Automated decision system for efficient resource selection and allocation in inter-clouds

By

Jungmin Son

Under the supervision of

Professor Rajkumar Buyya

and

Dr. Rodrigo N. Calheiros

A minor project thesis submitted in partial
fulfilment of the requirement for the degree of
Master of Information Technology

June 2013

Department of Computing and Information Systems
The University of Melbourne

Automated decision system for efficient resource selection and allocation in inter-clouds

Jungmin Son

Supervisors: Prof. Rajkumar Buyya and Dr. Rodrigo N. Calheiros

---

ABSTRACT

Cloud computing enables system administrators to dynamically allocate resources and create virtual machines whenever necessary. It provides scalable and elastic provisioning of multiple servers without purchasing any hardware or physical infrastructure. Instead, utility computing is introduced with pay-as-you-run mechanism where the service is charged based on the actual amount of time and resource used for the service.

In cloud computing market, there are various providers commercially available to serve dynamically allocable resources to their customers, and different vendors have different service types and pricing policies. Since they have their own policies and APIs, it is problematic for system administrators to select the best provider for their needs and to integrate different providers into a single service.

Also, resource allocation from the provider is not guaranteed even if the provider is chosen, because cloud computing is fundamentally elastic service that manages resources dynamically. If the usage of the datacentre exceeds over its capacity, it cannot provide more resources, thus new virtual machines cannot be created in the datacentre.

In this project, we suggests a system architecture that decides the best providers based on the given requirements, and manages the desired resources by automatically creating new virtual machines from available providers.

Declaration

I certify that

- this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person where due reference is not made in the text,
- where necessary I have received clearance for this research from the University's Ethics Committee and have submitted all required data to the Department,
- the thesis is less than 6,000 words in length excluding text in images, table, bibliographies and appendices.

Signature_____

Date_____

Acknowledgement

I would like to express my deep gratitude towards my supervisors, Professor Rajkumar Buyya and Dr Rodrigo Calheiros. Prof. Buyya provides me an opportunity to work on this project with inspiration and motivation to pursue an academic achievement for the research. Also, Dr Calheiros provides me the full guidance and encouragement during the work. His committed supervision with helpful feedbacks, sparkling ideas and technological advices encourages and leads me to complete this project successfully.

In addition, I thank to my friend, Sori Kang, for proofreading of this thesis with her sharp insight toward the technology.

Finally, I would like to appreciate all my friends and family for their unconditional support. They always listen to me whenever I suffer from disappointment and difficulties and encourage me to complete the degree.

Jungmin Son
Melbourne, Australia
June 2013

Table of Contents

List of Tables

List of Figures

1 INTRODUCTION

Cloud computing is substituting traditional server-client model as pay-as-you-go based utility model in the field of software, platform and infrastructure. System administrators can get a new virtual machine easily from cloud providers whenever they need it, which is called Infrastructure as a Service model (IaaS). In this model, purchasing a new hardware and related infrastructure is not necessary; instead, they are hired for the required time and payed as they are used. Resources are dynamically provisioned according to the requirements and Service Level Agreements between providers and customers. In addition to IaaS model, Software as a Service (SaaS) and Platform as a Service (PaaS) are also crucial models in the cloud computing as they provide software or platform instead of infrastructure [1].

In the industry of cloud computing, especially in IaaS model, various vendors have been successfully providing a cloud service to users for years. Most major companies in IT industry have already launched their cloud services to the customers, including Amazon [2], Microsoft [3], Google [4] and HP [5]. Also, some start-up companies, such as GoGrid [6] and CloudSigma [7], have been set-up the services for cloud computing.

Since there are huge ranges of different providers with different policies, it is hard for users to choose the best provider suited for their requirements. It is a time-consuming job for the system administrators to explore various providers, check their policies, choose the best one for their requirement, and request to allocate the virtual machines.

Also, integrating multiple providers into one system is another issue because different providers have different APIs for development. If users want to expand their service using another vendor due to the cost or a lack of capacity of the current provider, combining it with a new provider together into a single system would cost huge work for system administrators.

For this reason, a number of inter-cloud approaches are introduced to make it easy to deploy a system on multiple clouds environment. Although various providers are succeed to run their services, and some inter-clouds libraries are implemented to help developers to adapt easily on the clouds, the jobs to find and choose the best provider and manage allocation of virtual machines are still on user's side.

1.1 Motivation

Although various inter-clouds approaches are introduced, none of them can choose the best provider or providers automatically in user's viewpoint. Federation approaches [8] are useful only if all cloud providers agree to get request from other cloud service and offer their resources voluntarily.

For example, if one provider receives a request for certain resources but cannot allocate them, it can be redirected to another provider which is agreed in advance, and the resource is provided by it. However, this approach is not available for independent and proprietary providers since they do not allow another provider to co-work with them by federation.

In addition, multi-cloud service does not make any decision to users. It helps users to easily manage and aggregate different cloud providers on a single site and provides simpler configuration reducing initial installation cost. However, it still does not suggest or decide the best provider. Although the service holds information of various providers, it cannot choose the most suitable provider. Similarly, inter-cloud libraries do not have any intelligence to select the provider. Developers can use the libraries to build a program integrating multiple clouds, but the library itself cannot find the best option for users.

A provider may not guarantee the resource needed by customer at the time of request, and in this case the customer should look for another provider that can offer sufficient resources. Since cloud computing is based on limited amount of resources, providers may happen to refuse to allocate resources to customers if there are no remaining resources. In that case, customers need to find another provider manually which cost time, labour and budget.

Therefore, it remains a difficult job for users to find the best provider for the specific requirements at the right time. In order to accomplish this job easily and quickly, the automated decision system is necessary. The system can find the best provider based on user's requirements and check whether the resource is available on the selected provider. If the system perceives that the best provider cannot allocate the resource, it can look for an alternative automatically.

1.2 Research Question

With the motivation, the following questions come up that make us to begin this research.

- The best provider fitted to the requirements of a system administrator
  - How can we find the best provider that charges lowest fare?
  - How can we recommend a provider which satisfies all requirements for the system administrator, such as location, operating system and resources?
- Automated resource allocation
  - Even if we can find the best provider, can we assure that the provider can allocated the resource for the system administrator at the moment when it is needed?
  - How can we automatically create the desired number of virtual machines in the cloud provider?
  - What are the other options if the selected provider does not have available resources?

This research aims to solve the problems above by building a system that can contribute to select the best solution easily and quickly. Using this system, users do not have to consider and figure out complex policies for different providers. Also, they can avoid managing the discovery of available resources when they need it. Since this system finds the available resource automatically, it can reduce overall time and cost spent to wander around various providers for system administrators willing to use cloud computing.

2 RELATED WORK

Previously, Zhang *et al.* suggested a similar approach to build a system for automatic recommendation of cloud service [9]. In their approach, the system recommended the best provider to the user based on computing, storage and network service. The system consisted of three layers, including Widget, Application Logic and Configuration Management layer in order to give the best answer. Although it could suggest the cheapest provider, it actually did not allocate the resource from provider to the user. Users should find the available resources by themselves from the outcome of the recommender system. In the case that the resource was not able to be allocated from one provider, the user would try another provider manually, which is not fully automated. Therefore, our system in this project can work complementary to Zhang's system. The output list recommended from their system can be used as an input data of our system, which will allocate resources automatically.

2.1 Survey of Inter-cloud Approaches

There are a number of approaches designed for working on inter-clouds environment [8]. Grozev *et al.* reviewed overall inter-cloud architectures and brokering mechanisms. According to his work, inter-cloud approaches are broadly classified into two categories: volunteer federation and independent approaches.

Volunteer federation is a method to collaborate and exchange resources voluntarily between different providers and viable for governmental clouds and private cloud portfolios. It is further divided into centralised or peer-to-peer approach depending on existence of centralised server aggregating different clouds. In centralised approach, there is a central server which manages overall resources and forwards messages across the clouds, while peer-to-peer approach does not have the central server [8].

Independent approach, or multi-cloud approach in other word, is another way of inter-clouds being aggregated by an application or by a broker. This approach is independent to providers, so that they do not communicate to each other voluntarily. Instead, there is a separate broker or an application that aggregates and controls them independently. In detail this approach can be classified into a service or a library in terms of the method in which they aggregate providers. In service approach, a service runs separately from a client and controls different cloud machines within the service, while a client uses libraries to aggregate them in library approach [8].

There are various projects recently conducted by different groups of researchers on this field [8]. InterCloud by Buyya *et al.* [10], is a centralised volunteer federation project that illustrates the architecture and the elements of federation with Cloud Exchange, Cloud Coordinator and Cloud Broker. Cloud Exchange is the centralised server that aggregates information from various Cloud

Brokers on user side and finds the available supply using Coordinators. Cloud Broker identifies suitable cloud service providers through Cloud Exchange and negotiates with Cloud Coordinators in order to get allocated resources.

RESERVOIR [11] [12] by Rochwerger *et al.* is the similar approach of federation but with peer-to-peer architecture. This system does not have a centralised controller, like Cloud Exchange in InterCloud project, but manipulates requests across the providers on peer-to-peer basis. Instead of the central server aggregating information, it determines a need of extra resources within the original provider and allocates more resources when necessary. Although these approaches enable to integrate multiple clouds for on demand, and flexible and reliable access to resources, it is only available between providers in federation manner.

On the other hand, multi-clouds services integrate and manage multiple providers within one service. OPTIMIS [13] is one of research projects for independent multi-clouds service. It deploys agents into multiple data centres and controls them using central service. Also, RightScale [14] is another multi-cloud service which is commercially available for multi-clouds and gives a single Dashboard and API for multiple cloud resource pools. It provides the configuration framework that gives users constant templates to set up the server easily as well. However, these services do not choose the best provider, but only use the chosen ones.

In the field of implementation, various libraries are available to integrate the providers. Jclouds [15], a Java library to aid developing Java program on different cloud providers, has been developed as open source and widely used among Java developers. Other libraries based on different languages, e.g. Apache LibCloud [16] for Python, Apache DeltaCloud [17] for Ruby and SimpleCloud [18] for PHP, have been developed for this purpose. While the libraries are helpful to develop a service using various providers, they still cannot choose the provider. It is not able to recommend the provider to the user since libraries does not have any active intelligence to make a decision. Also, they cannot be used to manage allocation of resources by themselves. System administrators should implement a program to allocate resources automatically even if they use the inter-cloud libraries.

Table 1 Different types of inter-cloud approaches [8]

Approach Category	Examples
Federated inter-clouds, centralised	InterCloud [10]
Federated inter-clouds, peer-to-peer	RESERVOIR [11] [12]
Independent inter-clouds, service	OPTIMIS [13], RightScale [14]
Independent inter-clouds, library	Jclouds [15], Apache LibCloud [16], Apache DeltaCloud [17], SimpleCloud [18]

2.2 Review of Current Practices

There are hugely various providers in cloud computing industry, and each provider has its own policies. Since reviewing all providers is impossible in this report, we focus on the three providers: Amazon, GoGrid and CloudSigma, who have distinct policies from each other. The policies of these three providers can represent most of the other providers.

In this section, pricing policies are described in terms of resource allocation types, resource rates and other costs. Each provider has different way of allocating resources to the machine. Some providers, such as Amazon and GoGrid, have the number of predefined options not allowing customisation, thus users should choose one of the resource sets. Other providers, like CloudSigma, let users customise the resource of a machine; users can select the number of cores and the size of RAM as they want.

All rates in this research are represented in U.S. Dollars as it is the most widely used in the industry.

2.2.1 Amazon AWS EC2 [2]

Amazon is the world-leading company in cloud computing who provides various services in their Amazon Web Services such as EC2, a computing service, and S3, a storage service. They operate multiple datacentres in various locations including the United States, Europe, Singapore, Japan and Australia.

Table 2 Predefined resource sets available on Amazon

Instance type	CPU (cores)	RAM (GB)	Storage (GB)
M1 Small Instance	1	1.7	160
M1 Medium Instance	2	3.75	410
M1 Large Instance	4	7.5	850
M1 Extra Large Instance	8	15	1690
M3 Extra Large Instance	13	15	EBS
M3 Double Extra Large Instance	26	30	EBS
Micro Instance	1	0.613	EBS
High-Memory Extra Large Instance	6.5	17.1	420
High-Memory Double Extra Large Instance	13	34.2	850
High-Memory Quadruple Extra Large Instance	26	68.4	1690
High-CPU Medium Instance	5	1.7	350
High-CPU Extra Large Instance	20	7	1690
Cluster Compute Eight Extra Large Instance	88	60.5	3370
High Memory Cluster Eight Extra Large Instance	88	244	240
Cluster GPU Quadruple Extra Large Instance	33.5	22	1690
High I/O Quadruple Extra Large Instance	35	60.5	2048
High Storage Instances	35	117	48000

2.2.1.1 Resource allocation types

Basically, Amazon EC2 has 18 types of pre-defined machines including 6 Standard, 1 Micro, 3 High-memory, 2 High-CPU, 3 Clusters, 1 High-IO and 1 High-storage instances based on the number of cores, the size of memory and other performance parameters. Table 2 shows all resource allocation types used within Amazon.

2.2.1.2 Instance rates

Price is different depending on the instance types described above, location of the datacentre and the operating systems run on the server. In addition, Amazon has three different categories of pricing rules. They differ to each other based on the period of reservation, the availability and the reliability.

- **OnDemand**: The instance is allocated for users once it is requested if the datacentre contains enough resources to run it. If the resource is not available at the time of request, it is not guaranteed to allocate the resource. There is no commitment as well as no upfront fee to start new machine. The price is totally up to how long the user actually runs the machine. The hourly price is fixed by the provider and not changed. The minimum charge is equivalent to the one hour price even if it is terminated before one hour.

Table 3 Amazon's hourly price for OnDemand service

Instance type	Sydney Linux	Sydney Windows	US-East Linux	US-East Windows
M1 Small Instance	$0.08	$0.12	$0.06	$0.09
M1 Medium Instance	$0.16	$0.23	$0.12	$0.18
M1 Large Instance	$0.32	$0.46	$0.24	$0.36
M1 Extra Large Instance	$0.64	$0.92	$0.48	$0.73
M3 Extra Large Instance	$0.70	$0.98	$0.50	$0.78
M3 Double Extra Large Instance	$1.40	$1.96	$1.00	$1.56
Micro Instance	$0.02	$0.02	$0.02	$0.02
High-Memory Extra Large Instance	$0.50	$0.57	$0.41	$0.51
High-Memory Double Extra Large Instance	$0.99	$1.14	$0.82	$1.02
High-Memory Quadruple Extra Large Instance	$1.98	$2.28	$1.64	$2.04
High-CPU Medium Instance	$0.18	$0.29	$0.15	$0.23
High-CPU Extra Large Instance	$0.73	$1.14	$0.58	$0.90
Cluster Compute Eight Extra Large Instance	N/A	N/A	$2.40	$2.97
High Memory Cluster Eight Extra Large Instance	N/A	N/A	$3.50	$3.83
Cluster GPU Quadruple Extra Large Instance	N/A	N/A	$2.10	$2.60
High I/O Quadruple Extra Large Instance	N/A	N/A	$3.10	$3.58
High Storage Instances	N/A	N/A	$4.60	$4.93

Table 4 Amazon's Reserved price in Sydney for Linux with different contract period

Instance type	1 Year Up-front	1 Year Hourly	3 Year Up-front	3 Year Hourly
M1 Small Instance	$67.00	$0.05	$97.00	$0.04
M1 Medium Instance	$133.00	$0.09	$194.00	$0.08
M1 Large Instance	$266.00	$0.19	$388.00	$0.15
M1 Extra Large Instance	$531.00	$0.37	$776.00	$0.30
M3 Extra Large Instance	$585.00	$0.41	$899.00	$0.33
M3 Double Extra Large Instance	$1,170.00	$0.82	$1,798.00	$0.65
Micro Instance	$23.00	$0.02	$35.00	$0.02
High-Memory Extra Large Instance	$282.00	$0.23	$423.00	$0.19
High-Memory Double Extra Large Instance	$564.00	$0.46	$846.00	$0.37
High-Memory Quadruple Extra Large Instance	$1,128.00	$0.92	$1,692.00	$0.74
High-CPU Medium Instance	$174.00	$0.12	$266.00	$0.11
High-CPU Extra Large Instance	$696.00	$0.49	$1,064.00	$0.42

- **Reserved**: The instance is reserved for the period that customers have committed. In contrast to OnDemand service, the resource is guaranteed to be allocated to the user when it is requested. Therefore, customers need to pay the one-time reservation fee for contract periods upfront, plus hourly rate. Since there is long-term commitment, the hourly charge is cheaper than OnDemand pricing. Also, if users pay more up-front fee, they can get less hourly rate, although it is still needed to pay for hourly rate. The upfront fees and hourly rate is fixed by the provider. The minimum charge is the upfront fee. If users contract to reserved service but never run the machine, still they need to pay the upfront fee.

Table 5 Amazon's Spot price per hour in different locations and operating systems

Instance type	Sydney Linux	Sydney Windows	US-East Linux	US-East Windows
M1 Small Instance	$0.01	$0.02	$0.01	$0.02
M1 Medium Instance	$0.01	$0.03	$0.02	$0.04
M1 Large Instance	$0.03	$0.07	$0.04	$0.08
M1 Extra Large Instance	$0.05	$0.13	$0.08	$0.16
M3 Extra Large Instance	$0.06	$0.14	$0.09	$0.17
M3 Double Extra Large Instance	$0.12	$0.28	$0.18	$0.34
Micro Instance	$0.00	$0.01	$0.00	$0.01
High-Memory Extra Large Instance	$0.04	$0.07	$0.06	$0.10
High-Memory Double Extra Large Instance	$0.07	$0.14	$0.12	$0.20
High-Memory Quadruple Extra Large Instance	$0.14	$0.28	$0.24	$0.40
High-CPU Medium Instance	$0.02	$0.05	$0.03	$0.06
High-CPU Extra Large Instance	$0.07	$0.20	$0.11	$0.24
Cluster Compute Eight Extra Large Instance	N/A	N/A	$0.27	N/A
High Memory Cluster Eight Extra Large Instance	N/A	N/A	$0.34	N/A
Cluster GPU Quadruple Extra Large Instance	N/A	N/A	$0.60	N/A

- **Spot**: Users bid for the capacity, and the resource is allocated to the user only when the bid price is higher than current Spot price. Allocation of resource is guaranteed if the bid price is higher. The current Spot price is changing every five minutes based on the capacity and the demand of overall service. Once the Spot price exceeds the user's bidding price, resource is not available any more, and the machine terminates instantly. In this case, users are exempted from paying for the last hour if it was less than one hour. Spot instance is unreliable since it can be terminated anytime by the service provider, not by users, when the price gets higher.

2.2.1.3 Other costs

Users should pay extra for the data transfer and additional IP address. Data transfer is free for inbound and $0.05 to $0.12 per GB for outbound traffic. For the outbound traffic, the first 1GB per month is free of charge, and start with $0.12 per GB after 1GB up to 10TB per month. After this, price is gradually decreasing to $0.05 once it exceeds 350TB. Allocation of IP address is free of charge for the one address per machine, but additional addresses are charged $0.005 per address per hour.

2.2.2 GoGrid [6]

GoGrid is a cloud provider purely dedicated on IaaS service. They have three main locations including San Francisco CA, Ashburn VA, both in the United State, and Amsterdam in Netherland.

2.2.2.1 Resource allocation types

GoGrid has 7 pre-configured types of resources for cloud service depending on the number of cores, the size of memory and the size of storages. It is similar to Amazon, but has much simpler types. Table 6 shows the product types and their allocated resources.

Table 6 Resource pre-sets of GoGrid

Product	CPU(Cores)	RAM(GB)	Storage(GB)
X-Small	0.5	0.5	25
Small	1	1	50
Medium	2	2	100
Large	4	4	200
X-Large	8	8	400
XX-Large	16	16	800
XXX-Large	24	24	1200

2.2.2.2 Instance rates

Pricing options for GoGrid is simpler than for Amazon. It provides only four types of pricing schemes: hourly, monthly, semi-annually and annually. Hourly pricing is similar to OnDemand

pricing of Amazon, but the other three types are new in GoGrid. For the contract-based services, they charge all price upfront before the instance is running, and do not have additional usage charge. Comparing them to the hourly rate, users are offered discounts for the commitment of certain period. While Amazon's Reserved Pricing scheme charges separate hourly rate from the upfront fee, GoGrid integrates them together. Thus, users should pay same amount of money regardless of the actual running hour of the machine.

Table 7 Price set of GoGrid

Product	Hourly	Monthly	Semi-annual	Annual
X-Small	$0.04	$18.13	$99.69	$181.25
Small	$0.08	$36.25	$199.38	$362.50
Medium	$0.16	$72.50	$398.75	$725
Large	$0.32	$145	$797.50	$1,450
X-Large	$0.64	$290	$1,595	$2,900
XX-Large	$1.28	$580	$3,190	$5,800
XXX-Large	$1.92	$870	$4,785	$8,700

2.2.2.3 Other costs

Data transfer rate for GoGrid is similar to Amazon. All inbound and the first 1GB outbound traffic is free of charge, and it costs $0.12 to $0.08 per GB depending on monthly usage.

2.2.3 CloudSigma [7]

CloudSigma is an IaaS cloud company based in Zurich, Switzerland. They have distinctive pricing policy unlike other major companies. Datacentres are located in Zurich, Switzerland and Las Vegas, the U.S.

2.2.3.1 Resource allocation types

CloudSigma has more flexible pricing and resource allocation policy. It does not have bundled resource sets like Amazon or GoGrid. Instead, users are able to flexibly select resources, and price is proportional up to the users' choices. Table 8 shows minimum and maximum resources.

Table 8 Minimum and maximum resources in CloudSigma

Resources	CPU (Cores)	RAM (GB)	Storage (GB)
Minimum	0.5	0.5	1
Maximum	20	32	1024

2.2.3.2 Instance rates

CloudSigma provides two different schemes of pricing depending on commitment: Subscription pricing and Burst pricing. Since they do not have any pre-defined set of resource types, prices for both schemes are calculated solely by the amount of resource that users selected.

- **Subscription pricing:** This is similar to the GoGrid's pricing with minimum contract of one month. Users need to choose the resources and pay for certain period before they start the service. Table 9 shows the rate elements for various resources. Although the rate for CPU and RAM are described as hourly based in the table, they should be calculated monthly since the minimum commitment period is a month.

Table 9 Price table for Subscription pricing in CloudSigma USA

Resource	Price	Unit
CPU (Core)	$0.022	Core/hour
RAM (GB)	$0.025	GB/hour
Storage (GB)	$0.140	GB/month

Based on the above table, we calculate some examples of prices for different subscription periods in CloudSigma (Table 10).

Table 10 Price per unit for different subscription period in CloudSigma USA

Period	1 months	3 months	6 months	1 year	2 years	3 years
Discount	0%	3%	10%	25%	35%	45%
CPU (Core)	$15.84	$46.09	$85.54	$142.56	$247.10	$313.63
RAM (GB)	$18.00	$52.38	$97.20	$162.00	$280.80	$356.40
Storage (GB)	$0.140	$0.41	$0.76	$1.26	$2.18	$2.77

- **Burst pricing**: Burst pricing is pay-as-you-run plan for CloudSigma like OnDemand pricing of Amazon. Instead of having fixed hourly rate, the price of service is fluctuating with 20 levels based on how busy they are. The medium level, level 10, has same price as the monthly rate of their Subscription pricing, and lower levels are cheaper while higher levels are more expensive. They update the price on every five minutes like Amazon's Spot pricing, but the burst pricing on CloudSigma is reliable service that the machine will not be deleted by provider due to price update. The Spot service on Amazon, however, allows the provider to delete the machine once the updated price becomes higher than the bid price (Table 11).

2.2.3.3 Other costs

Data transfer is very simple on CloudSigma: free for incoming traffic and $0.05 per GB for outgoing traffic.

Table 11 Burst Pricing for CloudSigma USA

Level	CPU(Core/hour)	RAM(GB/hour)	Transfer(GB)
1	0.0106	0.0151	0.025
2	0.011	0.0154	0.0263
3	0.0114	0.0157	0.0288
4	0.0117	0.016	0.0327
5	0.0125	0.0167	0.0365
6	0.0132	0.0173	0.0398
7	0.0147	0.0186	0.0429
8	0.0169	0.0205	0.0455
9	0.0191	0.0225	0.0481
10	0.022	0.025	0.05
11	0.0235	0.0263	0.0519
12	0.0249	0.0275	0.0538
13	0.026	0.0285	0.0564
14	0.0268	0.0292	0.0602
15	0.0275	0.0298	0.0641
16	0.0282	0.0305	0.0679
17	0.0312	0.033	0.0718
18	0.0341	0.0356	0.0769
19	0.0385	0.0395	0.0833
20	0.0444	0.0445	0.091
Storage = $0.24 /GB/month			

2.2.4 Pricing factors and user options

Throughout the review, we can find common factors affecting to the total cost of the cloud computing systems charged from a provider.

- Resources: computing resources allocated to the virtual machine such as CPU, RAM, Disk, etc.
- Geographical location: different location has different price even in the same provider
- Operating Systems: some operating systems are applied additional charge
- Minimum commitment: some pricing types have contract period
- Reliability: some services, like Spot at Amazon, is not reliable as the virtual machine can be terminated depending on provider's situation
- Data traffic: providers charge for data traffic ingress and egress of the machine

As we discussed from the previous section, there are various factors consisting of final cost for cloud computing services. Some of the pricing factors are crucial for the customer when they select the service, while some are ignorable depending on the type of their job. For example, a customer

who wants to provide a web service solely within Australia and to give the best quality of service to its users, geographical location of the cloud machine is vital. In this case, the customer should choose a provider who has datacentre in Australia, such as Amazon, even though it is more expensive than others.

Similarly, if the machine must not be possibly deleted by the provider in the middle of the execution, they should avoid using Spot scheme in Amazon although it is the cheapest option. In other case, like keeping sensitive data in cloud service, it can be strictly prohibited by local law to store those sensitive data into another location outside of the country.

Therefore, the following factors are crucial for users when they choose a cloud provider:

- **Price:** definitely crucial for users as they prefer cheaper service if other conditions are equivalent.
- **Computational resources:** users need at least certain amount of resources to run their service stable
- **Geographical location of data centre:** some service can be fatal for the location if latency and response time are important
- **Specific Operating Systems:** if a program is developed on certain environment, it is necessary to have a specific operating system
- **Contract period:** how long the service runs on the virtual machine
- **Duration of actual usage of computing power:** actual usage different from contract. Although contract period stands for a whole period of the service, actual usage duration may be less than the contract period if the machine is not running for 24 hours a day.
- **Reliability:** some users do not care of the reliability as their service can be terminated anytime

When we implement the decision system in this project, the above factors should be taken into an account to gather user's requirement.

3 AUTOMATED DECISION SYSTEM

Based on the review of various providers, we design the automated decision system and implement the working prototype of the system. In this chapter, we will describe the architecture of the system with each component and the method of implementation for the prototype.

3.1 Design

The system is mainly composed with two independent parts: Resource Selection Decision Maker and Automated Resource Allocator (Figure 1). Decision Maker decides the best provider based on the given users' requirements and makes a list of suggested providers according to the price. This list becomes an input of the second part, Resource Allocator, which actually allocates the requested resource from the chosen provider, and tries another provider if it is not available.

As each part is designed to work independently, one part can be substituted by another program or module with better performance. For example, Declarative Recommender System suggested by Zhang *et al.* [9] can be substituted for Decision Maker. Since their system includes blob storage and network usage costs, it would produce better results for those services, while the Decision Maker in this project focuses more on compute service. In such case, the Resource Allocator can read the output of Zhang's system and try to allocate resources.

Two databases in the architects hold all information about the cloud providers. Cloud Information Database stores provider's locations, operating systems, resource sets and pricing policies. Provider Credential Database, on the other hand, holds credential information such as usernames and passwords used to log in to the provider.

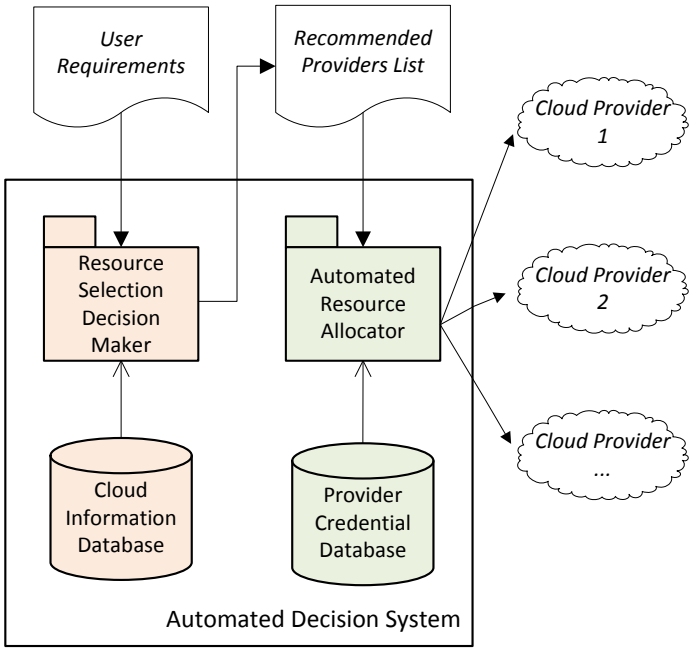

Figure 1 System Architecture of Automated Decision System

3.1.1 Resource Selection Decision Maker

Resource Selection Decision Maker lists the recommended providers and their services by analysing user's requirements. The components are designed in Model-View-Controller model, as it can easily separate the components reducing dependencies to other components. Figure 2 shows the Model-View-Controller architecture of Decision Maker.

At the beginning, users enter their requirements for the cloud service into the view (Table 12). Once the information is provided to the controller, the controller retrieves all records matching the requirement from the model through a query to Cloud Information Database. The requirements are used as constraints for the model, e.g. records retrieved should have more resources than the requirements, matched location and operating system, and shorter contract period. If the user wants three months contract, for example, a service with one month contract can be retrieved but not the six months contract service.

Once it gets all candidate providers and their service types, the model calculates expected price based on the usage period given by user. For some providers, like GoGrid, only contract period is affected to the total price, since they do not charge to actual usage if they had contracted for the period. However, in other cases, such as Amazon's Reserved pricing, the actual running time of the instance influences to usage charges, in addition to the upfront fee for the contract. Hence, the actual usage period should be included to calculate the correct price.

After the prices are calculated for each service and provider, the provider list is sorted by price and given to the user. Each item of the suggested list has information including the name of cloud provider, resources to be allocated, service type, contract period and expected price.

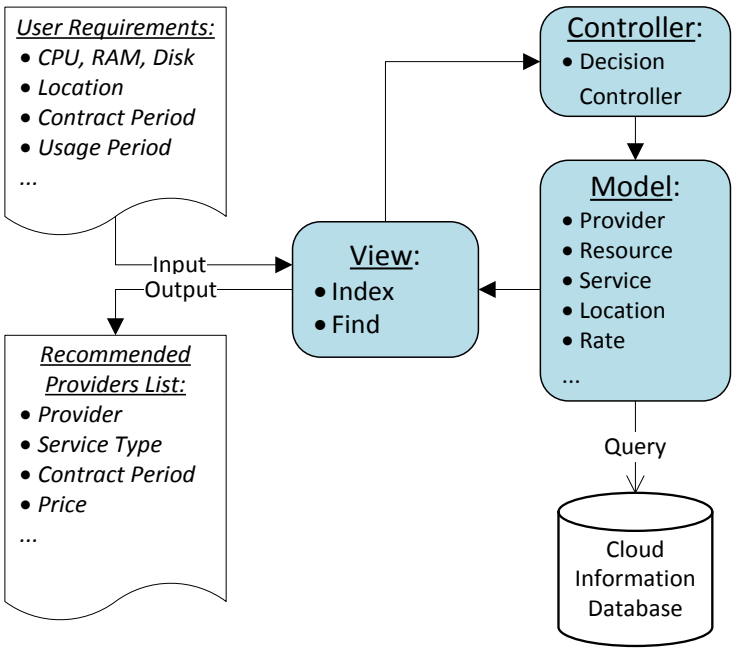

Figure 2 Resource Selection Decision Maker architecture of Model-View-Controller

Table 12 Fields of user requirements

Field	Description	Example
Resource (CPU, RAM, Disk)	Required resource by user.	CPU=0.5, RAM=512, Disk=20
Operating System	Operating system used for the system. User can select "Any" if there is no preference.	Linux
Location	Physical location of the system. User can select "Any" if there is no preference.	Any
Contract Period	Period to be contracted. If user do not want any contract, zero can be put	3 months
Allow Unreliable Services?	User can check this field if unreliable service like Spot pricing is allowed	Not allowed
Usage Period	Expected actual usage period within the contract. This period should be equal or less than contract period.	2 months
# of machines	How many machines are required?	2
Allow partial allocation?	If user wants more than one machine, is it necessary to allocate all of them within a single service? If allowed, machines can be separately allocated in different services or different providers.	Allowed

3.1.2 Cloud Information Database

Cloud Information Database stores all information about providers and their services including resource sets, geographical locations, operating systems, contract periods and prices. The database consists of various tables related to the service and the price.

As we discussed in the review of current practices, different providers have different policies, and some providers have the unique policy which is far different from other providers. In our system, however, all information should be consistently stored in a single database to calculate the expected cost without help from extra programs. To achieve it, each component consisting of the price is firstly defined as a single table, including Provider, ResourceSet, OperatingSystem, Geo and ServiceType table. As ResourceSet is dependent to the provider, it has a foreign key referring to Provider table.

After that, Rate table is defined with foreign keys to all components, but actual pricing elements are stored in a separate table from original Rate, named RateElement. The RateElement table has two fields: price and period, where the price is added to the total cost on every given period repeatedly for a whole usage time. Hence, total price can be calculated with the usage time *T*, and the price and period value in RateElement table with the equation:

$$Price_{Total} = \sum \left(Price_{RateElement} \times \left\lceil \frac{T}{Period_{RateElement}} \right\rceil \right)$$

For example, Amazon's one-year Reserved instance pricing is made up of an up-front fee payable once a year and an hourly rate charged for running hours of the machine. In this case, there are two entities in the RateElement table: a record with up-front price with one year in period field,

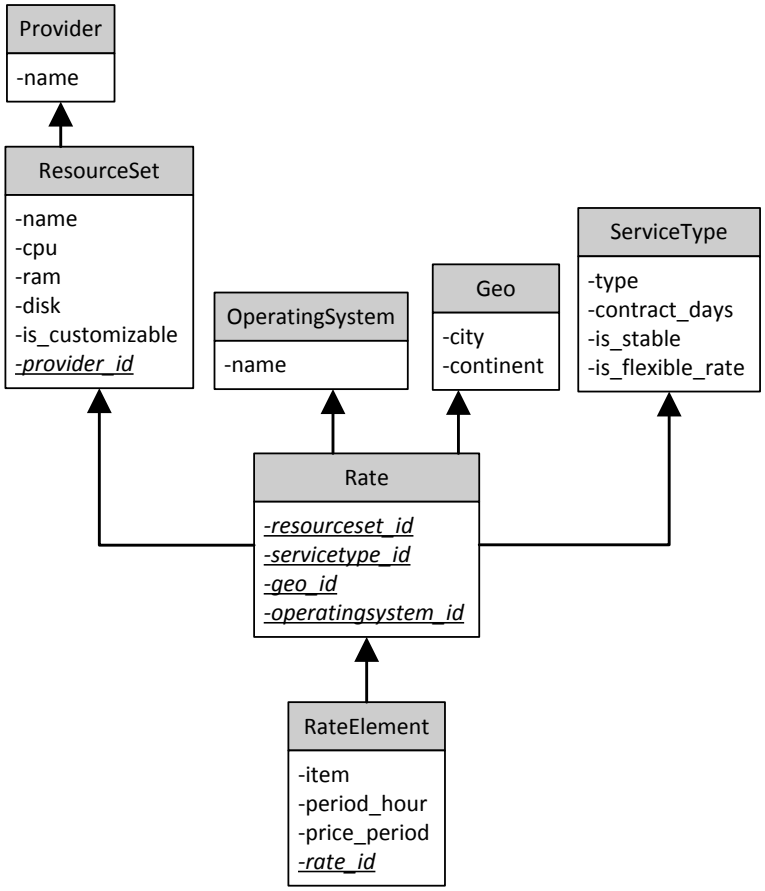

Figure 3 Cloud Information Database schema

and a record with hourly price with one hour in period field. Therefore, when we calculate the price for the specific service, all records in RateElemment belonging to the Rate are retrieved and added to the total cost.

Table 13 Fields in Cloud Configuration list

Field	Description	Example
Provider	Provider name to request resources	aws-ec2
Resource set	Resource-set ID if applicable, or actual amount of resources in the order of CPU cores, RAM in Mbytes and Disk size in Gbytes.	t1.micro *or* (2,512,20)
Location	Location of the datacentre. It can be blank if there is no preferred location.	ap-southeast-2a
Image details	Image ID if applicable, or detailed information of desired operating systems in terms of name, version and architecture	ami-e2ba2cd8 *or* Ubuntu,12.04,64
Number of machines	How many machines to be allocated for the spec	2
Allow partial allocation?	If user wants more than one machine, can they be allocated partially on different services, in the case of partial failure?	Yes

3.1.3 Automated Resource Allocator

Automated Resource Allocator actually connects to providers and requests allocation of the resources provided in the list of specifications. Figure 4 shows a class diagram of the Resource Allocator.

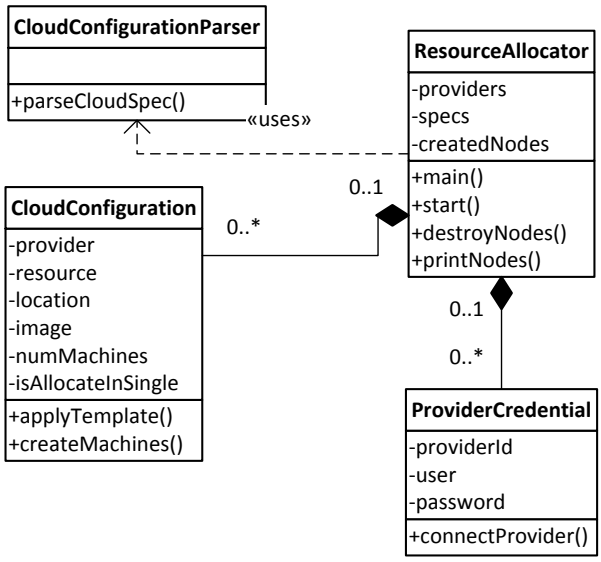

Figure 4 Class diagram of Automated Resource Allocator

The program requires a Cloud Configuration list as an input of the program. It is an actual cloud specification list; the output from the Decision Maker. Each row in the list should have a field of provider, resource set, location, image information and the number of machines to be allocated.

Figure 5 describes the flow of Resource Allocator. It starts parsing the given configuration list and building a collection of Cloud Configuration objects storing the parsed information. Once all elements are parsed, it tries to create virtual machines to the provider using that object. When the system connects to the provider, it authenticate the user with account information (ID, password) stored in Provider Credential Database. If the connection is successfully established, it requests the number of virtual machines with the specific information of resource, location and image to the cloud provider with a constant timeout. Once either the creation is done on the provider, or timeout exceeds, the Allocator analyses the results and proceeds to the next option.

Depending on the user's selection of whether allocation needs wholly on a single provider or not, there is an additional process before trying the next option. If the partial allocation to several providers is allowed, it adds remaining number of virtual machines to the next option in the list, and it tries again to the next provider. If the user specifies to force to allocate all machines in one provider, it destroys partially allocated machines from the previous provider. Since they are already created on the provider, the system should destroy the created machines before trying to the next provider. After all machines are destroyed, it adds up the entire number of machines to the next provider, and tries again for the next entry.

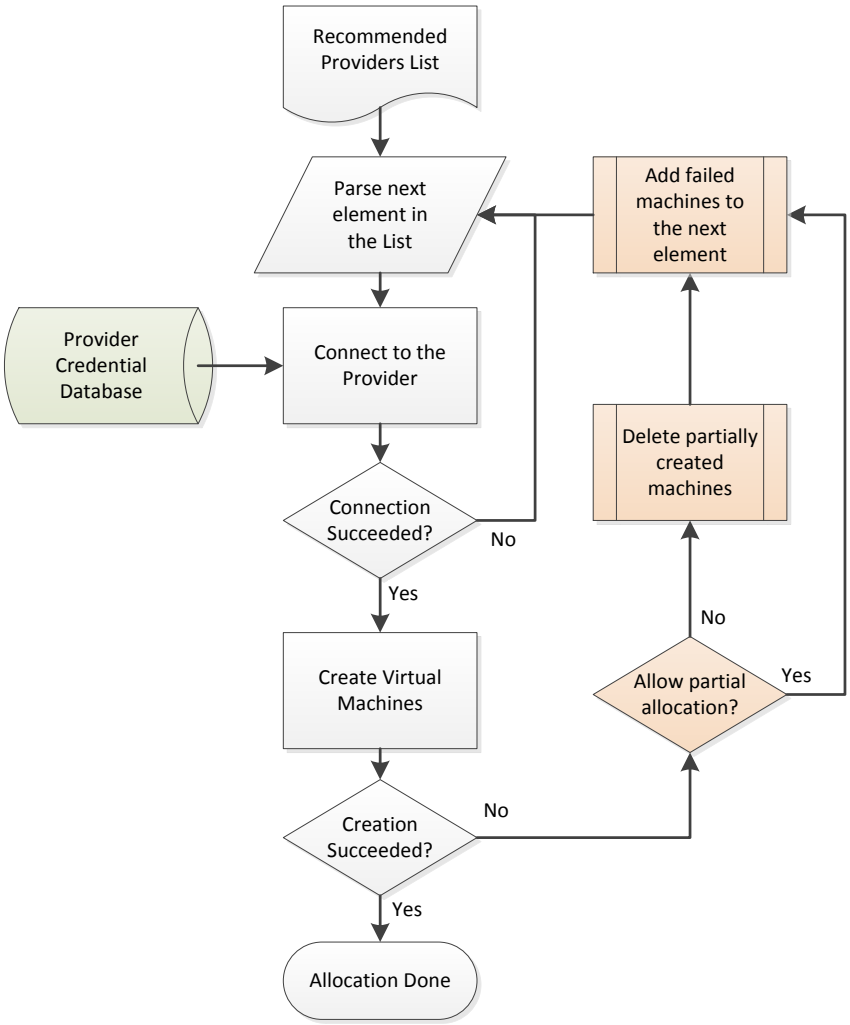

Figure 5 Flowchart of Automated Resource Allocator

3.1.4 Provider Credential Database

In order to connect to the provider listed in the configuration specs, we need credential information to authenticate the user for the provider. Provider Credential Database specifies the user information to log in to the provider. It consists of three fields: provider ID, user ID and password (Table 14). Since the provider information is same across all trials, authentication information in this database is globally used for a whole allocation process.

Table 14 Fields in Provider Spec list

Field	Description	Example
Provider ID	Provider name	aws-ec2
User ID	Encoded user's ID for the provider.	USERAKIAIXCOOUSER
Password	Encoded user's password for the provider	PASS3kIEWORDubPASS

3.2 Implementation

The prototype of the system is implemented using Java, Jclouds [15] and Ruby on Rails [19] in order to validate the suggested architecture.

First of all, Resource Selection Decision Maker is implemented as Model-View-Controller architecture using Ruby on Rails [19] web framework. The framework regards the database table as an object, named Active Record, using Object Relationship Mapping; hence it allows us to simplify the interaction between models and database. We implements web interface as a view model to make it simple to enter the user requirements and retrieve the result (Figure 6).

When the system produces a result, it builds two lists with the same contents but different formats. One is formatted with HTML tags to help users to find the result clearly, and the other is formatted with colon and comma to make it easy to parse from the Resource Allocator (Figure 7).

Cloud Information Database is implemented with SQLite [20], an integrated relational database in Ruby on Rails [19] framework, and the schema is automatically structured and organised along with the model of the framework.

Building the database up with full records of information for several providers and their services is another consideration, as there are tons of providers which have several different service types. Their service types and pricing policies are generally explained in their web site as HTML format, which is difficult to convert into relational database entities. Labour of parsing and converting them

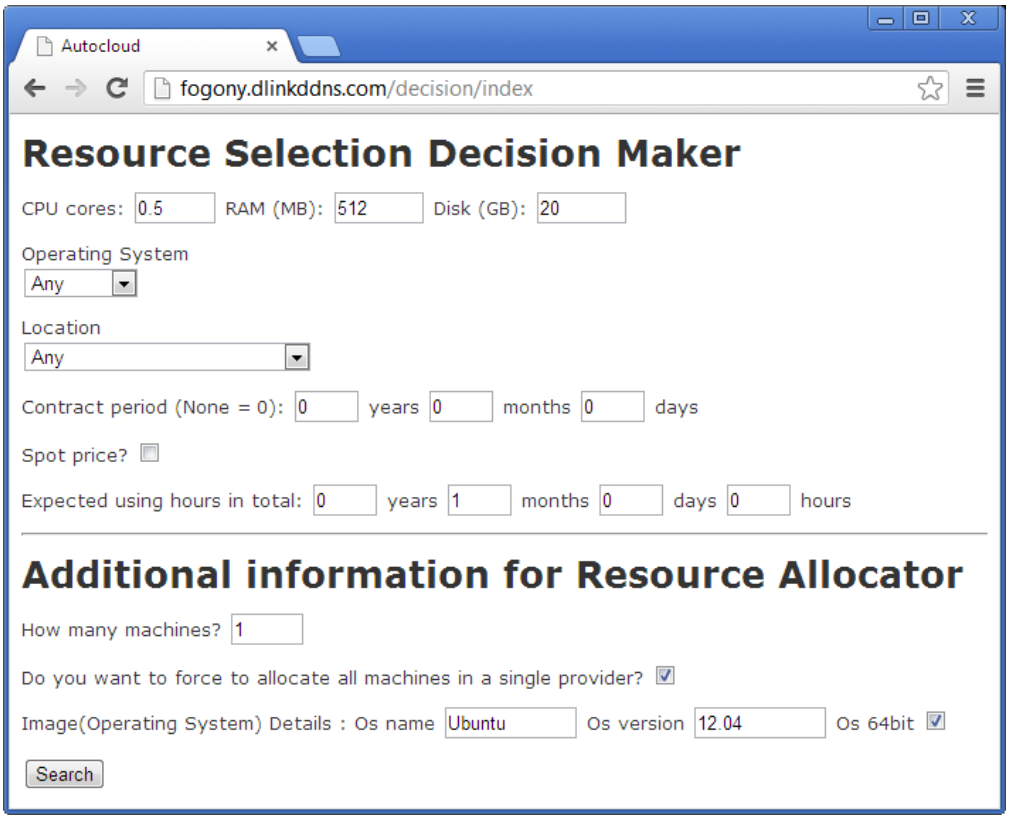

Figure 6 Screenshot of user input of Resource Selection Decision Maker

into the database manually is hard-working and time-consuming job, and even the data can be inconsistent as they are continuously updating by the provider. Therefore, further research for automatic data mining is expected to make it easy of crawling price data from the web pages, making a structure and inserting it into the database automatically.

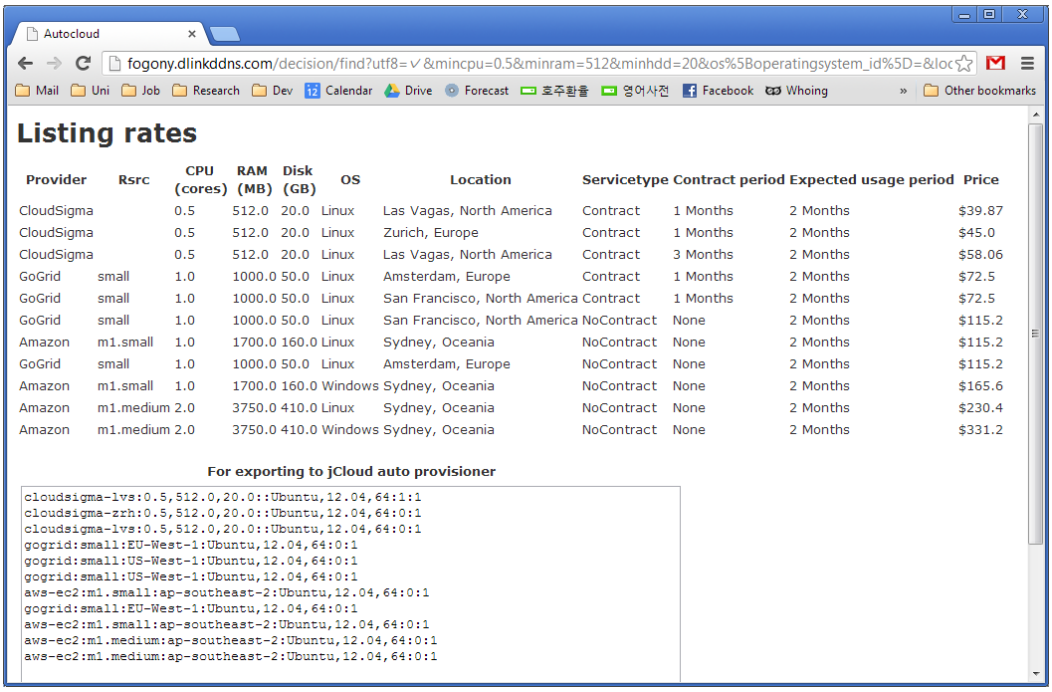

Figure 7 Suggested provider list resulting from Resource Selection Decision Maker

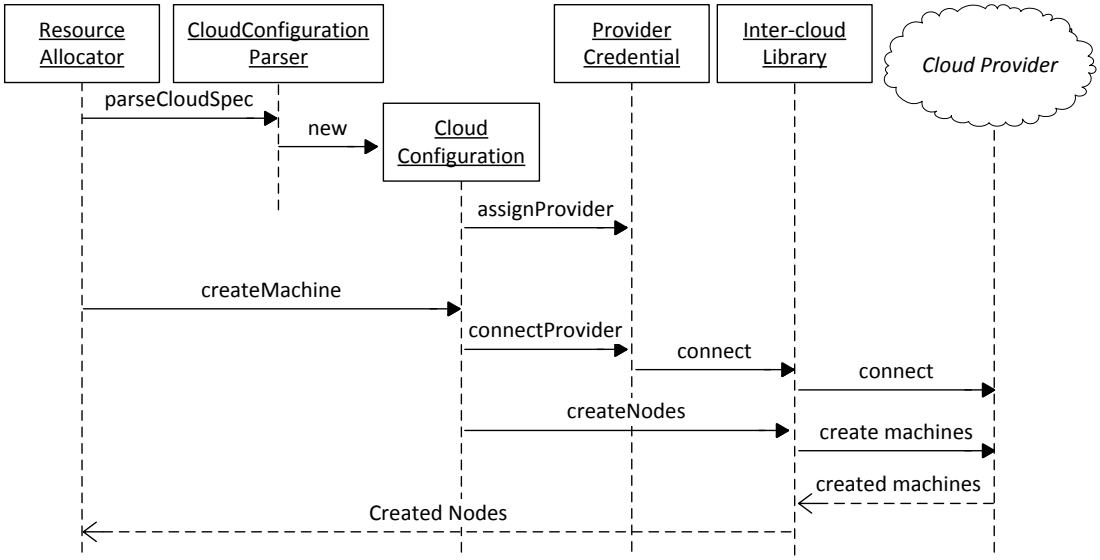

Figure 8 Sequence diagram of Automated Resource Allocator

Unlike the implementation of Decision Maker using Rails for a web interface, Automated Resource Allocator is implemented with Java on console interface (Figure 8, 9). In order to cover various providers in a single program, we deploy Jclouds [15], a multi-cloud library written in Java. Jclouds allows developers to use homogeneous APIs to connect to different clouds, thus, developers do not need to use different APIs used on different providers. Instead, Jclouds provides a single interface to connect, create and destroy virtual machines for various providers. The program is runnable on command line with two arguments: a file of cloud configuration list and a file of provider credential information list. Provider Credential Database is implemented as a list of provider name, user name and password and parsed by the Resource Allocator in our prototype.

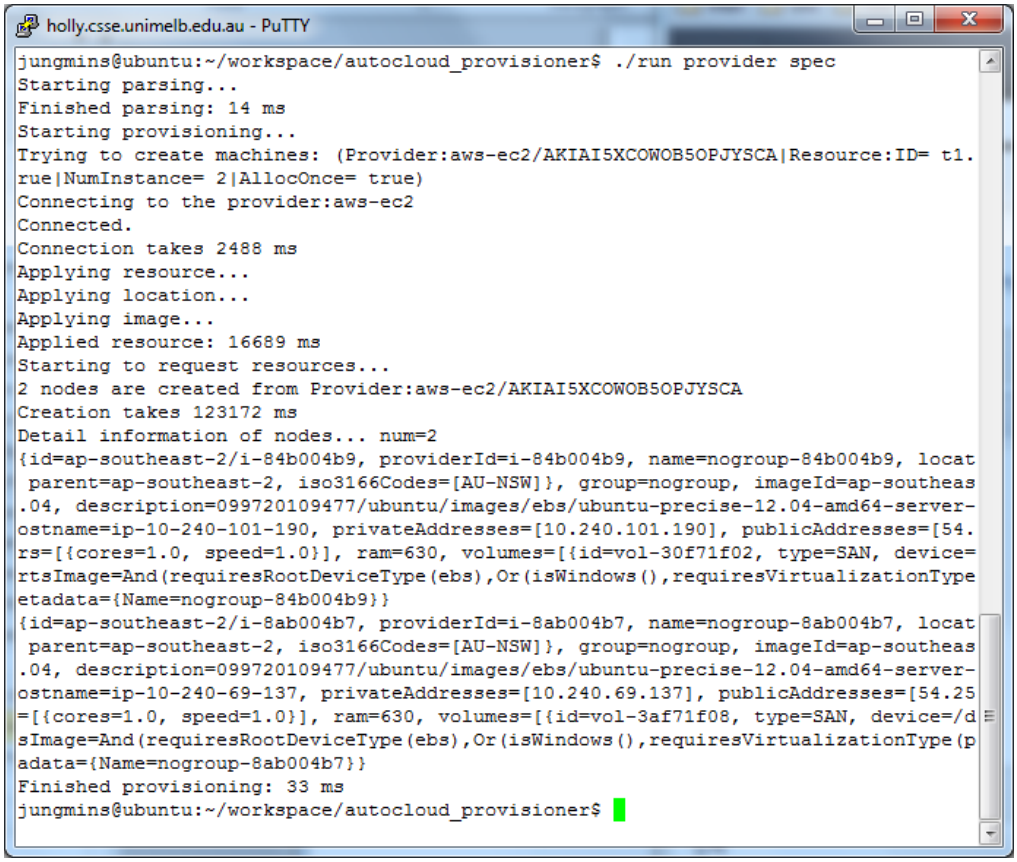

Figure 9 Screenshot of Resource Allocator

4 EVALUATION

We evaluate Resource Selection Decision Maker and Automated Resource Allocator separately, as two components are built independently. Decision Maker is verified by calculating the expected price and comparing it with the actual price. Resource Allocator is evaluated in terms of validation and quantitation.

4.1 Experiment: Resource Selection Decision Maker

Resource Selection Decision Maker should select a valid service and calculate the precise price based on the given requirements. For the evaluation of Decision Maker, the contents in Cloud Information Database are crucial to the result, as the quality of the result totally depends on the database entities. In this experiment, however, we make a small database including a little part of services for each provider. Thus, the result in this experiment may not cover all available services in the provider, but the result will be only applied to the services existing in our database. Figure 10 shows all database entities used for the experiment.

Resourceset	Servicetype	Operatingsystem	Geo
Amazon - m1.small (CPU=1.0, RAM=1700.0, HDD=160.0)	NoContract (0 days)	Linux	Sydney, Oceania
Amazon - m1.small (CPU=1.0, RAM=1700.0, HDD=160.0)	Contract (365 days)	Linux	Sydney, Oceania
Amazon - m1.small (CPU=1.0, RAM=1700.0, HDD=160.0)	Contract (1095 days)	Linux	Sydney, Oceania
Amazon - m1.medium (CPU=2.0, RAM=3750.0, HDD=410.0)	Contract (365 days)	Linux	Sydney, Oceania
Amazon - m1.medium (CPU=2.0, RAM=3750.0, HDD=410.0)	Contract (1095 days)	Linux	Sydney, Oceania
Amazon - m1.medium (CPU=2.0, RAM=3750.0, HDD=410.0)	NoContract (0 days)	Linux	Sydney, Oceania
Amazon - m1.small (CPU=1.0, RAM=1700.0, HDD=160.0)	NoContract (0 days)	Windows	Sydney, Oceania
Amazon - m1.small (CPU=1.0, RAM=1700.0, HDD=160.0)	Contract (365 days)	Windows	Sydney, Oceania
Amazon - m1.small (CPU=1.0, RAM=1700.0, HDD=160.0)	Contract (1095 days)	Windows	Sydney, Oceania
Amazon - m1.medium (CPU=2.0, RAM=3750.0, HDD=410.0)	Contract (365 days)	Windows	Sydney, Oceania
Amazon - m1.medium (CPU=2.0, RAM=3750.0, HDD=410.0)	Contract (1095 days)	Windows	Sydney, Oceania
Amazon - m1.medium (CPU=2.0, RAM=3750.0, HDD=410.0)	NoContract (0 days)	Windows	Sydney, Oceania
GoGrid - xsmall (CPU=0.5, RAM=500.0, HDD=25.0)	NoContract (0 days)	Linux	San Francisco, North America
GoGrid - xsmall (CPU=0.5, RAM=500.0, HDD=25.0)	NoContract (0 days)	Linux	Amsterdam, Europe
GoGrid - xsmall (CPU=0.5, RAM=500.0, HDD=25.0)	Contract (30 days)	Linux	Amsterdam, Europe
GoGrid - xsmall (CPU=0.5, RAM=500.0, HDD=25.0)	Contract (30 days)	Linux	San Francisco, North America
GoGrid - xsmall (CPU=0.5, RAM=500.0, HDD=25.0)	Contract (180 days)	Linux	San Francisco, North America
GoGrid - xsmall (CPU=0.5, RAM=500.0, HDD=25.0)	Contract (180 days)	Linux	Amsterdam, Europe
GoGrid - xsmall (CPU=0.5, RAM=500.0, HDD=25.0)	Contract (365 days)	Linux	Amsterdam, Europe
GoGrid - xsmall (CPU=0.5, RAM=500.0, HDD=25.0)	Contract (365 days)	Linux	San Francisco, North America
GoGrid - small (CPU=1.0, RAM=1000.0, HDD=50.0)	NoContract (0 days)	Linux	San Francisco, North America
GoGrid - small (CPU=1.0, RAM=1000.0, HDD=50.0)	Contract (30 days)	Linux	San Francisco, North America
GoGrid - small (CPU=1.0, RAM=1000.0, HDD=50.0)	Contract (30 days)	Linux	Amsterdam, Europe
GoGrid - small (CPU=1.0, RAM=1000.0, HDD=50.0)	Contract (180 days)	Linux	Amsterdam, Europe
GoGrid - small (CPU=1.0, RAM=1000.0, HDD=50.0)	Contract (180 days)	Linux	San Francisco, North America
GoGrid - small (CPU=1.0, RAM=1000.0, HDD=50.0)	Contract (365 days)	Linux	San Francisco, North America
GoGrid - small (CPU=1.0, RAM=1000.0, HDD=50.0)	Contract (365 days)	Linux	Amsterdam, Europe
GoGrid - small (CPU=1.0, RAM=1000.0, HDD=50.0)	NoContract (0 days)	Linux	Amsterdam, Europe
CloudSigma - (CPU=20.0, RAM=32000.0, HDD=1024.0)	Contract (30 days)	Linux	Zurich, Europe
CloudSigma - (CPU=20.0, RAM=32000.0, HDD=1024.0)	Contract (30 days)	Linux	Las Vagas, North America
CloudSigma - (CPU=20.0, RAM=32000.0, HDD=1024.0)	Contract (90 days)	Linux	Las Vagas, North America
CloudSigma - (CPU=20.0, RAM=32000.0, HDD=1024.0)	Contract (180 days)	Linux	Las Vagas, North America
CloudSigma - (CPU=20.0, RAM=32000.0, HDD=1024.0)	Contract (365 days)	Linux	Las Vagas, North America
CloudSigma - (CPU=20.0, RAM=32000.0, HDD=1024.0)	Contract (1095 days)	Linux	Las Vagas, North America

Figure 10 Database entities used for experiment

Table 15 User case for Resource Selection Decision Maker validation

Field	Value
Resource	CPU=1 cores RAM=512 MB Disk=20 GB
Operating System	Any
Location	Any
Contract Period	3 months
Allow Unreliable Services?	No
Usage Period	3 months

In order to validate the system, we simulate the user case (Table 15) and enter them as an input to the system. As a result, the system suggests eleven services described in Table 16.

As we can see from the result, the system suggests the services only with resources higher than the requirements, with any operating systems, in any locations and for any contract periods less than three months. Prices are calculated precisely and sorted in ascending order. In this case, CloudSigma has the lowest price as the amount of resource can be customized by a user, while other providers do not have fitted predefined resource set for the requirement.

Table 16 Results from Resource Selection Decision Maker

Rank	Provider	Resource	CPU (cores)	RAM (MB)	Disk (GB)	OS	Location	Contract period	Price
1	CloudSigma	custom	1	512	20	Linux	North America	3 m	$81.11
2	CloudSigma	custom	1	512	20	Linux	North America	1 m	$83.57
3	CloudSigma	custom	1	512	20	Linux	Europe	1 m	$91.80
4	GoGrid	small	1	1000	50	Linux	Europe	1 m	$108.75
5	GoGrid	small	1	1000	50	Linux	North America	1 m	$108.75
6	GoGrid	small	1	1000	50	Linux	North America	none	$172.80
7	Amazon	m1.small	1	1700	160	Linux	Oceania	none	$172.80
8	GoGrid	small	1	1000	50	Linux	Europe	none	$172.80
9	Amazon	m1.small	1	1700	160	Windows	Oceania	none	$248.40
10	Amazon	m1.medium	2	3750	410	Linux	Oceania	none	$345.60
11	Amazon	m1.medium	2	3750	410	Windows	Oceania	none	$496.80

4.2 Experiment: Automated Resource Allocator

Automated Resource Allocator is tested using the list of cloud configurations that is obtained from Decision Maker. Although the system supports every provider supported by Jclouds, we evaluate it only with Amazon's 't1.micro' instance, since it is free of charge. Other service types or providers will charge the cost once we create a virtual machine, so that they are exempted from the experiment.

The following cloud configurations list is used for the evaluation.

Table 17 Cloud configuration list used in the experiment

Provider ID	Resource set	Location	Image info	# of machines	Allocate in Single?
aws-ec2	t1.micro	ap-southeast-2	Ubuntu,12.04,64	2	True
aws-ec2	t1.micro	us-east-1	Ubuntu,12.04,64	0	True

With this list, the Allocator system tries to connect to Amazon-EC2 and to allocate two t1.micro instances with Ubuntu 12.04 images at Sydney datacentre (ap-southeast-2). If the trial is partially failed, which means that it creates only one machine, it destroys the partially created machine as the "Allocate single" field is set to true. If there is any failure, it attempts to the next line; the same set as the previous trial except for the U.S. East (us-east-1) location instead of Sydney.

After the system executes, it succeeds to allocate two machines in Sydney, and we can find the successfully created machines on the control panel of Amazon's website. In addition, we measure time consumed for allocating the machine. In the process of Jclouds, it waits until all allocation processes are done before it returns the created nodes to our system.

Table 18 Performance of operations executed during experiments

Operation	Component	Time(ms)
1.Parsing the cloud configuration list	Resource Allocator	14
2.Connection to the provider	Jclouds/Provider	2,488
3.Applying resource, location and image information to the provider	Jclouds/Provider	16,689
4.Creation of two virtual machines	Jclouds/Provider	123,172
5.Controlling allocation process	Resource Allocator	33
Total	Resource Allocator	47
	Jclouds/Provider	142,349

In the above result, it takes about 142 seconds to connect to the provider, apply the cloud configuration and create two virtual machines. However, our system responds immediately as it takes only 47 msec for parsing and controlling the process.

5 CONCLUSION AND FUTURE WORKS

This project aims on finding the best cloud provider for the given user's requirement and allocating the resources automatically. As there are lots of different types of cloud services in various providers, searching for the best option is annoying and time-consuming job for system administrators. The system suggested in the project helps them to automate and lighten those efforts.

Above all, we reviewed the current practices of cloud computing industry, described the architecture of our system and evaluated the implemented prototype. We could find distinctive pricing policies upon the providers and common factors for the price leading the requirements of the user. System design we proposed was composed with two independent components in charge of decision making and resource allocation, and two databases helping each components. Finally, we evaluated overall accuracy and performance of the system with a user case and a cloud configuration list. The evaluation showed that our approach suggested in this system worked properly in the given situation.

For the future works, the automatic data grabber for Cloud Information Database is beneficial to the system, as it can automatically parse the provider's website and update the information database. It can give the consistency between our system and actual service policies of the providers without any human labours. Also, Resource Selection Decision Maker can be improved with better algorithm to select the provider, or expanded to intelligently separate a whole resource set into the number of smaller resources to reduce overall cost. Other aspects, such as ping time measured from the user's host to each provider server, can be added to the decision process to find better provider with less latency. As the system components are independent to each other and pluggable with other programs, improvement will be easily deployed to the system.

REFERENCES

[1] P. Mell and T. Grance, "The NIST Definition of Cloud," National Institute of Standards and Technology, Gaithersburg, MD, 2011.

[2] "Amazon Elastic Compute Cloud (Amazon EC2)," Amazon, [Online]. Available: http://aws.amazon.com/ec2/. [Accessed 10 March 2013].

[3] "Windows Azure," Microsoft, [Online]. Available: http://www.windowsazure.com.

[4] "Google App Engine," Google, [Online]. Available: https://developers.google.com/appengine/.

[5] "HP Cloud," HP, [Online]. Available: https://www.hpcloud.com/.

[6] "GoGrid Cloud Servers," GoGrid, [Online]. Available: http://www.gogrid.com/products/infrastructure-cloud-servers. [Accessed 10 March 2013].

[7] "CloudSigma Cloud Hosting Price Schedule," CloudSigma, [Online]. Available: http://www.cloudsigma.com/en/pricing/price-schedules. [Accessed 1 May 2013].

[8] N. Grozev and R. Buyya, "Inter-Cloud Architectures and Application Brokering: Taxonomy and Survey," *Software: Practice and Experience*, 2013.

[9] M. Zhang, R. Ranjan, S. Nepal, M. Menzel and A. Haller, "A declarative recommender system for cloud infrastructure services selection," in *Proceedings of the 9th international conference on Economics of Grids, Clouds, Systems, and Services*, Berlin, Germany, 2012.

[10] R. Buyya, R. Ranjan and R. N. Calheiros, "InterCloud: Utility-Oriented Federation of Cloud Computing Environments for Scaling of Application Services," in *Proceedings of the 10th International Conference on Algorithms and Architectures for Parallel Processing*, South Korea, 2010.

[11] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich and F. Galan, "The Reservoir model and architecture for open federated cloud computing," *IBM Journal of Research and Development*, vol. 53, no. 4, pp. 4:1-4:11, 2009.

[12] B. Rochwerger, D. Breitgand, A. Epstein, D. Hadas, I. Loy, K. Nagin, J. Tordsson, C. Ragusa, M. Villari, S. Clayman, E. Levy, A. Maraschini, P. Massonet, H. Muñoz and G. Tofetti, "Reservoir - When One Cloud Is Not Enough," *Computer* , vol. 44, no. 3, pp. 44-51, 2011.

[13] A. J. Ferrer, F. Hernández, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, W. Ziegler, T. Dimitrakos, S. K. Nair, G. Kousiouris, K. Konstanteli, T. Varvarigou, B. Hudzia, A. Kipp, S. Wesner , M. Corrales , N. Forgó , T. Sharif and C. Sheridan, "OPTIMIS: A holistic approach to cloud service provisioning," *Future Generation*

Computer Systems, vol. 28, no. 1, pp. 66-77, 2012.

[14] "Product Overview," RightScale, [Online]. Available: https://www.rightscale.com/products/index.php. [Accessed 15 March 2013].

[15] "jClouds," jClouds, [Online]. Available: http://www.jclouds.org.

[16] "Apache LibCloud," Apache, [Online]. Available: http://libcloud.apache.org/.

[17] "DeltaCloud API," Apache, [Online]. Available: http://deltacloud.apache.org/.

[18] "SimpleCloud," Zend Technologies, [Online]. Available: http://www.simplecloud.org/.

[19] "Ruby on Rails," Ruby on Rails, [Online]. Available: http://rubyonrails.org/. [Accessed 1 May 2013].

[20] "SQLite," SQLite, [Online]. Available: http://www.sqlite.org/. [Accessed 1 May 2013].

[21] "Amazon Web Services," Amazon, [Online]. Available: http://aws.amazon.com. [Accessed 10 March 2013].