCapacity',
            as_name=AutoScalingGroup_name, scaling_adjustment=1, cooldown=180)
        conn.create_scaling_policy(scale_up_policy)

        print 'scale down'
```

```

scale_down_policy = ScalingPolicy(
    name=scale_down_policy_name, adjustment_type='ChangeInCapacity',
    as_name=AutoScalingGroup_name, scaling_adjustment=-1, cooldown=180)
conn.create_scaling_policy(scale_down_policy)

scale_up_policy = conn.get_all_policies(
    as_group=AutoScalingGroup_name, policy_names=[scale_up_policy_name])[0]
scale_down_policy = conn.get_all_policies(
    as_group=AutoScalingGroup_name, policy_names=[scale_down_policy_name])[0]

#Cloud Watch
print 'Cloud Watch'
regions= cloudwatch.regions()
region = regions[0]
cloudwatch_conn = CloudWatchConnection(aws_access_key_id=AWS_ACCESS_KEY_ID,
    aws_secret_access_key=AWS_SECRET_ACCESS_KEY,
    region=region)

#cloudwatch = boto.ec2.cloudwatch.connect_to_region(HomeRegion)
print 'dimensions'
alarm_dimensions = {"AutoScalingGroupName": '%s' % AutoScalingGroup_name}
print 'alarm up'
#print scale_up_policy.policy_arn
#CPU Average > 50 percent
scale_up_alarm = MetricAlarm(
    name=scale_up_alarm_avarageCPU_name + '-CPU50percent', namespace='AWS/EC2',
    metric='CPUUtilization', statistic='Average',
    comparison='>', threshold='50',
    period='60', evaluation_periods=1,
    alarm_actions=[scale_up_policy.policy_arn],
    dimensions=alarm_dimensions)
cloudwatch_conn.create_alarm(scale_up_alarm)

#CPU Average > 70 percent
scale_up_alarm = MetricAlarm(
    name=scale_up_alarm_avarageCPU_name+ '-CPU70percent', namespace='AWS/EC2',
    metric='CPUUtilization', statistic='Average',
    comparison='>', threshold='70',
    period='60', evaluation_periods=1,
    alarm_actions=[scale_up_policy.policy_arn],
    dimensions=alarm_dimensions)
cloudwatch_conn.create_alarm(scale_up_alarm)

#UnHealthy Instances
scale_up_alarm = MetricAlarm(
    name=scale_up_alarm_Unhealthy_name, namespace='AWS/EC2',
    metric='UnHealthyHostCount', statistic='Sum',
    comparison='>=', threshold='2',
    period='60', evaluation_periods=1,
    alarm_actions=[scale_up_policy.policy_arn],
    dimensions=alarm_dimensions)
cloudwatch_conn.create_alarm(scale_up_alarm)

print 'alarm down'
#CPU Average < 20 percent
scale_down_alarm = MetricAlarm(
    name=scale_down_alarm_name+ '-30percent', namespace='AWS/EC2',
    metric='CPUUtilization', statistic='Average',

```

```
comparison='<', threshold='20',  
period='60', evaluation_periods=10,  
alarm_actions=[scale_down_policy.policy_arn],  
dimensions=alarm_dimensions)  
cloudwatch_conn.create_alarm(scale_down_alarm)
```

```
except Exception, ex:  
    print ex
```



Appendix 2: Database migration

```
def MigrateDBToAWS():
    #migrate database from a database server to another database server
    #Local DB
    ListVars.RootDBServerUsername = 'username'
    ListVars.RootDBServerPassword = 'xxx'
    ListVars.database_name = 'databasename'
    ListVars.DBRootUsername = 'root'
    ListVars.DBRootPassword = 'xxx'
    ListVars.dbusername = 'webaccess'
    ListVars.dbpassword = 'xxx'
    ListVars.OriDB_ip = '192.168.0.100'
    #AWS DB
    ListVars.AWSDBMasterName = 'root'
    ListVars.AWSDBMasterPassword = 'xxx'
    ListVars.DBid = 'AwsNctu'
    ListVars.HomeRegion = 'ap-southeast-1'
    ListVars.AWS_ACCESS_KEY_ID = 'KEY_ID'
    ListVars.AWS_SECRET_ACCESS_KEY = 'xxx'
    db_instance_type='db.m2.2xlarge'
    temp = raw_input('Do you want to load the information that you had saved (Yes/No)?')
    if (temp.upper() == 'Y' or temp.upper() == 'YES'):
        LoadInfo = True
    else:
        LoadInfo = False

    if(LoadInfo == True):
        b = LoadMigrateDBInfo()
        if(b == False):
            return False
    else:
        # Enter some basic information
        EnterMigrateDBInfo()

    temp = raw_input('Do you want to load execute state that you had saved (Yes/No)?')
    if (temp.upper() == 'Y' or temp.upper() == 'YES'):
        LoadState = True
    else:
        LoadState = False

    if(LoadState == True):
        b = LoadMigrateDBState()
        if(b == False):
            return False

    # Check this function has done or not?
    if(b_CreateMySQLDB == False):
        # Create AWS MySQL DB
        b = CreateMySQLDB(region=ListVars.HomeRegion,
            allocated_storage=5,
            instance_class=db_instance_type,
            master_username=ListVars.AWSDBMasterName,
            master_password=ListVars.AWSDBMasterPassword,
            DBid=ListVars.DBid,
            db_name=ListVars.database_name,
            backup_retention_period=1)
        if(b == True):
```

```

# state 1
SaveMigrateDBState(CreateFile=True, Function='CreateMySQLDB')
else:
    return False

# Check this function has done or not?
if(b_MySqlSecurityGroupAddListServers == False):
    #Allow list of servers can access to DB Server
    print 'MySqlSecurityGroupAddListServers'
    b = MySqlSecurityGroupAddListServers(filename='hosts')
    if(b == True):
        # state 2
        SaveMigrateDBState(CreateFile=False, Function='MySqlSecurityGroupAddListServers')
    else:
        return False

# Get AWS DB IP Address
print 'Waiting for create DB Instance...'
while (True):
    # include get DB instance EndPoint
    result = GetAWSDBInstanceStatus(DBid=ListVars.DBid,
                                     region=ListVars.HomeRegion,
                                     AWS_ACCESS_KEY_ID=ListVars.AWS_ACCESS_KEY_ID,
                                     AWS_SECRET_ACCESS_KEY=ListVars.AWS_SECRET_ACCESS_KEY)

    if(result == False):
        print '...'

    # creating -> backing-up
    if(result == 'creating'):
        print 'creating DB Instance ...'

    # available - active
    if(result == 'available' or result == 'active'):
        break

    time.sleep(20)

# Doing migrate DB
ssh_local = paramiko.SSHClient()
ssh_local.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh_local.connect(hostname=ListVars.OriDB_ip,
                  username=ListVars.RootDBServerUsername,
                  password=ListVars.RootDBServerPassword)

# Check this function has done or not?
if(b_CreateAWSDatabaseUserViaLocalMySQLCommandLine == False):
    print 'DumpDBFromLocalToAWS\n'
    b = False
    print 'Waiting for AWS Database active'
    while (not b):
        b = DumpDBFromLocalToAWS(ssh=ssh_local,
                                  database_name=ListVars.database_name,
                                  dbusernameLocal=ListVars.DBRootUsername,
                                  dbpasswordLocal=ListVars.DBRootPassword,
                                  to_server=ListVars.AWSDB_ip,
                                  dbusernameAWS=ListVars.AWSDBMasterName,
                                  dbpasswordAWS=ListVars.AWSDBMasterPassword)

```

```

if(not b):
    print '...'
    time.sleep(20)

print 'CreateAWSDatabaseUserViaLocalMySQLCommandLine\n'
b = CreateAWSDatabaseUserViaLocalMySQLCommandLine(ssh=ssh_local,
    host_ip=ListVars.AWSDB_ip ,
    rootusername=ListVars.AWSDBMasterName,
    rootpass=ListVars.AWSDBMasterPassword,
    dbusername=ListVars.dbusername,
    dbpassword=ListVars.dbpassword)

if(b == True):
    # state 3
    SaveMigrateDBState(CreateFile=False,
Function='CreateAWSDatabaseUserViaLocalMySQLCommandLine')
else:
    return False

# Check this function has done or not?
if(b_DoMySqlGrantForListServersAWS == False):
    print 'DoMySqlGrantForListServersAWS\n'
    b = DoMySqlGrantForListServersAWS(filename='hosts',
        ssh=ssh_local,
        host_ip=ListVars.AWSDB_ip,
        rootusername=ListVars.AWSDBMasterName,
        rootpass=ListVars.AWSDBMasterPassword,
        database_name=ListVars.database_name,
        dbusername=ListVars.dbusername,
        dbpassword=ListVars.dbpassword)

if(b == True):
    # state 4
    SaveMigrateDBState(CreateFile=False, Function='DoMySqlGrantForListServersAWS')
else:
    return False

ssh_local.close()

# Update config file for website
UpdateConfigFile()

#Write DB configuration info to AWS message queue
b = WriteDBConfigToSQS()
if(b):
    print 'Create msg SQS'
else:
    print 'msg SQS failed'
'done'

```


Appendix 3: Web migration

```
def MigrateWebsiteToAWS():
    #migrate website from a server to another server
    ListVars.WebsiteDirectory = os.environ['HOME'] + '/web'
    ListVars.Key_Pem_File = os.environ['HOME'] + '/lcc.pem'

    ListVars.AWSWebServerRootUsername = 'ubuntu'
    ListVars.AWSWebServerRootPassword=""
    ListVars.AWSWebsiteDirectory = '/var/www'

    ListVars.ami = 'ami-bae2abe8'
    ListVars.instance_type = 'm1.large'
    ListVars.key_name = 'my_key'
    ListVars.security_group = 'quick-start-1'

    web_path = ListVars.WebsiteDirectory
    zip_file = 'web_temp.tar'
    config_file = 'config.ini'
    remote_folder = ListVars.AWSWebsiteDirectory

    temp = raw_input('Do you want to load information that you had saved (Yes/No)? ')
    if(temp.upper() == 'Y' or temp.upper() == 'YES'):
        LoadMigrateWebsiteInfo()
    else:
        EnterMigrateWebsiteInfo()

    # Creating AWS instance
    ret, ip, id = CreateNewAWSInstance()
    if(ret==False):
        print 'Cannot create AWS instance'
        return False

    #Update AWS list file
    UpdateAWSListFile(ip=ip)

    #Add new item to ListAWS
    AddListAWS(ip=ip,
               user=ListVars.AWSWebServerRootUsername,
               password=ListVars.AWSWebServerRootPassword)

    # Moving website
    # Save to code files folder
    zip = zip_web_directory(zip_file=zip_file, web_path=web_path, new_config_db=config_file)

    if(zip == False):
        return False

    ""check MD5 for original file""
    # Save to code files folder
    local_file_data = open(zip_file, "rb").read()
    md1 = md5.new(local_file_data).digest()

    ssh = paramiko.SSHClient()
    ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
    try:
        if(toAWS==True):
            # AWS
```

```

        ssh.connect(hostname=ListVars.AWSWebServerIP,
                    username=ListVars.AWSWebServerRootUsername,
                    key_filename=ListVars.Key_Pem_File)
    else:
        # Local
        ssh.connect(hostname=ListVars.AWSWebServerIP,
                    username=ListVars.AWSWebServerRootUsername,
                    password=ListVars.AWSWebServerRootPassword)
except Exception, e:
    print ListVars.AWSWebServerIP
    print ListVars.AWSWebServerRootUsername
    print 'connect to remote server failed'
    print e
    return False

print "SFTP connected successfully!"

sftp = ssh.open_sftp()
while(True):
    try:
        sftp.put(zip_file, zip_file)
    except Exception, e:
        print 'Upload website failed'
        print e
        return False

    # break
    """check MD5 for uploaded file"""
    remote_file_data = sftp.open(zip_file).read()
    md2 = md5.new(remote_file_data).digest()

    if(md1 == md2):
        break
    else:
        print 'Error MD5: '
        print 'original:', md1
        print 'remote:', md2
        time.sleep(10)

sftp.close()

#extract web file
cmd = 'sudo tar -xf %s -C %s .' % (zip_file, remote_folder)
re = ExecuteCommand(client=ssh,
                    cmd=cmd, sudo=True,
                    password=ListVars.AWSWebServerRootPassword)
if(re == False):
    print 'Extract files failed'
    return False

ssh.close()
print "Website copied successfully!"

AWSLoadBalanceAddInstance(instance=str(id))

```