Decision support for different migration types of applications to the Cloud

Mingzhu Xiu

Course of Study: Computer Science

Examiner: Prof. Dr. Frank Leymann

Supervisor: Dr. Vasilios Andrikopoulos

Commenced: March 25, 2013

Completed: September 24, 2013

CR-Classification: D.2.1, D.2.9, H.3.3, H.5.2
Abstract

Cloud computing brings many benefits to the enterprises who decide to migrate their applications to the Cloud partially or completely. This thesis focuses on helping users finding the most cost-efficient between different Cloud offerings which have similar features. Data from several Cloud providers are collected and organized in order to implement a Cloud provider knowledge base exposed as a set of web services. These services are validated against other similar systems and exposed as RESTful APIs. A decision support system is built in this work based on these APIs and evaluated by an existing use case with different migration plans.
Contents

1 Introduction 1
   1.1 Motivation .......................................................... 1
   1.2 Problem Definition and Goal ........................................ 2
   1.3 Outline .................................................................. 2

2 Background 5
   2.1 Fundamentals .......................................................... 5
   2.2 Migration-related Issues and Concerns .............................. 6
   2.3 Decision Support Systems for Cloud Migration ................. 8
   2.4 Cloud Services Cost Calculators .................................... 10
   2.5 Migration Decision Support System ............................... 11
   2.6 Summary ................................................................ 12

3 Specification & Design 13
   3.1 Requirements ......................................................... 13
   3.2 System Specification ................................................ 14
   3.3 Cloud Provider Knowledge Base .................................... 17
   3.4 Decision Support Services .......................................... 19
       3.4.1 Candidate Search Service ..................................... 19
       3.4.2 Cost Calculator Service ....................................... 20
   3.5 Decision Support System ............................................ 25
   3.6 Summary ............................................................... 36

4 Implementation 37
   4.1 Cloud Provider Knowledge Base of Nefolog System .......... 37
   4.2 Services of Nefolog System Implementation ..................... 39
   4.3 Decision Support Services of Nefolog System ................. 43
       4.3.1 Candidate Search Web Service ............................. 43
       4.3.2 Cost Calculator Web Service ................................. 47
   4.4 Decision Support System MiDSuS ................................ 51
   4.5 User Guide ................................................................ 59
       4.5.1 Selection of Migration Type ................................. 59
       4.5.2 Requirement Collection of Candidate Offerings ......... 59
       4.5.3 Selection of Candidate Offerings ......................... 60
       4.5.4 Collection Requirements of Cost Calculator ............ 60
       4.5.5 Checking Results of Cost Calculator and Ranking ...... 61

5 Evaluation 63
   5.1 Nefolog Validation .................................................. 63
   5.2 MiDSuS Evaluation ................................................ 68
6 Conclusions
  6.1 Summary ......................................................... 71
  6.2 Future Work ..................................................... 72

Appendix
  A.1 Use Case of Applications on Cloud .............................. 75

Bibliography ........................................................... 81
List of Figures

1.1 Structure of Thesis .................................................. 3
2.1 Conceptual Model of Decision Support System for Cloud Migration [1] .................................................. 8
2.2 Cloud Adoption Conceptual Framework [2] .................................. 9
2.3 Overview of CloudGenius Approach [3] .................................. 10
3.1 Cloud Servers from Provider Rackspace Pricing Details [4] .................. 14
3.2 Envisioned System Functionalities ........................................ 15
3.3 Modules of new System .................................................. 16
3.4 Entity-Relation Diagram of MDSS [5] .................................... 17
3.5 Entity-Relation Diagram .................................................. 18
3.6 Details of Type I ...................................................... 26
3.7 Candidate Requirements Details of Type I ................................ 27
3.8 Result of Candidate Search Details of Type I ............................ 28
3.9 Cost Requirements Details of Type I .................................... 29
3.10 Results of Type I ...................................................... 29
3.11 Details of Type II ....................................................... 30
3.12 Candidate Requirements Details of Type II ............................. 31
3.13 Result of Candidate Search Details of Type II ........................... 32
3.14 Cost Requirements Details of Type II .................................. 33
3.15 Results of Type II ....................................................... 34
3.16 Details of Type III ...................................................... 35
3.17 Details of Type IV ...................................................... 35
4.1 Data model .......................................................... 38
4.2 UML Classes Diagram of Nefolog ....................................... 41
4.3 User Interface of MiDSuS: Type I ....................................... 52
4.4 User Interface of MiDSuS: Type II ....................................... 52
4.5 Subdivision of MiDSuS .................................................. 53
4.6 User Interface: First Step of Type I ..................................... 55
4.7 User Interface: Second Step of Type I: Candidate Search Step ........... 56
4.8 User Interface: Third Step of Type I .................................... 57
4.9 User Interface: Fourth Step of Type I ................................... 58
4.10 User Interface: Fourth Step of Type I with Details ....................... 59
List of Tables

2.1 Overview of Cost Calculators ........................................ 11

4.1 Overview of Service Types between MDSS and new System ........ 39
4.2 Overview of Languages and Frameworks to Support Restful Web Services

4.3 Resource URIs supported by Nefolog ................................ 40
4.4 Decision Support Services URIs supported by Nefolog ......... 43
4.5 Provenances of each Application Component in MiDSuS ........ 54

5.1 Comparison of Candidate Offerings Search ......................... 64
5.2 Static Rate Test Comparing with PlanForCloud ..................... 65
5.3 Dynamic Rate Test 1 Comparing with PlanForCloud ............... 66
5.4 Dynamic Rate Test 2 Comparing with PlanForCloud ............... 67
5.5 Same Application with different Migration Types ................. 68

A.1 Application Migration Use Cases ................................... 75
A.2 Mapping of Application Migration Use Cases to Migration Types . 76
List of Algorithms

1. Candidate Search ......................................................... 20
2. Cost Calculator .......................................................... 22
Listings

4.1 SQL Query Expression for Candidate Search Service . . . . . . . . . . . . 44
4.2 Candidate Search . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 45
4.3 Default initial Values . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 47
4.4 Dynamic Rate Situation . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48
1 Introduction

In this chapter, the motivation of this thesis is explained on both the business market side and the development of IT technologies side. Based on technical practice, this motivation is defined as a specific problem. In order to solve this problem, a feasible solution is executed and described in the rest of this thesis.

1.1 Motivation

An enterprise which needs some support from IT technologies would face either a shortage of resources or an over-investment problem during the enterprise’s growth and development. From an economics perspective, enterprises desire to gain a maximum benefit with least resources. In the past, people chose to solve those problems directly. They bought advanced equipment to suit the business growth and employed more IT staff for maintenance and update of this equipment locally. But for small or medium size enterprises they may not have so many funds to achieve this update. They are also less able to bear the risk of more investment. In other words, if the benefit did not reach the expected goal after the investment, they could not afford a big loss. With the help of Cloud computing, such enterprises do not worry about these apprehensions any more. They have a new choice to solve their problems. The advantages of Cloud computing are very obvious. There are a number of providers. Many Cloud services are pay-as-you-use. Even if their desired services have many restrictions, they can also find an appropriate alternative offering. Some large enterprises have even a collaborative relationship with Cloud services providers. For example, currently the worldwide largest software enterprise SAP has a collaborative alliance with Amazon Web Services (AWS). The SAP customers can deploy their SAP solutions on Amazon EC2 instances which are already certified by SAP [6].

During the development of Cloud computing market, more and more applications are migrated to the Cloud. There are already many applications which are even developed for the Cloud. This type of development solves some problems but brings also some new problems.

Because of the different kind of Cloud services providers and different features of one Cloud offering, there are various kinds of Cloud offerings with different pricings. How to obtain a suitable offering is very important for consumers who want to use a Cloud service, or who have already chosen a Cloud service but want to find a better one. They need a system to provide candidate offerings which are selected according to available information from the providers. If a Cloud service can achieve all the user’s requirements with the lowest cost, it should be obviously considered as one of the best candidate offerings. If additionally this migration type or migration process undertakes the minimum
1 Introduction

Risk, it is also a good candidate offering. In [7] four migration types are defined. Type I and Type II are partial migration while Type III and Type IV are completely migration. In addition, Type I and Type III are migrated based on application components but Type II and Type IV are based on application layer. The specific definitions of each type are explained in Chapter 2 Section 2.1.

1.2 Problem Definition and Goal

In this thesis, a decision support system is built to help users to find an efficient offering for their needs. In this system, the decision which Cloud offering(s) to use is based on a comparison between the cost of suitable offerings. The system collects information from two sides to support this comparison. One side is the users’ side which means all requirements from users; the other side is the database side that summarizes all the information from Cloud services providers.

From the users’ requirements, a migration type as defined in Chapter 2 (Section 2.1) can be decided. With different migration type the selection result is different and it undertakes different risk. In the application use case A.1 for example, the European Space Agency process astronomic data which are summarized from one billion stars in our galaxy [8] and migrated all their DBMS to the Cloud with VMs using Migration Type III. Users can also determine which provider he trusts more. Furthermore, there are some restrictions about the migrated component itself. More specifically, the user can restrict the characteristics of the Cloud offering which he wants to rent and the usage time or amount of this offering. These are the most basic conditions for the further comparison.

From the providers’ information, the features and pricings of each offering can be summarized together. The same offering with different features is divided into different configurations. These features are for example the number of CPU cores, CPU speed, storage capacities etc. Offerings are represented into cost calculation formulas by different parameters, although they may provide similar services. Simple Storage Service (S3) from AWS, for example, has a cost calculation formula with 4 parameters. They are: storage GB, put, copy, post, or list requests, glacier archive and restore requests, and get and all other requests. Cloud Storage which provides the same service as S3 but is from Google has a formula with 2 parameters. Except from storage GB, it has a transactions parameter to represent the pricing of this offering. The location of the data-center is also a related parameter to separate offering into different pricing.

After that, comparisons are done between each candidate offering. According to users’ desires, offerings with the lowest costs are identified for the actual migration of their application to the Cloud.

1.3 Outline

This thesis consists of 6 chapters. Figure L.1 outlines the core chapters of this work. After introducing the background of this thesis (Chapter 2), the system specification is
abstracted from the system requirements in Chapter 3. In addition, according to this specification, a system is described to support all the requirements. In the system design section, a Cloud provider knowledge base is discussed, in order to support all the decision support services. These services, exposed as APIs, can be used in a decision support system. The implementation of this decision support system is explained in Chapter 4. The Nefolog system is presented first. It contains all the decision support services which are based on the Cloud provider knowledge base. Furthermore, a decision support system called MiDSuS (Migration Decision Support System) is presented based on the API supported by the Nefolog system. After that, the correctness and efficiency of these two systems is checked in Chapter 5. The validation of Nefolog system is compared with the results from PlanForCloud [9]. The evaluation of MiDSuS system is based on a case study related to the migration of an existing application to the Cloud.
2 Background

In this chapter, an analysis of existing works is performed across several dimensions. The result of this analysis shows that providing a decision support system for the Cloud migration is essential.

2.1 Fundamentals

The definition of Cloud computing is a presentation of computing as a service for delivering ubiquitous, convenient and on-demand resource to customers and this service is different to the traditional production definition \[10\]. Customers do not have to know the design details of the technology infrastructure in the Cloud or need to have an understanding of the related knowledges. This service is based on the network to support dynamic, scalable and virtualized resource. This new technology brings new opportunities and challenges in the IT field.

Current Cloud services are classified by three main service models, software as a service (SaaS), platform as a service (PaaS) and infrastructure as a service (IaaS) \[11\]. In the SaaS model, this application software is referred as on-demand software. User does not need to worry about the installation or management of the application. For example Google Apps, user uses it through some clients. In PaaS model, Cloud providers deliver platforms to application developers who want to develop their solutions on a Cloud platform which normally contains operation system, database, web server, programming language execution environment. Google App Engine (GAE) is a typical Cloud service in PaaS model. In the most basic IaaS Model, providers provide some fundamental resources with virtual machines which can help customers to deploy their own systems on Cloud like AWS EC2.

After classifying these offerings by service models, application can be distributed by migration types. An application can be distributed physically into different components. Some components can composite an application layer with logical functionality. In order to recognize whether the user distributes his application physically or logically and whether the user attempts to migrate his application partially or completely, four migration types have been proposed in \[7\].

- **Type I**
  
  Replace one or more components of the application to the Cloud. It is the least invasive and risk migration type. Some measures like configurations, rewiring and adaptation should be done to fix the problems which may happen after replacement.
• Type II
Migrate a part of the application functionality like a set of architectural components from one application layer to the Cloud. These components should have an interconnection-relationship.

• Type III
Migrate the whole software stack of the application to the Cloud. Normally it is based on virtualization of application and achievable by a number of VMs.

• Type IV
Migrate the application completely to the Cloud. It means that all the data layer and business logic layer are re-engineered as a composition of Cloud services.

The applications use cases in Appendix A.1 show how these applications are organized before and after migrating to the Cloud by different migration types in Table A.2. In addition, these applications are marked by "Y" to represent which Cloud offering(s) they have chosen already and by "A" to signify which Cloud offering(s) they could use as an alternative, in Table A.1. These 4 types are enough to cover all the migration situations in Appendix A.1. The same application component can be migrated to the Cloud with different offerings in different migration types. For example, the payroll processing has already been migrated to the Cloud with VMs. One of the migrated components is DBMS. For the DBMS, user can also choose a database Cloud service instead of VM. It should be noted that different migration types will bring different risks and difficulties to the migration.

At present, it seems like that Cloud computing brings us so many benefits such as: it is not so expensive; it reduces the IT support team; it allows management to scale his enterprise with demand etc. How can we know whether we really get benefits from it and how many they are? This problem actually is to estimate the total cost of one Cloud project which is assumed that the components would be migrated to the Cloud with a set of suitable offerings. So far, the cost of an offering is considered as the most important criterion in this decision making process. Actually, there are also other decision factors, for example, same migrated application components with different Cloud providers can lead to different benefit and risk; security and compatibility of these Cloud services can also be the criteria for making process; trust of user for one provider may also change the decision etc.

2.2 Migration-related Issues and Concerns

From the previous discussion, if user wants to find an appropriate Cloud offering, a set of decisions should be made at first. Furthermore, the relationships and influences between each decision cannot be ignored either. These decisions call for a system to find a trade-off between cost estimation and performance prediction. In this section, some
researches and works are studied to discuss which factors play an important role in this decision making process.

In the research by Andrikopoulos et al. [1], a conceptual model of decision support system is proposed (Figure 2.1) and described by two types of concepts, decision and task. The decision concept consists of 4 actions (distribute application, select service provider/offering, define multi-tenancy requirements and define elasticity strategy). The transparent arrows in the figure mean that there are influences between each decisions. The task concept consists of 7 activities (work load profiling, compliance assurance, identification of security concerns, identification of acceptable QoS levels, performance prediction, cost analysis and effort estimation). These tasks support these decisions while these decisions depend also on these 7 tasks. According to the previous section, an application can be distributed logically and physically by different migration types [7]. Different migration types bring different risks and troubles in follow-up work to user. It is necessary to choose an appropriate way with the minimum risk. More and more commentators accept multi-tenancy as a feature of Cloud computing. It means that providers let multiple tenants to share a single version of software in order to reduce the total costs of ownership (TCO). The elasticity strategy can be considered from horizontal (more application instances) and vertical aspects (more computational resources). In the task concept part, it can be seen in the figure that the application distribution can decide performance prediction, effort estimation and cost analysis tasks in the meantime these tasks can also in turn send feedbacks to application distribution decision. The rest three decisions have similar situations as application distribution decision. It means that these tasks are determined by decisions at the same time influence the decisions.

Another work presented in [12] discusses 4 sociable factors which should be treated with carefully in decision making process. They are trust reputation management, risk assessment, green assessment and cost and economical sustainability. Trust can be known by reputation mechanisms [12]. It is an invisible presence but can be made aware from some experiences in the past. If a Cloud services provider is in a blacklist which means that he is a dishonest provider, user should be suggested by decision support system that it is not safe to use this provider’s services. Risk is suggested to be considered not traditionally but with the cost of reconfiguration and migration, the reliability of Cloud services and also some other factors together [12]. As well as the risk which may happen in application distribution, there are so many other unforeseen incidents. In the research by Khajeh-Hosseini et al. [13], such incidents are proposed, like: IT staff has low morale and sometimes is very anxious while department may make some changes in his work or reduce staff number during migration; a major service interruption may lead to an extensive outages, unavailability of services or data loss etc. All of these risks are collected from five perspectives, security, technical, organizational, legal and financial. Green assessment is a very common viewpoint in our daily life but it is proposed in ICT industry until recently. From the environmental protection, green cloud computing means that providers try to achieve the desired QoS with as low energy consumption as possible.
2 Background

2.3 Decision Support Systems for Cloud Migration

In this section some existing decision support systems are introduced. Each system is implemented based on cost estimation. Several other factors are also considered.

Khajeh-Hosseini et al. [2] developed the Cloud Adoption Toolkit to support decision making. This toolkit contains a collection of tools which focus on both cost calculation and socio-technique. The conceptual framework of this toolkit is shown in Figure 2.2. Decision maker starts from technology suitability. This technology suitability analysis is built by a checklist in the table of Figure 2.2, which contains 12 questions in 8 technology characteristics. If it is found suitable for the user’s work, the next step is to analyze either cost of Cloud services in a public Cloud or cost of energy in his private Cloud. The cost is calculated with the help of a UML deployment diagram which models the deployment of his system on Cloud. At the same time, the stakeholder impact analysis is also done to evaluate the socio-political benefits and risks. The weighted average of benefits and risks which are summarized from some case studies in the research by Khajeh-Hosseini et al. [13] can be calculated and are shown in two radar graphs in Figure 2.2. After these two analyses, if his work is found viable, it goes to the responsibility modelling. This modelling is to check the operational viability of his work. Then the last step is to implement his work in the Cloud.

The second decision support system is called CloudGenius which presented in the work...
by Menzel et al. [3]. In Figure 2.3, the approach is illustrated. It is a continual and evolutionary migration process. First of all, the decision between Cloud and no Cloud is achieved by a \((MC^2)^2\) framework whose structure is on the top right of this figure. If Cloud infrastructure is viable, this system goes to the core part called CloudGenius migration process which is shown in bottom right of this figure. In the beginning of this migration process, the goal and preferences of the user should be set up. After that, VM image evaluation and Cloud services evaluation are implemented in parallel. These two evaluation methods are implemented by the \((MC^2)^2\) framework which uses Analytic Hierarchy Process (AHP) reducing decision modelling effort instead of Analytic Network Process (ANP). Then the user receives the results which contain some suggestions of VM image and infrastructure. A best combination between Cloud image and Cloud service is selected to achieve this migration. The final step is to make a loop of this making process with some other alternative requirements until the user is satisfied with the solution. This migration process will stop if no more Cloud application is needed. It means that this application is migrated to the Cloud successfully.
2.4 Cloud Services Cost Calculators

The information about one Cloud service contains the service features, usage conditions, pricing and some other related works. In order to help a user to have a clear understanding of his project or his migrated application cost, it is easy to find a cost calculator on some providers’ official website. There are also several other developers who focus on Cloud services cost calculator of several different Cloud services providers but they do not provide any Cloud services themselves.

Table 2.1 is used to show a general introduction of some cost calculators. The first three rows contain cost calculators for Cloud services which are provided by the Cloud provider itself. All of them provide the calculator which can make a hybrid calculation among their offerings and the combination of each offering is selected by user. The last two websites can help the user to calculate his migrated application with some available Cloud providers. PlanForCloud supports a relative all-round system to help user to
## 2 Background

Table 2.1: Overview of Cost Calculators

<table>
<thead>
<tr>
<th>Cloud Services Cost Calculator</th>
<th>Service(s)</th>
<th>Cloud Provider(s) Considered</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows Azure Pricing Calculator [15]</td>
<td>web sites, virtual machines, mobile services, cloud services, data management</td>
<td>Windows Azure</td>
</tr>
<tr>
<td>Rackspace Cloud Calculator [16]</td>
<td>Cloud Servers, Cloud Files, Load Balancers</td>
<td>Rackspace</td>
</tr>
</tbody>
</table>

calculate his whole project. In addition, it can calculate the trend of a project cost in a certain term with some defined usage patterns. It contains many different geographical areas of server data centers. It includes more services than Cloudorado which has only IaaS. Cloudorado on the other hand provides a calculator with a lot of Cloud providers. It offers simple mode and advanced mode. In advanced mode, user can also choose more than one server to migrate his application at the same time the data transfer between each server should be considered.

### 2.5 Migration Decision Support System

The current work is a continuation and is based on MDSS which is developed by Song [5] in early 2013. In this section, the work of MDSS is explained in detail.

MDSS consists of a database, data handler and user interface. The database called Cloud provider knowledge base contains 2 providers, Google and Windows Azure and works on SQL Server 2008. Each provider contains a set of offerings and according to various performance characteristics with different values an offering can be divided into several configurations. In the database, 16 features are summarized for each offering. Pricing of
each offering is represented by a formula with multi-parameters. These pricing formulas are collected by different locations and different usage amounts. After explaining the information which is collected from the providers, the developer defined also 5 service types to organize these offerings from system side.

The data handler contains two main functions, offerings matcher and costs calculator. The offerings matcher function is implemented by a comparison between user’s requirements and the data in the database. These user’s requirements should include: which service type is migrated to the Cloud; which offering user prefers to choose; which performances the Cloud offering should have according to user’s demands. The costs calculator is achieved by calculating the pricing formula with certain values which are assigned by user. A user can also choose his desired data center location and usage pattern with a certain rate in a fixed period of time.

All these requirements both for offerings matcher and for costs calculator are given on the first page of MDSS UI. The results are listed in the second page with configurations, locations, details, providers, costs, sums and parameters. In the end of each row in this list, there is a button called “Info”. With this button, user can get more information concerning this configuration’s performance characteristics and a graph of cost in each month. In the bottom of this page, there’s a drop down list to select a parameter to rank. The Ranking result is shown in the third page.

### 2.6 Summary

Knowledge of service models and migration types are set up as foundation in this work. As studying some researches about decision support for Cloud migration, a number of factors are introduced and should be considered in making process. Furthermore, two existing decision support system are illustrated step by step. If all these factors are considered, the decision making process may be separated by amount of evaluations and comparisons. A mature decision support system should be generated by a long-term data collection and based on a lot of migration practices. An attempt to build a decision support system is done by Song. In his work, he introduced a system called MDSS which achieves to a trade-off between cost and performances. The composition and operation of MDSS are summarized in this chapter.
3 Specification & Design

This chapter focuses on extracting requirements and presenting the specification and design of a decision support system for Cloud migration. To complete all the requirements a decision support system is established which is based on some RESTful APIs. These APIs are working with a database named Cloud providers knowledge base. This knowledge base summaries information of several Cloud services providers. The information is about features and pricings of each offering.

3.1 Requirements

Cloud service is a better solution for a private consumer or an enterprise to expand the service coverage area or to reduce the IT support team members or the calculation time of big data processing. It is reliable, flexible and scalable. Besides these characteristics, some of these Cloud services are pay-as-you-use. It is an efficient way to reduce the cost. In today’s market, there are a few providers which provide a variety of Cloud services. It means that there are many different available offerings for consumer to achieve his strict requirements about the cost, the provider or even the data center from which to be served.

In Figure 3.1, the shown provider is Rackspace and this Cloud offering is Cloud Servers. It supports VMs with different performance on different operating system with different database manager. In addition, the pricing for each configuration is different. Higher performance has a little more expensive price than the lower one. For one single provider, it has already many different configurations. For many providers, the possible options are larger. Consumer spends too much time on choosing a practical and cost-effective Cloud offering among them. Supporting the user in this task is perceived as this work’s goal. A decision support system is built to help consumer to find a set of suitable offerings quickly and directly.

As a whole system, a friendly user interface should be developed. It will support a concrete and intuitive interaction for user. A comparison between the initial application and migrated application is necessary to help user to choose an appropriate migration type. The migration types are already explained in Section 2.1. Choosing the appropriate migration type can reduce the risk during migrating to the Cloud and save investments.

It should be considered as the first step for an application which would like to be migrated to the Cloud. Second step, a description of application will be asked for. Actually, the definition of service type is described more obviously and directly by the application components and application layers. After that it’s time for the initial requirements part. The requirements may contain which provider user wants to choose, which kind
of performances he hopes the Cloud service has, how long he wants to use this service, etc.

All these requirements will be collected and transferred to the data processing part. This system will search for several candidate offerings which could reach or even exceed the demands of user in all offerings. For each candidate offering, the service cost and data transfer between this offering and local or another offering will be calculated with the help of cost formulas with multiple parameters whose values are already given as initial conditions. The total cost is a sum of the usage amount of the service, cost of data transfer and the upfront of this service. User can also know the trends of the total cost by giving different usage patterns in order to decide which offering is worthy to utilize during a long-term plan.

### 3.2 System Specification

The envisioned functionalities of the system are illustrated using a Use Case diagram in Figure 3.2. It is abstracted from the requirements in Section 3.1. All the requirements of user are separated into 3 parts. The first is to Select Application Description. It is used to determine the appropriate migration types. The second is to Enter Candidate Requirements. It provides finer initial values for candidate offerings search like provider name and service type which is based on the migrated components. Using these requirements and necessary data from the database the candidate offerings can be found. According to the user’s desire a number of candidate offerings can be selected to get concrete evaluations of these offerings’ costs. The third is Enter Cost Requirements. For
different configurations the calculation variables are different. These optional evaluation variables are collected by a service in backend. With the help of these requirements and cost formulas from database the costs and cost trends can be calculated. All these results can be returned and visualized for the user. There are also several sorts of Rankings to help the user to have a better view of the costs, update dates or data centers comparisons.

Extending the work in [5], this new system is composed of two main parts. One is called Cloud Services as back-end of the system while the other is called Cloud Migration as front-end of the system. Cloud Services part is built up by web services and database. It implements the offerings matcher and costs calculator functionality. Cloud Migration part is a web application part which is based on the Cloud Services part. It contains UI components and some logic behind it which is used to transfer data between front-end and back-end.

As it can be seen in Figure 3.3, the consumer can give his requirements in the web application which is called Decision Support System in Figure 3.3. In the Decision Support System, there is a description of the consumer’s application and the components which are meant to be migrated on the Cloud. Consumers can choose one Cloud services provider or all of them. After some necessary values are ready for the data processing part, it’s time to begin candidate offerings search then the system works about offerings’ cost calculation. All these requirements are considered as requests to web services. Many web services are built to support this system. One of them is to look for the candidate offerings. Another one is to calculate the total cost of offerings and the trends of this cost in the each following month. This search and calculation part is called Decision Support Services in Figure 3.3. All the necessary data are stored and retrieved from the Cloud Provider Knowledge Base. After that the responses of this service are sent back to the consumer and displayed through the web application.
The knowledge base is expanded based on the previous knowledge base in MDSS while the functions of data handler are transplanted as services in the new system. The similar parts will therefore not be described again; only the different part will be given an account here. More specifically:

- **Interaction**
The UI of this work is different from MDSS. It will support consumer to complete his selection from 4 migration types not only from service types. In addition, for each type all the necessary initial conditions will be given in a table, in order to help consumer to clearly see which components his application contains and which one or ones he wants to migrate to the Cloud. Instead of choosing the service types in MDSS, this system separates all the service types into application components and collects them into different service types for Type I & III and into different application layer for Type II & IV. He does not need to choose which Cloud offering he wants to use but to choose which Cloud provider. This way supports consumer to get more apposite available offerings than in MDSS.

- **Web Service instead of Function**
All the functions in MDSS will be transplanted to the new system as restful web services. A platform-independent and program language-independent system is a goal for this work. All the services can be put in a WAR file in the end and run on any platform. The 2 main functions in MDSS, offerings matcher and costs calculator, will be changed into candidate search service and cost calculator service, respectively. Some algorithms will be kept while others will be overridden. But the general idea is identical.

- **Data**
There are more providers covered than MDSS, 6 (Google, Windows Azure, Amazon Web Services, Hp Cloud, Rackspace and Flexiscale) instead of 2 (Google and Windows Azure). More providers mean more service types of an application and more factors of calculating a total cost and even more process time of one service. A refinement and reworking of service types must be done to make a better organization among all the offerings in the knowledge base. Cost of data transfer between two offerings or between offering and local will be considered as one factor. In addition, upfront costs of one Cloud service cannot be ignored. The definition
of location in MDSS will be changed to geographical area and a definition of zone will be added into the Knowledge Base in order to make a better partition for different provider’s Cloud service which is from different data-center. In the data collection, the pricing of the Cloud services from the provider Flexiscale is different. During the processing of costs calculator, the provider Flexiscale should be separately treated with.

### 3.3 Cloud Provider Knowledge Base

This knowledge base refines and extends the equivalent base from MDSS. Figure 3.4 shows the Entity-Relation diagram of the MDSS’s database while Figure 3.5 shows the new Entity-Relation diagram. All the MDSS’s basic entities are information included in the new ER diagram. At the same time, some more information has been added to supply the users with more possibilities when they intend to utilize the Cloud services.

In Figure 3.4 there are six main entities and relationships among them. The providers’ information is abstracted as the Provider entity and it has a one-to-many relationship with the Offering entity which is also classified by the Service Type entity. One offering

http://www.flexiscale.com/products/flexiscale/
may supply more configurations that are sorted by different performance. The cost of each configuration is calculated by some variables according to different location and usage amount.

It can be seen in Figure 3.5, the six main entities are kept. The Cost entity is divided into three parts. The reason is that there are three factors which are not independent on each other but can affect the final cost. One of these factors is the upfront cost of using the service. It is only required by Amazon Web Service (AWS) offerings. MDSS does not include the AWS provider, so it does not have this entity. Another factor is the price of data transfer. MDSS also considers this cost but only as a single service type. In this work, the data transfer is considered as a part of the cost calculation, because there must be some data which transfer in or out from a cloud service. The last part is the pricing of the Cloud service as provided in MDSS. These three factors have a one-to-many relationship with the Cost entity. To make a final cost estimate, a sum of these three parts must be done.

There's a special provider in this new system, Flexiscale. The cost calculation for this provider is related to the number of units which the configuration utilizes and not to a
fixed price. It has a one-to-one relationship with the Configuration entity and can be represented by a one-to-many relationship with the Variable entity.

The location of the new system is specific to each geographical area that related to all the providers. The areas are classified into several zones. The two entities have a one-to-many relationship.

### 3.4 Decision Support Services

To achieve these requirements which were already mentioned in Section 3.1 there are many possible solutions. A web application which is based on web services has been determined to be the more appropriate design which fits the system specification described in the previous.

In this work, we choose web service and abandon the use of functions which have been developed in MDSS. Beside the ease of interoperation with a Web-based system, there are also some other reasons to explain this decision.

Web service is platform-independent and program language-independent. It means that the client side can be programmed in C++ and running under Windows, while the server side is programmed in Java and running under Linux. In other words, what the client side looks like is not important in this work. The task is about the server side and developing it with web services is easier for the next developer who wants to continue this work later.

In short, all the resources in Figure 3.5 should be identified by URIs. The 2 decision support functions are transformed into decision support services. The consumer’s requirements are transported to the server side through the query part of URI. The appropriate results are returned back in 2 standard representations, XML and JSON.

Before a search or a calculation starts, the initial conditions must be checked whether it is in the range. If not, an error message will be returned back and new initial conditions should be given.

#### 3.4.1 Candidate Search Service

The candidate search service in [5] is called offering matcher. The filter has been separated by the numerical-performance characteristics and non-numerical-performance characteristics. In the new system, the non-numerical-performance characteristics contain performance characteristics of operating system, software licence and I/O (it is described by providers with low, moderate, high and very high); the rest of the performance characteristics are numerical-performance characteristics. The necessary performance characteristics for one configuration are given to consumer before he starts the candidate searching. Except all the requirements which are defined by the consumer, the rest of the performance characteristics which are also defined by providers are not considered in the matching process.
Under the consideration of the expansion of database the algorithm of candidate search service is modified from the offering matcher algorithm in MDSS. It is showed in Algorithm 1. The data type of a few parameters is HashMap class instead of List class in MDSS. HashMap class is used to implement a mapping between key and value. It is easy to search and compare with another value of the same key. But it has also some disadvantages that the sequence of one map is unfixed and for one key there can be only one value. It means if one key maps to more than one values, the class List could be used otherwise the last value will be overridden all the previous values. The input parameter is $M_p$. It’s a collection of the requirements from the URI query part. The parameter $M_c$ is identified by the configuration id as key and maps to a performances-list called $L_{cp}$ as value which contains all the performances of this configuration. $L_{cp}$ is received by a database sql query search. Then comparing each performance of $L_{cp}$ with $M_p$, if the performances are the same or $L_{cp}$’s value is bigger than $M_p$’s, this configuration is a candidate configuration. Otherwise the configuration id will be deleted from $M_c$.

**Algorithm 1 Candidate Search**

```plaintext
1: procedure CANDIDATESEARCH($M_p$
2:  $M_c := SQLquery(M_p("servicetype"), M_p("offering"), M_p("provider"));$
3:  for each $c \in M_c$ do
4:      $L_{cp} := M_c(c);$  
5:      for each $p \in L_p$ do
6:          if $\neg M_p(p) = NULL$ then
7:              if $\neg L_{cp}(p) = NULL$ then
8:                  if $p = "os" \lor p = "io" \lor p = "licence"$ then
9:                      if $\neg L_{cp}(p) = M_p(p)$ then
10:                         remove $c$ from $M_c$;
11:                         break;
12:                     end if
13:                  else
14:                      if $L_{cp}(p) < M_p(p)$ then
15:                         remove $c$ from $M_c$;
16:                         break;
17:                     end if
18:                 end if
19:             end if
20:          end if
21:      end for
22:  end for
23: end procedure
```

### 3.4.2 Cost Calculator Service

In the cost calculator service part, the algorithm is completely changed with respect of the cost calculation algorithm of MDSS. The specific reasons for this change are listed in the following:
In MDSS, for the static unit-price the initial condition for no input-value is a default value (1); for the dynamic unit-price the initial condition is the minimum and maximum consumption in every amount range and a list of costs was shown to consumer. An appropriate default value should be re-defined. This value should be at least help consumer to get a general idea of the cost of this service. For static unit-price the initial value is 5000 except the time value. If this service is calculated by hour, the initial value is 500. In addition, the utilization time of a Cloud service is 10 months as default. For dynamic unit-price the initial value of no input-value is the maximum consumption of the biggest amount range.

Some new providers are added to the new knowledge base, so the cost calculator should contain all the situations for each provider. There are 2 special providers, Amazon Web Services and Flexiscale. If the provider is Amazon Web Services, the variable which is used to separate the usage amount range should be processed carefully. Because only for the provider AWS, there is upfront for some Cloud services and it is charged per 1 year or 3 years. It means this variable’s unit is maybe "Hour" but the unit of usage amount range is "Year" or "Month". There should be a unit-transformation for the "Hour" unit and a check-process for the value of "Hour" variable which cannot be more than 730 hours per month. If the provider is Flexiscale, the usage amount is a formula with multiple variables that the result is static according to pricing table. The pricing is divided into different tariff by the amount of units.

The cost calculator is divided into static-calculator and dynamic-calculator. The static-calculator part is relatively straightforward; the dynamic-calculator part however requires additional logic. The formula’s usage amount range should be checked always as long as the variable value changes. So all the results which are selected by user’s requirements are retained whether the minimum or maximum values of these cost formulas does not reach the requirement. If the cost formula is determined by a fixed rang as what has been done in MDSS, there may be mistakes when the value of the dynamic variable is out of range.

The total cost consists of service usage amount cost, data transfer cost and upfront. Before adding these 3 parts together, the condition should be checked whether they stay at the same data-center. Especially, if the service data-center is worldwide but the data transfer data-center is in EU, the total cost should be calculated differently. The result is built up by 2 parts, service cost with data transfer cost in EU and service cost without data transfer in other geographical areas.

Actually, the process method of all the providers except Flexiscale is using the same idea, but what are mentioned above must be treated with carefully and all the possible situations should be included in the implementation part. The provider Flexiscale is a dynamic unit-price but for the usage amount not for the cost calculation formula.
Algorithm 2 Cost Calculator

1: procedure costcalculator(id, M_v, M_cd, M_p, months)
2:     service_c := 0;
3:     upfront_c := 0;
4:     datatransfer_c := 0;
5:     L_c := SQLquery(id);
6:     M_c := SQLquery(id);
7:     L_v := getVars(L_c);
8:     M_v := unitCheck(M_v);
9:     for index = 0 \rightarrow \text{Size}(L_c) do
10:         cid := L_c(index);
11:         L_p := M_c(cid);
12:         if ¬Flexiscale then
13:             if staticunit − price then
14:                 upfront_c := Upfront(cid, M_v);
15:                 service_c := costCalculator(formula, M_v);
16:                 datatransfer_c := dataTransfer(cid, M_v);
17:                     if static − rate then
18:                         service_c, datatransfer_c, upfront_c do not change for each month;
19:                     else
20:                         if month < max(M_p) then
21:                             month := max(M_p);
22:                         end if
23:                         for i = 1 \rightarrow month do
24:                             for each vp ∈ M_p do
25:                                 if i ∈ M_p(vp) then
26:                                     M_v(vp) := M_v ∗ (1 + rate/100);
27:                                     M_v := unitCheck(M_v);
28:                                 end if
29:                             end for
30:                         service_c := costCalculator(formula(cid), M_v)
31:                         datatransfer_c := dataTransfer(cid, M_v);
32:                     end for
33:                 end if
34:             else
35:                 get min, max from L_p;
36:                 v := the variable which identifies the usage amount range;
37:                 if −(M_v(v) < max ∧ M_v(v) > min) then
38:                     for start := 0 \rightarrow \text{Size}(L_c) do
39:                         cid := L_c(start);
40:                         L_p := M_c(cid);
41:                         get new min, max from L_p;
42:                         if M_v(v) < max ∧ M_v(v) > min then
43:                             get new cid;
44:                             index := start;
45:                         break;
46:                     end if
47:                 end for
48:             end if
49:         else
50:             get min, max from L_p;
51:             v := the variable which identifies the usage amount range;
52:             if −(M_v(v) < max ∧ M_v(v) > min) then
53:                 for start := 0 \rightarrow \text{Size}(L_c) do
54:                     cid := L_c(start);
55:                     L_p := M_c(cid);
56:                     get new min, max from L_p;
57:                     if M_v(v) < max ∧ M_v(v) > min then
58:                         get new cid;
59:                         index := start;
60:                     break;
61:                 end if
62:             end for
63:         end if
64:     end if
65: end if

Decision support for different migration types of applications to the Cloud
upfront := Upfront(cid, Mv);
service := costCalculator(formula(cid), Mv);
datatransfer := dataTransfer(cid, Mv);

if static - rate then
    service, datatransfer, upfront do not change for each month;
else
    if month < max(Mp) then
        month := max(Mp);
    end if
    for i = 1 → month do
        for each vp ∈ Mp do
            if i ∈ Mp(vp) then
                Mv(vp) := Mv * (1 + rate/100);
                Mv := unitCheck(Mv);
                if vp(name) = v(name) then
                    if ¬(Mv(vp) < max ∧ Mv(vp) > min) then
                        for start := 1 → Size(Lc) do
                            cid := Lc(start);
                            Lp := Mc(cid);
                            get new min, max from Lp;
                            if Mv(vp) < max ∧ Mv(vp) > min then
                                get new cid;
                                index := start;
                                break;
                            end if
                        end for
                    end if
                end if
            end if
        end for
    end if
end if

else
    get min, max from Lp;
    value_u := UsageamountCalculator(formula(cid), Mv);
    if ¬(value_u < max ∧ value_u > min) then
        for start := 0 → Size(Lc) do
            cid := Lc(start);
            Lp := Mc(cid);
            get new min, max from Lp;
            if value < max ∧ value > min then
                get new cid;
                index := start;
                break;
            end if
        end for
    end if
else
    value_u := UsageamountCalculator(formula(cid), Mv);
    if ¬(value_u < max ∧ value_u > min) then
        for start := 0 → Size(Lc) do
            cid := Lc(start);
            Lp := Mc(cid);
            get new min, max from Lp;
            if value < max ∧ value > min then
                get new cid;
                index := start;
                break;
            end if
        end for
    end if
end if

Decision support for different migration types of applications to the Cloud
end if

\( \text{service}_c := \text{cost value in } L_p; \)

\( \text{datatransfer}_c := \text{dataTransfer}(cid, M_v); \)

\[ \begin{align*}
\text{if} \quad & \text{static - rate then} \\
& \text{service}_c, \text{datatransfer}_c, \text{upfront}_c \text{ do not change for each month;} \\
\text{else} \\
& \text{if} \quad \text{month} < \max(M_p) \text{ then} \\
& \quad \text{month} := \max(M_p); \\
\text{end if} \\
& \text{for} \ i = 1 \rightarrow \text{month} \text{ do} \\
& \quad \text{for each } \text{vp} \in M_p \text{ do} \\
& \quad \quad \text{if} \ i \in M_p(\text{vp}) \text{ then} \\
& \quad \quad \quad M_v(\text{vp}) := M_v \ast (1 + \text{rate}/100); \\
& \quad \quad \quad M_v := \text{unitCheck}(M_v); \\
& \quad \quad \text{end if} \\
& \quad \text{end for} \\
& \quad \text{value}_u := \text{UsageamountCalculator}(\text{formula}(cid), M_v); \\
& \quad \text{if} \quad \neg(\text{value}_u < \max \land \text{value}_u > \min) \text{ then} \\
& \quad \quad \text{for} \ \text{start} := 1 \rightarrow \text{Size}(L_c) \text{ do} \\
& \quad \quad \quad \text{cid} := L_c(\text{start}); \\
& \quad \quad \quad L_p := M_c(\text{cid}); \\
& \quad \quad \quad \text{get new } \min, \max \text{ from } L_p; \\
& \quad \quad \quad \text{if} \ \text{value} < \max \land \text{value} > \min \text{ then} \\
& \quad \quad \quad \quad \text{get new } \text{cid}; \\
& \quad \quad \quad \quad \text{break}; \\
& \quad \quad \text{end if} \\
& \quad \text{end for} \\
& \quad \text{end if} \\
& \text{end if} \\
\end{align*} \]

\( \text{service}_c := \text{cost value in } L_p; \)

\( \text{datatransfer}_c := \text{dataTransfer}(cid, M_v); \)

end for

end if

end for

end procedure

Decision support for different migration types of applications to the Cloud
The details of how to realize this algorithm are illustrated in Algorithm 2. This algorithm will be explained from 2 parts, initial conditions (line 2 – line 8) and concrete steps (line 9 – the end). Concrete steps is separated into 2 sub-parts, the provider is Flexiscale (line 84 – line 130) and other providers (line 12 – line 83). Furthermore the sub-part of other providers consists of static unit-price (line 13 – line 33) and dynamic unit-price (line 34 – line 83). For each unit-price situation, static rate and dynamic rate are considered.

There are 5 input data which are id, $M_v$, $M_{cd}$, $M_p$ and month. id is the configuration’s id. $M_v$ is the start values which are given by consumer through the URI query part. $M_{cd}$ is a mapping between coefficient id and a list of data transfer ids. $M_p$ is a mapping between necessary variable for cost formula calculation and a list of $[\text{start, end, rate}]$s which are from the URI query part with usage_pattern parameter. months is the utility time of this offering. Additionally, some initial values are retrieved with the help of SQLquery() method and getVars() method. The SQLquery() method is to find all the coefficient ids in a list assistance of SQL database query. $L_c$ is a list of coefficient ids and $M_c$ is a mapping between coefficient id and a list which is built up by a collection of values and each index’s meaning is cost formula, location id, zone id, usage amount id, minimum value and maximum value. The location id and zone id are used later to check whether there is a restriction of geographical area while the usage amount id is used later to identify whether this cost formula is a static unit-price or dynamic unit-price. $L_v$ is a list of all the necessary variables for this configuration’s cost formula to calculate. Besides this initial conditions, there are two maximum-usage methods, unitCheck() and dataTransfer(). unitCheck() method is used to control which variable is over range, e.g. "Hour" is bigger than 730. dataTransfer() is to calculate the data transfer cost if this configuration has a data transfer cost.

The static rate means the value is not changed. In other words, there’s no input value of usage_pattern. The rate is changed by usage pattern which are given by consumer with the help of URI query part. In the dynamic rate, if the variable is the one which has a usage amount range, its value should be checked whether it still satisfies the range after the new value is calculated. If not, the new range should be found. As long as the value is changed, the following step must be unitCheck() method.

For the provider Flexiscale, it is an obvious dynamic unit-price situation. But for each range the coefficient cost value is fixed, the usage amount is a cost formula. It is different with different input values. The result of the formula is compared with the minimum and maximum of the range. If it is in the range, the coefficient cost value is the cost of this offering. If not, a new range will be searched in order to get a new coefficient cost value.

### 3.5 Decision Support System

In this section, the envisioned Decision Support System is described through a series of User Interfaces. Balsamiq Mockups[^2] is used to build these user interfaces. All the UIs

[^2]: http://balsamiq.com/
are summarized by the following figures. These figures are combined with some tables and buttons. Tables are used to described an application or some requirements while buttons are working with submitting information together and sometimes it will start a new page to collect more information.

Migration Type I and Migration Type II are interpreted in this section. All the necessary requirements of Type I are listed in Figure 3.6, Figure 3.7, Figure 3.8 and Figure 3.9. In general, the user has to provide input to the system by answering one or more of the following questions:

1. Which component/components his application contains.
2. Which component/components he wants to migrate/replace to the Cloud.
3. Which Cloud service provider he prefers.
4. Which performance characteristics of his Cloud offering he hopes to constrain.
5. Which configurations among all the candidate offerings he wants a cost calculation for.
6. How long he wants to rent this offering.
7. Where he wants to his data-center to be located.
8. Which usage variables should be assigned with specific values.
9. What kind of trend among these variables is.

![MiDSuS](http://www.diva.com/)

Figure 3.6: Details of Type I
Generally, if the user finishes answering these questions, it means he has already found these offerings which he wants. The concrete implementation of Type I is explained as follows: in the beginning, user answers the first three questions above as shown in Figure 3.6. These three questions can be thought as application description like what was talked in Figure 3.2 of Section 3.2. In Type I, this application description is collected by different components of one application. In this example, the existing application components are web server, resizable compute, big data workload, RDBMS and raw block level storage. Two of these components are attempted to migrated to the Cloud, they are resizable compute component and raw block level storage component. Obviously, resizable compute component belongs to infrastructure service type while raw block level storage is in the block storage service. The providers for these components are selected by "Any of them". It means the user wants to get all these appropriate offerings but does not care about their providers.

Afterwards all the possible performance characteristics are collected by a new window in Figure 3.7. Before user enters his new requirements, more than one performance have already been determined in the application description step. As what is said above, they are the service type and provider. In Figure 3.7 for infrastructure service type, a set of performances are entered by user. In this example, 4 features are give. CPU cores is 8; CPU speed is 1.2 GHz; RAM of this offering is 10 GB; Operating system is Windows. As long as these necessary performances are entered, the candidate search service can
be started.

All the candidate configurations will be listed in a new page like in Figure 3.8. It’s time for user to answer the fifth question. Namely, he should select some offerings from all the candidate offerings in Figure 3.8. At the same time, the last four questions will be settled in Figure 3.9. Some evaluate variables’ values are entered. In this example, the basic value of Hour variable is 240 hours, the basic value of storage is 500 GB. If there are also some other variables which are necessary during the cost formula calculation, some default values should be defined in such situation. The specific values will be talked in implementation chapter. In the dynamic rate situation, the Hour variable is increased by 15% between the second month and the 6th month and decreased by $-20\%$ between the 10th month and the 12th month; the storage variable is increased by 20% between the first month and the 5th month. So far all the questions are already answered. Now
it's time to calculate the costs. All these results are showed in Figure 3.10. Use the
details button, a trend of each offering’s cost is displayed on the same page. There also
some other functions can be done, like sorts all the information by update date, data
center or cost. With this Ranking result user can directly see which one is the best one.
The other example is about migrating to the Cloud with Type II. In Figure 3.11, all application components are classified into three layers. We focus only on a three layer application architecture in this work. These three layers are presentation layer, business layer and data layer. Among these defined components in this work there is also a cross layer. All application components which is defined in Figure 3.6 is generalized in different layer in Figure 3.11. The implementation of Type II is similar as what was introduced in Migration Type I. But in one layer there may be more than one service type. In this example, the data layer is chosen to migrate to the Cloud. In this layer this application contains RDBMS component and raw block level storage. It means the SQL database service type and block storage service type are ready to migrate.
In Figure 3.12, the performances of both service types are entered. On SQL database side, user needs a Cloud offering which has 4 virtual cores with 10 GB of local instance storage and MySQL. On block storage side, user has only one requirement that instance storage is at least 1000 GB.

![Figure 3.12: Candidate Requirements Details of Type II](image)

As a result, in Figure 3.13 there are two set of configurations. The one on the left of this figure is about the SQL database service type and the other one which is on the right side is about the block storage service type. The first five configurations are chosen in order to make a cost comparison.
### Figure 3.13: Result of Candidate Search Details of Type II

Type I: I want to replace one or more components of my application to the Cloud.

Type II: I want to migrate some of my application functionality to the Cloud.

Select configurations which I attempt to use and these configuration will be calculated in the next step:

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Type I</th>
<th>Type II</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Select</strong> Configuration</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block Storage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CBS snapshots to Amazon S3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Persistent Disk</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Snapshots</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EBS provisioned IOPS volumes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data assistedBD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Standard Volumes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EBS snapshots to and from disk(BD)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OSS Volumes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Volume Snapshots</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Storage Geographically</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Storage Locally</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EBS standard volumes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Select configurations of saas/Database**

<table>
<thead>
<tr>
<th>Select</th>
<th>Configuration</th>
<th>Type I</th>
<th>Type II</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL</td>
<td>Block Storage</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td>CBS snapshots to Amazon S3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td>Persistent Disk</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td>Snapshots</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td>EBS provisioned IOPS volumes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td>Data assistedBD</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td>Standard Volumes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td>EBS snapshots to and from disk(BD)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td>OSS Volumes</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td>Volume Snapshots</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td>Storage Geographically</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td>Storage Locally</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL</td>
<td>EBS standard volumes</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Submit Configurations
The initial value are entered in Figure 3.14. In this example, user only entered two values for the SQL database, no value for the block storage. It means he cares only about the cost comparison of SQL database’s configurations. The usage variable has a value of 400 hours and storage variable has a value of 10000 GB. After setting the evaluate variables’ values the cost calculations of each selected configuration are shown in Figure 3.15. It is a long list whose same configuration has different cost with different data center. User can click details button to get more information about one configuration’s cost in months and also the necessary variables’ values of this cost calculation.
Cost of each configuration which was selected before.

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Update date</th>
<th>Instance type</th>
<th>Region</th>
<th>Price</th>
<th>Service cost</th>
<th>Disk transfer cost</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Double Extra Large Heavy Utilization Reserved High Memory</td>
<td>2012-04-12</td>
<td>Northern Virginia</td>
<td>$19820.00</td>
<td>$142.00</td>
<td>$1340.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Double Extra Large Heavy Utilization Reserved High Memory</td>
<td>2012-04-12</td>
<td>California</td>
<td>$19764.00</td>
<td>$142.00</td>
<td>$1364.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Double Extra Large Heavy Utilization Reserved High Memory</td>
<td>2012-04-12</td>
<td>Sao Paulo</td>
<td>$32000.00</td>
<td>$186.00</td>
<td>$2330.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Double Extra Large Heavy Utilization Reserved High Memory</td>
<td>2012-04-12</td>
<td>Tokyo</td>
<td>$22076.00</td>
<td>$142.00</td>
<td>$1399.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Double Extra Large Heavy Utilization Reserved High Memory</td>
<td>2012-04-12</td>
<td>Singapore</td>
<td>$18164.00</td>
<td>$122.00</td>
<td>$1364.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Double Extra Large Medium Utilization Reserved High Memory</td>
<td>2012-04-15</td>
<td>Oregon</td>
<td>$10380.00</td>
<td>$737.00</td>
<td>$813.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Double Extra Large Medium Utilization Reserved High Memory</td>
<td>2012-04-15</td>
<td>California</td>
<td>$18903.00</td>
<td>$377.00</td>
<td>$831.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Double Extra Large Medium Utilization Reserved High Memory</td>
<td>2012-04-15</td>
<td>Sao Paulo</td>
<td>$25482.00</td>
<td>$377.00</td>
<td>$831.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Double Extra Large Medium Utilization Reserved High Memory</td>
<td>2012-04-15</td>
<td>Tokyo</td>
<td>$9974.00</td>
<td>$377.00</td>
<td>$831.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Double Extra Large Medium Utilization Reserved High Memory</td>
<td>2012-04-15</td>
<td>Singapore</td>
<td>$14803.00</td>
<td>$377.00</td>
<td>$831.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Double Extra Large Medium Utilization Reserved High Memory</td>
<td>2012-04-15</td>
<td>Ireland</td>
<td>$14803.00</td>
<td>$377.00</td>
<td>$831.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Double Extra Large Medium Utilization Reserved High Memory</td>
<td>2012-04-15</td>
<td>Sydney</td>
<td>$922.00</td>
<td>$238.00</td>
<td>$831.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Double Extra Large Heavy Utilization Reserved High Memory Instance Standard</td>
<td>2012-04-15</td>
<td>Oregon</td>
<td>$1326.00</td>
<td>$238.00</td>
<td>$831.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Double Extra Large Heavy Utilization Reserved High Memory Instance Standard</td>
<td>2012-04-15</td>
<td>California</td>
<td>$18809.00</td>
<td>$238.00</td>
<td>$831.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Double Extra Large Heavy Utilization Reserved High Memory Instance Standard</td>
<td>2012-04-15</td>
<td>Sao Paulo</td>
<td>$25482.00</td>
<td>$238.00</td>
<td>$831.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Double Extra Large Heavy Utilization Reserved High Memory Instance Standard</td>
<td>2012-04-15</td>
<td>Tokyo</td>
<td>$9974.00</td>
<td>$238.00</td>
<td>$831.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Double Extra Large Heavy Utilization Reserved High Memory Instance Standard</td>
<td>2012-04-15</td>
<td>Singapore</td>
<td>$14803.00</td>
<td>$238.00</td>
<td>$831.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Double Extra Large Heavy Utilization Reserved High Memory Instance Standard</td>
<td>2012-04-15</td>
<td>Ireland</td>
<td>$14803.00</td>
<td>$238.00</td>
<td>$831.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Double Extra Large Heavy Utilization Reserved High Memory Instance Standard</td>
<td>2012-04-15</td>
<td>Sydney</td>
<td>$922.00</td>
<td>$238.00</td>
<td>$831.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Extra Large Medium Utilization Reserved Instance Standard</td>
<td>2012-04-15</td>
<td>Oregon</td>
<td>$2490.00</td>
<td>$392.00</td>
<td>$279.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Extra Large Medium Utilization Reserved Instance Standard</td>
<td>2012-04-15</td>
<td>California</td>
<td>$27624.00</td>
<td>$392.00</td>
<td>$279.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Extra Large Medium Utilization Reserved Instance Standard</td>
<td>2012-04-15</td>
<td>Sao Paulo</td>
<td>$2490.00</td>
<td>$392.00</td>
<td>$279.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Extra Large Medium Utilization Reserved Instance Standard</td>
<td>2012-04-15</td>
<td>Tokyo</td>
<td>$7996.00</td>
<td>$392.00</td>
<td>$279.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Extra Large Medium Utilization Reserved Instance Standard</td>
<td>2012-04-15</td>
<td>Singapore</td>
<td>$3032.00</td>
<td>$392.00</td>
<td>$279.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Extra Large Medium Utilization Reserved Instance Multi-AZ</td>
<td>2012-04-15</td>
<td>Oregon</td>
<td>$2490.00</td>
<td>$392.00</td>
<td>$279.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Extra Large Medium Utilization Reserved Instance Multi-AZ</td>
<td>2012-04-15</td>
<td>California</td>
<td>$27624.00</td>
<td>$392.00</td>
<td>$279.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Extra Large Medium Utilization Reserved Instance Multi-AZ</td>
<td>2012-04-15</td>
<td>Sao Paulo</td>
<td>$2490.00</td>
<td>$392.00</td>
<td>$279.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Extra Large Medium Utilization Reserved Instance Multi-AZ</td>
<td>2012-04-15</td>
<td>Tokyo</td>
<td>$7996.00</td>
<td>$392.00</td>
<td>$279.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Extra Large Medium Utilization Reserved Instance Multi-AZ</td>
<td>2012-04-15</td>
<td>Singapore</td>
<td>$3032.00</td>
<td>$392.00</td>
<td>$279.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Extra Large Heavy Utilization Reserved Instance Multi-AZ</td>
<td>2012-04-15</td>
<td>Oregon</td>
<td>$37624.00</td>
<td>$392.00</td>
<td>$279.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Extra Large Heavy Utilization Reserved Instance Multi-AZ</td>
<td>2012-04-15</td>
<td>California</td>
<td>$37624.00</td>
<td>$392.00</td>
<td>$279.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Extra Large Heavy Utilization Reserved Instance Multi-AZ</td>
<td>2012-04-15</td>
<td>Sao Paulo</td>
<td>$37624.00</td>
<td>$392.00</td>
<td>$279.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Extra Large Heavy Utilization Reserved Instance Multi-AZ</td>
<td>2012-04-15</td>
<td>Tokyo</td>
<td>$7996.00</td>
<td>$392.00</td>
<td>$279.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
<tr>
<td>Extra Large Heavy Utilization Reserved Instance Multi-AZ</td>
<td>2012-04-15</td>
<td>Singapore</td>
<td>$3032.00</td>
<td>$392.00</td>
<td>$279.00</td>
<td>$0.00</td>
<td>Details</td>
</tr>
</tbody>
</table>

Figure 3.15: Results of Type II

Decision support for different migration types of applications to the Cloud
The starting UI pages for Type II, Type III and Type IV are shown in Figure 3.11, Figure 3.16 and Figure 3.17. For these three types only the main tables are given. The details steps for requirements collection and results calculation are similar as Type I. With the help of Figure 3.7 and Figure 3.9 the candidate requirements and cost requirements are determined as these initial conditions for the APIs. In addition with the requirements from the type table all the requirements collections are finished like what is represented in use case Figure 3.2. As discussed in Section 2.1, the most fundamental difference among these 4 types is either a migration between the application components or a migration of a whole application layer. In other word, whether there is a functionality relationship between each components is what distinguishes each migration type. Migration Type
III is a specific classification of Type I. Both of them collect application descriptions from different components. In Figure 3.6 and Figure 3.16, it can be seen that some application components are defined. There are several components for each service type. These definitions are collected and summarized from the Cloud services providers. Migration Type III is a complete migration type of Type I. But there is only one offering can be chosen to complete Migration Type III that is VMs offering.

Type IV is a special classification of Type II. They all collect requirements from different layer. In this work as mentioned above, an application is structured as a three-layered application. The components between each layer are collected in cross-layer. In Type IV, it can choose any offering which user wants except the VMs offering, because it is already included in Type III. It should be careful that Type IV is also a complete migration type. It means with Type III and Type IV application can be migrated to the Cloud completely. Actually, in today’s market, there is not so many applications which are migrates with Type IV, because using this migration type there are very higher risks and much more troubles than other particular migration types. Generally speaking, people prefer to choose Type III rather than Type IV, when they have to consider a complete migration situation. Type III is the most frequent choice. It is easy to understand and operate with the application. In Appendix A.1 there is only a little size use case of Cloud migration, but Type III has a large proportion. Compare with Type IV, only one use case is given.

3.6 Summary

In this chapter, a concrete structure of this work is explained. From the available information, a set of requirements are identified as a direction to build this new system. The specification is abstracted from these requirements and illustrated as a Use Case diagram. According to these foundations, we choose to use some web services to complete the data handler part of the new system instead of the previous functions in MDSS, because it is easy to re-use in future works. As a basic support part a database named Cloud providers knowledge base is built which is expanded from the database of MDSS. This new database includes all the available costs of one offering and a more refined service types. It contains as many data centers of each provider as possible in order to help user to have a better knowledge of each offering. Actually, all these preparations are done as the foundation for a decision support system. The further system completes the migration tasks completely. It helps user from the migration type on until the costs of each selected candidate offerings are calculated. All of this searching and calculating are based on those web APIs envisioned in the previous step.
4 Implementation

In this chapter, the implantations of the system designed in the previous section is presented. Two systems are actually built in this work. One is called Nefolog, it composed by database and a set of web restful services. The other is called MiDSuS, it is the decision support system which realized the tasks of migration decision. MiDSuS uses the RESTful APIs which are offered by Nefolog to implement a web application.

4.1 Cloud Provider Knowledge Base of Nefolog System

As in MDSS, the Cloud Provider Knowledge Base was implemented as a relational database on the basis of the Entity-Relation diagram shown in Figure 3.5.

However, in MDSS, Microsoft SQL Server was used, a non-open source solution with strict licensing requirements. In order to get more scalability and save development and licensing costs, it decided to realize the database with an open source software solution. The PostgreSQL RDBMS solution was chosen for this purpose.

PostgreSQL is released under the PostgreSQL license and is free. It is developed by the PostgreSQL Global Development Group. This database system is very similar to the other database systems because it implements data queries by using SQL language, too. These data are linked through the foreign key together and exist as a series of tables. The PostgreSQL main advantage relative to other competitors is its programmability.

Figure 4.1 presents the data model of the new system. This database consists of 16 entities. In the provider entity, the numbers of providers is increased from 2 to 6. Except from Windows Azure and Google, it also includes Amazon Web Services, HP Cloud, Rackspace and Flexiscale. So the number of offerings is increased from 18 to 51 and these offerings contain 520 configurations with 472 performances. It’s almost 4 times as large as before. The 5 previously available service types are separated into 12 types. A comparison of the two service types tables is shown in Table 4.1. This more fine-granular classification for the service type allows users to find an appropriate Cloud service much more directly and quickly.

For the identification of matching offerings, the general idea is similar to MDSS. Through the performancemapping entity the user knows which performance he should submit if the service type is already chosen. The performancemapping entity is a collection of the performances which are not null and separated by the service types. It means the system can compare each configuration according to the performances between users require and the provider provides.

[^1]: http://www.postgresql.org/
Figure 4.1: Data model
On the cost calculator side, different parameters (location, zone or amount of usage) can lead to a number of different costs for the same factors (upfront, data transfer or coefficient) which was separated from the final cost. There are 15 geographical areas included and they belong to 7 zones. The 3 factors are seen as 3 coefficients of the final cost. Users need to submit the usage time and the GB of data transfer in order to achieve the cost calculation. The final cost table has a total of 21856 combinations with 3686 upfront costs, 185 costs of data transfer and 4733 prices of configurations. There is also a problem which must be advertent before the programming of web services starts. Usually it is possible that there are more than one parameter for one cost calculator formula. Some formulas are separated by different usage amount. It means that the parameter which is decided the formula should be indicated and set in the first position in order to make the formula filter much easier. The special provider Flexiscale has 18 configurations and they are priced on units. It means the system supplies the cost of this provider’s service through the amount of units. There are 11 prices of buying units. It has a one-to-one relationship with the Configuration entity.

### 4.2 Services of Nefolog System Implementation

The new system is a restful web services system and is named Nefolog. An open-source software Java EE has been chosen to build it instead of Microsoft Visual Studio 2010. Restlet Framework is selected to achieve these RESTful APIs.
Table 4.2: Overview of Languages and Frameworks to Support Restful Web Services System

<table>
<thead>
<tr>
<th>Framework</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jersey</td>
<td>Java</td>
</tr>
<tr>
<td>Restlet</td>
<td>Java</td>
</tr>
<tr>
<td>Django</td>
<td>Python</td>
</tr>
<tr>
<td>Spring</td>
<td>Java</td>
</tr>
<tr>
<td>Konstrukt</td>
<td>PHP</td>
</tr>
<tr>
<td>Recess</td>
<td>PHP</td>
</tr>
<tr>
<td>Rails</td>
<td>Ruby</td>
</tr>
</tbody>
</table>

There are some many program languages and corresponding frameworks that can build up a restful web services system. From the Table 4.2 some of the available framework are presented.

In fact Python with Django is a simple way, because Python is easy to learn even for a beginner and the framework of a restful web services system on Django is uncomplicated. But finally Python was not chosen. It is a problem about trend and popularity. A restful web services in Java is so common in the market and examples are everywhere. It does not mean that Python is not a trend or not popular, but more people learn Java as a first language of OOP. In other words, Java is more familiar to people than Python. On PHP side, there are so many different kinds of PHP versions and for different version the syntax is different. It is hard for others to continue this work later or even could bring them some new unnecessary troubles. When program language is determined, the next problem is about suitable framework. Restlet is probably the first REST framework and is available in market from 2005. Comparing with Jersey and Spring Restlet is a lightweight framework and provides some low-level REST support. All of them support both server side and client side. There is no conclusive evidence that Restlet is better or Jersey is better. Restlet was therefore chosen for the implementation as a matter of personal preference.

There are many tutorials in [19] on how to build RESTful API. There are 2 types of REST resources, collection resource and instance resource. By definition, a collection resource is always a plural form of a word and means a data collection; an instance resource is a singular form of a word and means one from of this collection. We can use a Restlet component as a container of Restlet applications or Servlet engine.

In this work, all the resources identified by URIs and have 2 representations such as XML data and JSON data. The Restlet applications are developed under separate URI paths. Servlet engine which is considered as a component to contain all these applications and provides the server HTTP connector is chosen to finish this work, because we hope to use this project finally under no base URI path, it means finally a package of this Servlet
project will be done as a WAR file. With this WAR file this project will run in any other environments by rebuilding the provider knowledge base from a database backup file.

Each resource is built up by two classes, a `ServerResource` class and a class which is used to implement the representations. A `Representation` instance is used to for both XML representation and JSON representation. Class `DomRepresentation` is a `XMLRepresentation` class, so our XML representation is based on a DOM document. Class `JsonRepresentation` is a `WriteRepresentation` class, so this JSON representation is based on a JSON document. Then an `Application` class is created. With the help of a `Restlet` class a `Router` object is initialized to create some `attach` instances and assigns each `attach` to a `ServerResource` class.

![Class/ServiceTypesApplication](aclass4.png)

**Figure 4.2: UML Classes Diagram of Nefolog**

In Figure 4.2 the UML classes diagram is given to have an overview of the whole Nefolog...
Implementation

system. As last paragraph mentioned, each resource is implemented by two classes. The classes which inherit their specifications from class ServerResource are connected by Router objects in the class which inherit from class Application. The operations between database and services are done by the class ConnUtil and called in classes which extend from ServerResource. In addition, there is a class called ErrorMessage which is used to warn the user when an error occurred. For example, the configuration id is not in the offering which user typed in the URI.

Table 4.3: Resource URIs supported by Nefolog

<table>
<thead>
<tr>
<th>URI</th>
<th>XML/JSON Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>(...=<a href="http://localhost:8080/nefolog">http://localhost:8080/nefolog</a>)</td>
<td>service types DNA/JSON Content</td>
</tr>
<tr>
<td>.../serviceTypes</td>
<td>URI of each service type</td>
</tr>
<tr>
<td>.../serviceTypes/{serviceTypeName}s</td>
<td>offerings of the service type</td>
</tr>
<tr>
<td>.../providers</td>
<td>providers</td>
</tr>
<tr>
<td>.../providers/{providerName}</td>
<td>offerings of the provider</td>
</tr>
<tr>
<td>.../offerings</td>
<td>offerings</td>
</tr>
<tr>
<td>.../offerings/{offeringName}</td>
<td>URI of each offering</td>
</tr>
<tr>
<td>.../offerings/{offeringName}/configuration/{configId}</td>
<td>URI of each configuration</td>
</tr>
</tbody>
</table>

All the supported by the Nefolog system URIs are listed in Table 4.3 and Table 4.4. The system name nefolog is used as root directory. All these services are hosted on Tomcat and PostgreSQL. Each URI has a mapping to 2 representations. The contents of these representations are the same. The outline of each URI’s XML/JSON contents can be seen in Table 4.3.
4.3 Decision Support Services of Nefolog System

In Table 4.4 the decision support services are given. The usability of these two services is according to the query part of the URI. Different result will be showed as XML or JSON with different query part. The consumer could first get initial information about how one service is processed and which parameters should be known before running this service.

Table 4.4: Decision Support Services URIs supported by Nefolog

<table>
<thead>
<tr>
<th>URI</th>
<th>XML/JSON Content</th>
</tr>
</thead>
<tbody>
<tr>
<td>(<a href="http://localhost:8080/nefolog">http://localhost:8080/nefolog</a>)</td>
<td>performances</td>
</tr>
<tr>
<td>.../candidateSearch</td>
<td>service types</td>
</tr>
<tr>
<td></td>
<td>URI of each service type</td>
</tr>
<tr>
<td></td>
<td>offerings of the service type</td>
</tr>
<tr>
<td></td>
<td>URI of the offering</td>
</tr>
<tr>
<td>.../candidateSearch?all</td>
<td>necessary performances of the offering</td>
</tr>
<tr>
<td></td>
<td>maximum value of the performance among all the configurations</td>
</tr>
<tr>
<td>.../candidateSearch?{query}</td>
<td>configurations</td>
</tr>
<tr>
<td></td>
<td>URI of each configuration</td>
</tr>
<tr>
<td>.../costCalculator</td>
<td>variables</td>
</tr>
<tr>
<td></td>
<td>necessary variables of the configuration</td>
</tr>
<tr>
<td>.../costCalculator?configid={id}</td>
<td>update time of the configuration</td>
</tr>
<tr>
<td></td>
<td>initial condition</td>
</tr>
<tr>
<td></td>
<td>total cost of each geographical area</td>
</tr>
<tr>
<td></td>
<td>upfront cost</td>
</tr>
<tr>
<td></td>
<td>total service cost</td>
</tr>
<tr>
<td></td>
<td>service cost per month</td>
</tr>
<tr>
<td></td>
<td>total data transfer cost</td>
</tr>
<tr>
<td></td>
<td>data transfer cost per month</td>
</tr>
</tbody>
</table>

4.3.1 Candidate Search Web Service

As what was discussed in Section 3.4.1 and according to the Algorithm 1 two classes are created to achieve it, class CandidateParams and class CandidateParamsResource.
4 Implementation

_CandidateParams_ is used to finish the representations part. In this work two representations are needed. _CandidateParamsResource_ is used to solve such problems which should connect to the database and interact with URI. _CandidateParamsResource_ class is a _ServerResource_ class. The method _candidateSearch()_ in Listing 4.2 is a concrete solution.

First of all, we should have a parameter called _inputPerformance_map_. It contains all the requirements from consumer. After that, we should have comparison values which can get from knowledge base. The database SQL query expression is in Listing 4.1. A parameter _configIDWithPerformanceList_ in _CandidateParams_ class is set to summarize all the necessary data from database in a list according to a configuration id. This list has to be set in a special sequence. Because of the character of _Map_ class we can compare these requirements with the help of a sequence of _keys_ list. These _keys_ list has a name _perform_. _perform_ is set at the beginning of Listing 4.2. Obviously, it is a special sequence for indexes of a list.

The comparison process is uncomplicated. We reject all the data which are smaller than the required values and collect the remaining data as the result of this service.

Listing 4.1: SQL Query Expression for Candidate Search Service

```java
String sql="";
if(this.inputPerformance_map.get("servicetype")!=null){
  sql=" and servicetype.type="+this.inputPerformance_map.get("servicetype")+";"
  if(this.inputPerformance_map.get("provider")!=null){
    sql=sql+" and provider.provider="+this.inputPerformance_map.get("provider")+";"
    if(this.inputPerformance_map.get("offering")!=null){
      sql=sql+" and offering.offering="+this.inputPerformance_map.get("offering")+";"
    }
  }
}
else{
  if(this.inputPerformance_map.get("offering")!=null){
    sql=sql+" and offering.offering="+this.inputPerformance_map.get("offering")+";"
  }
}
else{
  if(this.inputPerformance_map.get("provider")!=null){
    sql=sql+" and provider.provider="+this.inputPerformance_map.get("provider")+";"
    if(this.inputPerformance_map.get("offering")!=null){
      sql=sql+" and offering.offering="+this.inputPerformance_map.get("offering")+";"
    }
  }
  else{
    if(this.inputPerformance_map.get("offering")!=null){
      sql=sql+" and offering.offering="+this.inputPerformance_map.get("offering")+";"
    }
  }
}
```

Decision support for different migration types of applications to the Cloud
sql="select configuration.configid, performance.*, configuration.
configuration, offering.offering, provider.provider from servicetype,
offering, performance, configuration, provider where servicetype.
typeid=offering.typeid and offering.providerid=provider.providerid
and offering.offeringid=configuration.offeringid and configuration.
performanceid=performance.performanceid"+sql;

Listing 4.2: Candidate Search

```java
class CandidateSearch {
    public Map<String, String> candidateSearch() throws SQLException {
        Map<String, String> result_map = new HashMap<String, String>();
        for (int i = 3; i < this.all_query_list.size(); i++) {
            result.setPerformance(this.all_query_list.get(i));
        }
        Connection conn = ConnUtil.getConnection();
        conn.setAutoCommit(false);
        Statement stmt = null;
        ResultSet rs = null;
        stmt = conn.createStatement();
        rs = stmt.executeQuery(sql);
        while (rs.next()) {
            List<String> temp = new LinkedList<String>();
            String configid = rs.getString(1);
            if (rs.getString(2) != null) {
                temp.add(rs.getString(3));
                temp.add(rs.getString(4));
                temp.add(rs.getString(5));
                temp.add(rs.getString(6));
                temp.add(rs.getString(7));
                temp.add(rs.getString(8));
                temp.add(rs.getString(9));
                temp.add(rs.getString(10));
                temp.add(rs.getString(11));
                temp.add(rs.getString(12));
                temp.add(rs.getString(13));
                temp.add(rs.getString(14));
                temp.add(rs.getString(15));
                temp.add(rs.getString(16));
                temp.add(rs.getString(17));
                temp.add(rs.getString(18));
            }
            result.setConfigName(configid, rs.getString(19));
            result.setOfferingName(configid, rs.getString(20));
            result.setProviderName(configid, rs.getString(21));
            result.setConfigIDWithPerformanceList(configid, temp);
        }
        conn.commit();
        conn.close();
        Set<Entry<String, List<String>>> set_configIDWithPerformanceList =
        result.getConfigIDWithPerformanceList().entrySet();
        Iterator<Entry<String, List<String>>> it_configIDWithPerformanceList =
        set_configIDWithPerformanceList.iterator();
        while (it_configIDWithPerformanceList.hasNext()) {
            Entry<String, List<String>> entry_configIDWithPerformanceList =
            it_configIDWithPerformanceList.next();
            List<String> performanceList = new LinkedList<String>();
            performanceList = entry_configIDWithPerformanceList.getValue();
        }
    }
}
```
4 Implementation

```java
Iterator<String> it_list = performanceList.iterator();
Iterator<String> it_perform = result.getPerformance().iterator();
while (it_list.hasNext() && it_perform.hasNext()) {
    String text_list = it_list.next(); // value of one parameter
    String text_perform = it_perform.next(); // the parameter
    if (this.inputPerformance_map.get(text_perform) != null) {
        if (text_perform.equals("os") || text_perform.equals("io") ||
            text_perform.equals("licence") ||
            text_perform.equals("licence")) {
            if (text_list != null) {
                if (text_perform.equals("io")) { // there are 4 values of io
                    if (this.inputPerformance_map.get(text_perform).equals("very
                        high")) {
                        if (!text_list.equals(this.inputPerformance_map.get(
                            text_perform))) {
                            it_configIDWithPerformanceList.remove();
                            break;
                        }
                    } else if (this.inputPerformance_map.get(text_perform).equals
                        ("high")) {
                        if (text_list.equals("low") || text_list.equals("moderate")) {
                            it_configIDWithPerformanceList.remove();
                            break;
                        }
                    } else if (this.inputPerformance_map.get(text_perform).equals
                        ("moderate")) {
                        if (text_list.equals("low")) {
                            it_configIDWithPerformanceList.remove();
                            break;
                        }
                    } else if (!text_list.equals(this.inputPerformance_map.get(
                        text_perform))) {
                        it_configIDWithPerformanceList.remove();
                        break;
                    }
                }
            } else if (!text_list.equals(this.inputPerformance_map.get(
                text_perform))) {
                it_configIDWithPerformanceList.remove();
                break;
            }
        }
    } else if (text_list != null) {
        if (Double.valueOf(text_list) < Double.valueOf(this.
            inputPerformance_map.get(text_perform))) {
            it_configIDWithPerformanceList.remove();
            break;
        }
    }
}
Iterator<String> it = result.getConfigIDWithPerformanceList().keySet().
    iterator();
while (it.hasNext()) {
    String id = it.next();
    result_map.put(id, result.getConfigName().get(id));
}
return result_map;
```
4 Implementation

4.3.2 Cost Calculator Web Service

The cost calculator service works with the help of configuration id. Classes CostResource and Cost are created to realize this cost calculation task.

After doing the SQL database query search, some necessary data are obtained. As what was discussed in Section 3.4.2, the amount range is not considered as a restriction. The one restriction is a configuration id and the other restriction is a geographical area id, if consumer requires the data-center.

Before starting to calculate, the initial values of all the necessary variables are given. If not, some default values should be complemented as a part of initial conditions. The default value of "Hour" is 500; the default value of "month" is 10; the other default values are 5000, e.g. "GB", "GBExternalNetworkEgress" etc. These default values are not depending on each configuration, because a comparison should be made in the same initial condition, otherwise the comparison between each cost result makes no sense. The concrete process can be seen in Listing 4.3. All information about the initial variables is inserted in a Map instance called varsList_additional. Its key is implemented by a list which contains the value of this variable and a Boolean parameter. This Boolean parameter represents whether this value is from consumer or from the system. Although the consumer does not know the default value for each variable, with the help of this Boolean parameter all the initial conditions will be shown at the returned data. In that case, the consumer will know clearly how the result can be calculated.

Listing 4.3: Default initial Values

```java
Iterator<String> i_var_list = this.varsList.iterator();
while (i_var_list.hasNext()) {
    String insert_string = i_var_list.next();
    List<String> insert_list = new LinkedList<String>();
    if (this.query_vars.get(insert_string) == null) {
        if (insert_string.equals("Hour")) {
            this.query_vars.put(insert_string, "500");
            insert_list.add("500");
        } else if (insert_string.equals("Month")) {
            this.month = 10;
            this.query_vars.put(insert_string, "10");
            insert_list.add("10");
        } else {
            this.query_vars.put(insert_string, "5000");
            insert_list.add("5000");
        }
    } else {
        insert_list.add(this.query_vars.get(insert_string));
        insert_list.add("true");
    }
    this.varsList_additional.put(insert_string, insert_list);
}
```
There are two Boolean parameters, \textit{IsAmazon} and \textit{IsFlexiscale}. They are used to divide this method into 3 directions. One is that the provider is AWS; another one is that the provider is Flexiscale and the last one is that the provider is not AWS or Flexiscale. For each direction, the program idea is the same. One direction will also split into 2 branches. They are static unit-price and dynamic unit-price. For the dynamic unit-price, how to identify a cost formula with different usage amount range is very important. One unit-price situation can also be separated into 2 rates, static rate and dynamic rate. A Boolean parameter \textit{doMonthsCost} is used to implement this partition of rate. For the dynamic rate, how to calculate the change rate for each month and each variable is very complicated and should be treated carefully. All the possible situations should be considered. E.g. a variable is changed not from 1th month on but later; there is a change time overlapping for two variables; a variable is changed with different rate in different times, etc. A dynamic unit-price situation with a dynamic rate is a very complicated situation. For each changed value a range check must be done. A for loop is used to find a new limit range. If the trend of this variable is increased, the loop will continue in normal sequence. If the trend is decreased, this loop will continue in reverse order. The following parameters are created for this situation. First one is a HashMap instance \textit{pattern.map}. It is used to map each variable as key to its values for each month in a list \textit{varvalue.month} as value. Second one is \textit{coststring.list}. Its use is to collect cost formulas for each month. The last one is \textit{data.dataid}. It contains all the data transfer id for each month. The implementation of this situation is in Listing 4.4.

Listing 4.4: Dynamic Rate Situation

```java
if(Integer.valueOf(e_list.get(0).get(0))!=1){
    for(int index=1;index<Integer.valueOf(e_list.get(0).get(0));index++){
        varvalue_month.add(temp_cost);
        coststring_list.add(costString);
        data_dataid.add(data_ids);
    }
} else {
    varvalue_month.add(temp_cost);
    coststring_list.add(costString);
    data_dataid.add(data_ids);
    step=1;
}
for(int index=Integer.valueOf(e_list.get(0).get(0))+step;index<=Integer.
    valueOf(e_list.get(0).get(1));index++){
    double rate=Double.valueOf(e_list.get(0).get(2));
    temp_cost=temp_cost*(1+(rate/100));
    if(e_pattern.getKey().equals("Hour")){
        if(temp_cost>730){
            temp_cost=730;
        }
    }
    if(e_pattern.getKey().equals(compare_var)){
        if(!(temp_cost<=max & & temp_cost>=min)){
            if(rate>0){
                for(int start=varsList_index;start<varsList.size();start++){
                    min=Double.valueOf(coeff_map.get(String.valueOf(varsList.get(
                        start))).get(4));
                    max=Double.valueOf(coeff_map.get(String.valueOf(varsList.get(
                        start))).get(4));
                }
            }
        }
    }
}
```
if( temp_cost <= max && temp_cost >= min) {
    coefficientid=varsList.get(start);
data_ids=this.coeff_datatransferList.get(coefficientid);
costString=coeff_map.get(String.valueOf(varsList.get(start))).get(0);
    varsList_index=start;
    break;
}
else{
    for (int start = varsList_index; start >= 0; start--){
        min=Double.valueOf(coeff_map.get(String.valueOf(varsList.get(start))).get(4));
        max=Double.valueOf(coeff_map.get(String.valueOf(varsList.get(start))).get(5));
        if( temp_cost <= max && temp_cost >= min) {
            coefficientid=varsList.get(start);
data_ids=this.coeff_datatransferList.get(coefficientid);
costString=coeff_map.get(String.valueOf(varsList.get(start))).get(0);
            varsList_index=start;
            break;
        }
    }
}
varvalue_month.add(temp_cost);
coststring_list.add(costString);
data_dataid.add(data_ids);
for (int i = 1; i < e_list.size(); i++) {
    if( Integer.valueOf(e_list.get(i).get(0)) - Integer.valueOf(e_list.get(i-1).get(1)) != 1) {
        for (int index = Integer.valueOf(e_list.get(i-1).get(1)) + 1; index < Integer.valueOf(e_list.get(i).get(0)); index++) {
            varvalue_month.add(temp_cost);
coststring_list.add(costString);
data_dataid.add(data_ids);
        }
    }
    double rate=Double.valueOf(e_list.get(i).get(2));
temp_cost=temp_cost*(1+(rate/100));
    if(e_pattern.getKey().equals("Hour")){
        if(temp_cost>730){
            temp_cost=730;
        }
    }
    if(e_pattern.getKey().equals(compare_var)) {
        if(!(temp_cost <= max && temp_cost >= min)){
            if(rate>0){
                for (int start = varsList_index; start < varsList.size(); start++){
                    min=Double.valueOf(coeff_map.get(String.valueOf(varsList.get(start))).get(4));
                    max=Double.valueOf(coeff_map.get(String.valueOf(varsList.get(start))).get(5));
                    if(temp_cost<=max && temp_cost>=min){
                        coefficientid=varsList.get(start);
data_ids=this.coeff_datatransferList.get(coefficientid);
costString=coeff_map.get(String.valueOf(varsList.get(start))).get(0);
                        varsList_index=start;
                        break;
                    }
                }
            }
        }
    }
}
if(temp_cost <= max && temp_cost >= min){
    coefficientid = varsList.get(start);
    data_ids = this.coeff_datatransferList.get(coefficientid);
    costString = coeff_map.get(String.valueOf(varsList.get(start))).get(0);
    varsList_index = start;
    break;
}
}
}
}
else{
    for(int start = varsList_index; start >= 0; start--){
        min = Double.valueOf(coeff_map.get(String.valueOf(varsList.get(start))).get(4));
        max = Double.valueOf(coeff_map.get(String.valueOf(varsList.get(start))).get(5));
        if(temp_cost <= max && temp_cost >= min){
            coefficientid = varsList.get(start);
            data_ids = this.coeff_datatransferList.get(coefficientid);
            costString = coeff_map.get(String.valueOf(varsList.get(start))).get(0);
            varsList_index = start;
            break;
        }
    }
}
}
varvalue_month.add(temp_cost);
coststring_list.add(costString);
data_dataid.add(data_ids);
}
}
else{
    for(int index = Integer.valueOf(e_list.get(i).get(0)); index <= Integer.valueOf(e_list.get(i).get(1)); index++){
        double rate = Double.valueOf(e_list.get(i).get(2));
        temp_cost = temp_cost *(1+(rate/100));
        if(e_pattern.getKey().equals("Hour")){
            if(temp_cost > 730){
                temp_cost = 730;
            }else{
                if(e_pattern.getKey().equals(compare_var)){
                    if(!!(temp_cost <= max && temp_cost >= min)){
                        for(int start = varsList_index; start < varsList.size(); start++){
                            min = Double.valueOf(coeff_map.get(String.valueOf(varsList.get(start))).get(4));
                            max = Double.valueOf(coeff_map.get(String.valueOf(varsList.get(start))).get(5));
                            if(temp_cost <= max && temp_cost >= min){
                                coefficientid = varsList.get(start);
                                data_ids = this.coeff_datatransferList.get(coefficientid);
                                costString = coeff_map.get(String.valueOf(varsList.get(start))).get(0);
                                varsList_index = start;
                                break;
                            }
                        }
                    }
                }
            }
        }
    }
}
4 Implementation

}{
} else {
    for (int start = varsList_index; start >= 0; start--){
        min = Double.valueOf(coeff_map.get(String.valueOf(varsList.get(start))).get(4));
        max = Double.valueOf(coeff_map.get(String.valueOf(varsList.get(start))).get(5));
        if (temp_cost <= max && temp_cost >= min){
            coefficientid = varsList.get(start);
            data_ids = this.coeff_datatransferList.get(coefficientid);
            costString = coeff_map.get(String.valueOf(varsList.get(start))).get(0);
            varsList_index = start;
            break;
        }
    }
}
}

varvalue_month.add(temp_cost);
coststring_list.add(costString);
data_dataid.add(data_ids);
}
}

if (Integer.valueOf(e_list.get(e_list.size()-1).get(1))<month) {
    for (int index = Integer.valueOf(e_list.get(e_list.size()-1).get(1)) + 1; index <= month; index++) {
        varvalue_month.add(temp_cost);
        coststring_list.add(costString);
        data_dataid.add(data_ids);
    }
}

4.4 Decision Support System MiDSuS

This decision support system named MiDSuS is built up by a web application interface with RESTful APIs which were already done in Nefolog system. The web application interface supports a friendly interaction between users and system.

This frontend interface is developed in JSP environment and used JS, CSS, JAVA and HTML together. The main page contains 5 subpages, one introduction subpage and 4 migration types’ projects subpages. In each Migration Type, there are 4 inner subpages to achieve actually 4 steps to find the final offerings. These 4 steps are the first step collection of candidate offerings’ requirements, the second step search candidate offerings, the third step collection of cost calculation’s requirements and the last step calculation of each configuration’s cost.
As discussed in Section 3.5, the decision process starts from the selection of migration types. Figure 4.3 and Figure 4.4 show the user interfaces of first two types. In these two figures it can be seen that either service types or application layers are divided by some application components.

<table>
<thead>
<tr>
<th>MigrationDecisionSupportSystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
</tr>
<tr>
<td>Type I</td>
</tr>
<tr>
<td>Type II</td>
</tr>
<tr>
<td>Type III</td>
</tr>
<tr>
<td>Type IV</td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Figure 4.3: User Interface of MiDSuS: Type I**

<table>
<thead>
<tr>
<th>MigrationDecisionSupportSystem</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduction</td>
</tr>
<tr>
<td>Type I</td>
</tr>
<tr>
<td>Type II</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Type III</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>Type IV</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td></td>
</tr>
</tbody>
</table>

**Figure 4.4: User Interface of MiDSuS: Type II**

Decision support for different migration types of applications to the Cloud
Figure 4.5: Subdivision of MiDSuS

Figure 4.5 explains the mappings between application components and service types, application components and application layer. In this figure, one application is divided into 3 layers. In each layer, there is at least one mapped service type. Besides these 3 layers, there is also a cross layer. This layer contains the service types which support networking between layers and monitoring of components across layers. In each service type, at least one application component is defined. It helps users more easily to find the mapping between their migrated components and service types. Some of these application components are defined by the Cloud services providers. Some are from the applications use case in Appendix A.1. All the defined application components in Figure 4.5 are listed with their provenances in Table 4.5. Actually, if the migrated components or application layers are identified, it means the service types are identified.
Table 4.5: Provenances of each Application Component in MiDSuS

<table>
<thead>
<tr>
<th>Application Component</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application Server</td>
<td>Payroll Processing [8] Ice.com [23]</td>
</tr>
<tr>
<td>Resizable Compute Capacity</td>
<td>HP Cloud Compute [24]</td>
</tr>
<tr>
<td>Failure Resilient Application</td>
<td>AWS EC2 [26]</td>
</tr>
<tr>
<td>Testing Application</td>
<td>Application testing [25]</td>
</tr>
<tr>
<td>RDBMS</td>
<td>-</td>
</tr>
<tr>
<td>Distributed Database</td>
<td>AWS DynamoDB [27]</td>
</tr>
<tr>
<td>Caching System</td>
<td>-</td>
</tr>
<tr>
<td>Raw Block Level Storage</td>
<td>HP Cloud Block Storage [28] AWS EBS [29]</td>
</tr>
<tr>
<td>Expandable File System</td>
<td>HP Cloud Block Storage [28]</td>
</tr>
<tr>
<td>Media Data</td>
<td>Windows Azure Blob Storage [30]</td>
</tr>
<tr>
<td>Backup Data</td>
<td>AWS Glacier [31]</td>
</tr>
<tr>
<td>Archival Data</td>
<td>AWS Glacier [31]</td>
</tr>
</tbody>
</table>

In order to have a better idea about inner 4 steps in each migration type, the selection process of Type I is chosen to explain as an example. In Figure 4.3 whose tasks are to select an appropriate migration type and the desired provider, the migrated application components are resizable compute capacity which belongs to infrastructure service type and RDBMS which belongs to SQL database service type. At the same time user chooses AWS provider for RDBMS component.
After Figure 4.3, system runs to the first inner page which is called the first step of Type I to collect the user’s requirements of offerings’ performances in Figure 4.6. Following the selection of previous page, components which are from two service types are ready to be migrated to the Cloud. In addition, user can use "Add" button to increase any performances with certain values while use "Delete" button in the end of each performance row to decrease the performance in this row. In this example, some non-numerical and numerical performances for both service types are entered. Then system goes to the candidate search step.

In Figure 4.7, the candidate offerings with providers are listed and with the help of "Info" button user can have a complete look of this configuration. In this example, the first candidate configuration of infrastructure service type has 8 CPU cores with 2.6 GHz CPU speed and 52 GB of memory, 3540 GB of local storage instance, and 99.95% SLA. User selects first 5 configurations of each service type to calculate the costs.

**Figure 4.6: User Interface: First Step of Type I**

Decision support for different migration types of applications to the Cloud
Here are the candidate configurations!

Select configurations for each service type whose costs will be calculated later.

Configuration for infrastructure

<table>
<thead>
<tr>
<th>Select</th>
<th>Candidate Configuration</th>
<th>Provider</th>
<th>Info</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Google</td>
<td>Info</td>
</tr>
</tbody>
</table>

Figure 4.7: User Interface: Second Step of Type I Candidate Search Step
4 Implementation

Figure 4.8: User Interface: Third Step of Type I

In Figure 4.8, system starts to collect the requirements of cost calculation. User can also use "Add" button to increase requirement variables with certain values and use "Delete" button to decrease one row. This design idea is similar as the first step design. The usage variables of the tables in Figure 4.8 are generally separated into static rate and dynamic rate. The infrastructure service type has a usage variable called "usage_pattern". This variable means that a variable will be changed during a period time with a certain rate (this rate cannot be zero). This situation is called dynamic rate. The other service type SQL database is no "usage_pattern" variable, so it is in static rate which means rate is equal to zero. In this situation, cost calculator will also calculate each month cost with invariant rate. It is important to say one more time that these necessary usage variables which should be valued in the beginning but here no-values from the user have default values which has already been talked in Section 4.3.2.

After the third step which collects all the necessary information, system comes to the last step in Figure 4.9 which gives the results of this migrated application. Clicking the "Details" button user can see the trend of this configuration in each month, the usage variables in this cost calculation and the features of this offerings. The Figure 4.10 shows the details of the first offering. User can also use the "Ranking" drop down list to choose a parameter to make a ranking. This helps user to have a better look which candidate offering(s) is the most worthy to invest.
### Migration Decision Support System

#### Cost of each configuration which was selected before according to different service type!

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Data Center</th>
<th>Total Cost</th>
<th>Service Cost</th>
<th>Data Transfer Cost</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small Office 1</td>
<td>$6,500.00</td>
<td>$2,000.00</td>
<td>$4,500.00</td>
<td>$1,000.00</td>
<td>Details</td>
</tr>
</tbody>
</table>

#### Configuration for applications

<table>
<thead>
<tr>
<th>Configuration</th>
<th>Data Center</th>
<th>Total Cost</th>
<th>Service Cost</th>
<th>Data Transfer Cost</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small Office 1</td>
<td>$6,500.00</td>
<td>$2,000.00</td>
<td>$4,500.00</td>
<td>$1,000.00</td>
<td>Details</td>
</tr>
</tbody>
</table>

**Figure 4.9: User Interface: Fourth Step of Type I**

Decision support for different migration types of applications to the Cloud
4.5 User Guide

This MiDSuS system works according to user’s migrated application, some requirements of the available Cloud offerings’ performances and some restrictions of usage variables’ values.

4.5.1 Selection of Migration Type

The relationships of each migration type are dispersed in this work, e.g.: definitions of each type are in Section 2.1; identical and different parts of each two types are explained in Section 3.5 and also in the section above. Figure 4.5 show a clear division of an application and with the help of this figure migration types based on application components are like Figure 4.3 while migration types based on application layers are like Figure 4.4.

4.5.2 Requirement Collection of Candidate Offerings

The requirements are collected by a set of performance parameters. All the available parameters are summarized by a drop down list in Figure 4.6. The addition and delete operations are achieved by button to get a dynamic table. The descriptions and units of some commonly used parameters are explained in the following:

- CPU Core: number of virtual cores
4 Implementation

- CPU Speed: speed of CPU with GHz
- Memory: size of RAM with GB
- IO: I/O performance with 4 values (very high, high, moderate and low)
- Storage: local disk with GB
- SLA: Service-Level Agreement with a [0, 1] value
- Licence: software licence with 5 values (MySQL, Oracle, SQLServer, SQLWeb and SQLStandard)
- Platform: 32 or 64-bit
- OS: operating system with 3 values (Linux, Windows and Enterprise Linux)

When entering the non-numerical performance characteristics (IO, Licence, OS), user should be aware that the values are case sensitive. About the no-value performance parameter on both the database side and the user side, it is thought as default that this performance satisfies the requirement.

4.5.3 Selection of Candidate Offerings

According to the requirements in the above section, all the candidate offerings with corresponding provider are listed in a table in Figure 4.7. With the “Info” button user can see a table with all the information about this offering’s performances. Furthermore, a set of offerings are selected by the check box to be calculated in next steps. This is the first filter of candidate offerings.

4.5.4 Collection Requirements of Cost Calculator

The cost calculator uses a cost formula with multiple parameters to calculate the cost. These parameters should be provided by user in Figure 4.8. If there is no input value, some default values are set up for all parameters. In Figure 4.8 all the usage variables are summarized in a drop down list and user can use "Add" and "Delete" button to control the number of variables. The default values are talked in Section 4.3.2. If the parameter is “Hour”, the default value is 500 hours per month and the input value should not be more than 730 hours per month; if the parameter is "month", the default value is 10 months; All of the other parameters have a same default value of 5000. As said in last section parameter ”usage_pattern” is used to set up a dynamic rate situation. The form of this parameter should be treated carefully. This variable should be valued in a form like \((Hour, start = 2, end = 6, rate = -20)\), it means the variable “Hour” will decreased between the second month and the sixth month with a rate value −20%. User can add such ”usage_pattern” as many times as he wants.
4.5.5 Checking Results of Cost Calculator and Ranking

Figure 4.9 shows the results page of MiDSuS. It collects results of each offering together in a table and at the end of each row there is a control button called ”Details” to see the cost trends in each month, the information about usage variables and a list of performance characteristics of this offering. User can also use Ranking drop down list to check a ranking with update date, data center, total cost, upfront cost, service cost or data transfer cost. With the help of ranking user can know which is the cheapest or which is the newest. These results will support a better comparison for user to make a correct decision.
5 Evaluation

In this chapter, a set of tests is used to validate these web services in Nefolog. Especially for the candidate search service and cost calculator service, the results of them are compared with the results which are calculated by some other available decision support systems. Some use cases in Appendix A.1 are migrated to the Cloud with different migration types in MiDSuS in order to demonstrate that with MiDSuS user can find the best cost for his migration project.

5.1 Nefolog Validation

In Nefolog system, the two decision support services are candidate search service and cost calculator service. The available decision support systems among several Cloud providers with cost calculator are PlanForCloud [9] and Cloudorado [17] which have already been discussed in Section 2.4.

The candidate search service begins with requirements collection. These requirements contain all the available features. There are totally 16 features in Cloud provider knowledge base of Nefolog. However, PlanForCloud and Cloudorado do not support as many features as Nefolog. In PlanForCloud system, only during the selection of server and database user has chances to enter his requirements to control the search results. In addition, Cloudorado system supports only IaaS Cloud service search with 4 features. In order to have a valid comparison between each other, an infrastructure service type with CPU cores and size of RAM is chosen as an example to migrate to the Cloud. The provider data bases of both systems have several same providers. After considering Nefolog system with PlanForCloud and Cloudorado from features of services and the status of data bases perspectives, a set of requirements is set as following:

- **Service Type** = *Infrastructure*
- **CPU Core** ≥ 8
- **RAM** ≥ 8GB
- **Provider** = *AWS or Rackspace*

The results of each system are listed in Table 5.1. Comparing the results of Nefolog with PlanForCloud, they present exactly the same search results of provider AWS but the light, medium and heavy utilization reserved instance of AWS EC2 are separated by different offering names in Nefolog while these different reserved instance types are selected in purchase option step in PlanForCloud. In addition, the results of provider Rackspace are almost the same. Both of them have a Cloud offering called 30GB, but
### Table 5.1: Comparison of Candidate Offerings Search

<table>
<thead>
<tr>
<th>Provider</th>
<th>Nefolog</th>
<th>PlanForCloud</th>
<th>Cloudorado</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>m1.xlarge</td>
<td>m1.xlarge (Standard, light, medium, heavy)</td>
<td>Standard Extra Large</td>
</tr>
<tr>
<td></td>
<td>m1.xlarge light utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>m1.xlarge medium utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>m1.xlarge heavy utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>m2.2xlarge</td>
<td>m2.2xlarge (Standard, light, medium, heavy)</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>m2.2xlarge light utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>m2.2xlarge medium utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>m2.2xlarge heavy utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>m2.4xlarge</td>
<td>m2.4xlarge (Standard, light, medium, heavy)</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>m2.4xlarge light utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>m2.4xlarge medium utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>m2.4xlarge heavy utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>m3.xlarge</td>
<td>m3.xlarge (Standard, light, medium, heavy)</td>
<td>Standard Second Generation Extra Large</td>
</tr>
<tr>
<td></td>
<td>m3.xlarge light utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>m3.xlarge medium utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>m3.xlarge heavy utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>m3.2xlarge</td>
<td>m3.2xlarge (Standard, light, medium, heavy)</td>
<td>Standard Second Generation Double Extra Large</td>
</tr>
<tr>
<td></td>
<td>m3.2xlarge light utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>m3.2xlarge medium utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>m3.2xlarge heavy utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cc1.4xlarge</td>
<td>cc1.4xlarge (Standard, light, medium, heavy)</td>
<td>Cluster Compute Quadruple Extra Large</td>
</tr>
<tr>
<td></td>
<td>cc1.4xlarge light utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cc1.4xlarge medium utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cc1.4xlarge heavy utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cc2.8xlarge</td>
<td>cc2.8xlarge (Standard, light, medium, heavy)</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>cc2.8xlarge light utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cc2.8xlarge medium utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cc2.8xlarge heavy utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cg1.4xlarge</td>
<td>cg1.4xlarge (Standard, light, medium, heavy)</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>cg1.4xlarge light utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cg1.4xlarge medium utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cg1.4xlarge heavy utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cr1.8xlarge</td>
<td>cr1.8xlarge (Standard, light, medium, heavy)</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>cr1.8xlarge light utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cr1.8xlarge medium utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>cr1.8xlarge heavy utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>hl1.4xlarge</td>
<td>hl1.4xlarge (Standard, light, medium, heavy)</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>hl1.4xlarge light utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>hl1.4xlarge medium utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>hl1.4xlarge heavy utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ha1.8xlarge</td>
<td>ha1.8xlarge (Standard, light, medium, heavy)</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td>ha1.8xlarge light utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ha1.8xlarge medium utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>ha1.8xlarge heavy utilization</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Rackspace 30GB (Windows)</td>
<td>30GB (Windows, Linux) server</td>
<td>30 GB RAM Instance</td>
</tr>
<tr>
<td></td>
<td>30GB (Linux)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>30GB (Windows+SQL Standard)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>30GB (Windows+SQL Web)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>30GB (Enterprise Linux)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Decision support for different migration types of applications to the Cloud
Nefolog includes 5 combinations of different operating systems and licences while PlanForCloud contains only the option of operating systems. Comparing with Cloudrado, the results of Cloudrado do not cover all the Cloud offerings but only some standard offerings. Furthermore, the CUP core calculator is different from the other two systems. As a result of these deficiencies, Cloudrado is not to be considered in the comparison of Nefolog evaluation.

It can be seen from results of comparison with Nefolog and PlanForCloud that the results of candidate offerings search service in Nefolog cover all the results of PlanForCloud and even more. It means this candidate search service supports a sufficient, useful and practical result.

Table 5.2: Static Rate Test Comparing with PlanForCloud

<table>
<thead>
<tr>
<th>Provider</th>
<th>Offerings</th>
<th>Configuration</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon Web Service</td>
<td>RDS</td>
<td>Large Medium Utilization Reserved Instance Multi-AZ Deployment</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Location</th>
<th>Upfront cost once</th>
<th>Service per Month</th>
<th>Data transfer per Month</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nefolog</td>
<td>PlanForCloud</td>
<td>Nefolog</td>
</tr>
<tr>
<td>Oregon</td>
<td>$2000</td>
<td>$2000</td>
<td>$1052.80</td>
</tr>
<tr>
<td>SaoPao</td>
<td>$3780</td>
<td>$3780</td>
<td>$1987.84</td>
</tr>
<tr>
<td>N.California</td>
<td>$2000</td>
<td>$2000</td>
<td>$1167.68</td>
</tr>
<tr>
<td>Tokyo</td>
<td>$2100</td>
<td>$2100</td>
<td>$1278.24</td>
</tr>
<tr>
<td>Singapore</td>
<td>$2000</td>
<td>$2000</td>
<td>$1167.68</td>
</tr>
<tr>
<td>Ireland</td>
<td>$2000</td>
<td>$2000</td>
<td>$1167.68</td>
</tr>
<tr>
<td>NorthernVirginia</td>
<td>$2000</td>
<td>$2000</td>
<td>$1052.80</td>
</tr>
<tr>
<td>Sydney</td>
<td>$2000</td>
<td>$2000</td>
<td>$1167.68</td>
</tr>
</tbody>
</table>

After that, the cost calculator service should also be evaluated too and compared with PlanForCloud. 3 test cases which were recorded on the end of June 2013 are implemented in order to cover all the situations of cost calculator. The different situation is cause of different rates of usage variable, static rate and dynamic rate.

The results in Table 5.2 which is calculated in the static rate situation are listed with the results of PlanForCloud together. The information in URI means that the configuration id is equals to 122; the usage hour is 240 hours per month; the I/O operation number is 5000; the storage size is 5000GB; the data transfer out from this offering is 50000 GB per month; the total usage time is 15 months. From the database, it can be known that this offering is a relational database service which is provided by AWS and it is a large medium utilization reserved instance multi-AZ deployment. This offering has 8
different locations of data center. Because of the static rate the cost of each month stays the same, so only the upfront cost, service cost and data transfer cost are essential to be compared with different locations. Comparing with upfront cost, both of them have the exactly same result. Comparing with service cost, the maximum difference is $0.14 in SaoPaulo among different locations and the ratio of this difference is about 0.007%. Comparing with data transfer cost, the maximum different is $10.12 in Tokyo and the ratio is about 0.12%. Generally speaking, the difference between two systems is tiny and not over 0.2%. This difference maybe occurs during the identification of different usage amount ranges. To describe this difference clearly, for example in data transfer cost calculation, the cost in Nefolog is a little smaller than in PlanForCloud. It is known from the provider that the data transfer out from RDS offering is free in the first 1 GB. After that, it should be charged. Maybe PlanForCloud does not remove this free part. Therefore PlanForCloud has a little higher price than Nefolog.

Table 5.3: Dynamic Rate Test 1 Comparing with PlanForCloud

<table>
<thead>
<tr>
<th>Provider</th>
<th>Offering</th>
<th>Configuration</th>
<th>Location: worldwide</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>worldwide</td>
</tr>
<tr>
<td></td>
<td>Nefolog</td>
<td>PlanForCloud</td>
<td>Nefolog</td>
</tr>
<tr>
<td>1th Month</td>
<td>$0</td>
<td>$0</td>
<td>$460.00</td>
</tr>
<tr>
<td>2th Month</td>
<td>-</td>
<td>-</td>
<td>$572.5</td>
</tr>
<tr>
<td>3th Month</td>
<td>-</td>
<td>-</td>
<td>$713.13</td>
</tr>
<tr>
<td>4th Month</td>
<td>-</td>
<td>-</td>
<td>$888.91</td>
</tr>
<tr>
<td>5th Month</td>
<td>-</td>
<td>-</td>
<td>$1108.63</td>
</tr>
<tr>
<td>6th Month</td>
<td>-</td>
<td>-</td>
<td>$1383.29</td>
</tr>
<tr>
<td>7th Month</td>
<td>-</td>
<td>-</td>
<td>$1726.61</td>
</tr>
<tr>
<td>8th Month</td>
<td>-</td>
<td>-</td>
<td>$2155.77</td>
</tr>
<tr>
<td>9th Month</td>
<td>-</td>
<td>-</td>
<td>$2692.21</td>
</tr>
<tr>
<td>10th Month</td>
<td>-</td>
<td>-</td>
<td>$3362.76</td>
</tr>
<tr>
<td>11th Month</td>
<td>-</td>
<td>-</td>
<td>$4200.95</td>
</tr>
<tr>
<td>12th Month</td>
<td>-</td>
<td>-</td>
<td>$5207.65</td>
</tr>
</tbody>
</table>

In the dynamic rate situation, 2 test cases are implemented to make sure that Nefolog system has also valid results in other providers excepted AWS. Table 5.3 shows the results of provider Rackspace while Table 5.4 shows the results of provider Hp Cloud. The usage patterns in PlanForCloud are fixed. It means that user can choose only the given usage
patterns with special variables. Because of this special rule, the usage patterns of the first test case in Table 5.3 contains the trends of GB and GBExternalNetworkEgress. GB is increased by 20% from the first month to 12th month and GBExternalNetworkEgress is decreased by 10% from the first month to 12th month. The location of Cloud Files’ data center is worldwide. The comparisons between two systems are listed by each month. Although the requirement month is 5 months but the total month in usage pattern is 12 months, the final calculated month chooses the longer one. The comparison of upfront cost is similar and the result is that there is no upfront cost of the Cloud Files. The results of service cost are almost the same. The biggest difference is $6.27 in the 12th month and the ratio is about 0.12%. The difference of comparison results of data transfer cost is relatively small. The maximum one is $0.11 in the 9th month and also in the 12th month and the ratio is about 0.058%. Table 5.4 is implemented with the same method like what did in the first test case. This offering called Object Storage is provided by Hp Cloud and also has no upfront cost with a worldwide data center location. The maximum difference of service cost is $0.08 in the 3th month and the 4th month with a ratio of 0.015%. The results of data transfer cost are exactly the same.

Table 5.4: Dynamic Rate Test 2 Comparing with PlanForCloud

<table>
<thead>
<tr>
<th>Provider</th>
<th>Hp Cloud</th>
</tr>
</thead>
<tbody>
<tr>
<td>Offering</td>
<td>Object Storage</td>
</tr>
<tr>
<td>Configuration</td>
<td>Object Storage</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Location: worldwide</th>
<th>Upfront cost once</th>
<th>Service per Month</th>
<th>Data transfer per Month</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Nefolog</td>
<td>PlanForCloud</td>
<td>Nefolog</td>
</tr>
<tr>
<td>1st Month</td>
<td>$0</td>
<td>$0</td>
<td>$450.01</td>
</tr>
<tr>
<td>2nd Month</td>
<td>-</td>
<td>-</td>
<td>$495.01</td>
</tr>
<tr>
<td>3rd Month</td>
<td>-</td>
<td>-</td>
<td>$544.51</td>
</tr>
<tr>
<td>4th Month</td>
<td>-</td>
<td>-</td>
<td>$598.96</td>
</tr>
<tr>
<td>5th Month</td>
<td>-</td>
<td>-</td>
<td>$658.85</td>
</tr>
<tr>
<td>6th Month</td>
<td>-</td>
<td>-</td>
<td>$724.73</td>
</tr>
<tr>
<td>7th Month</td>
<td>-</td>
<td>-</td>
<td>$797.21</td>
</tr>
<tr>
<td>8th Month</td>
<td>-</td>
<td>-</td>
<td>$876.93</td>
</tr>
<tr>
<td>9th Month</td>
<td>-</td>
<td>-</td>
<td>$964.62</td>
</tr>
<tr>
<td>10th Month</td>
<td>-</td>
<td>-</td>
<td>$1061.08</td>
</tr>
<tr>
<td>11th Month</td>
<td>-</td>
<td>-</td>
<td>$1167.19</td>
</tr>
<tr>
<td>12th Month</td>
<td>-</td>
<td>-</td>
<td>$1283.91</td>
</tr>
</tbody>
</table>

It can be seen from the results that the maximum ratio of difference between Nefolog and PlanForCloud is smaller than 1%. It means that these two systems have an almost
similar cost result. Although there could also be some deviations between the cost result which is calculated by the provider itself and the cost results from these two systems, the results of Nefolog can be thought as the closest results as the exact ones.

5.2 MiDSuS Evaluation

The MiDSuS evaluation is realized by estimating the cost of re-migrating an application in Appendix [A.1] with different migration types in order to demonstrate that with the help of MiDSuS user can find the best cost of his migration project.

The application payroll processing which has been migrated to the Cloud with Migration Type III already is chosen as a test case here. This application contains one application component and one DBMS and all of them are migrated to the Cloud with 5 VMs. In this test case, Migration Type IV is chosen for this application instead of Type III. The data layer and business layer are migrated to the Cloud. Specifically, the application server and RDBMS are selected to migrate to the Cloud with any of the available providers.

<table>
<thead>
<tr>
<th>Migration Types</th>
<th>Type III</th>
<th>Type IV</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Requirements</strong></td>
<td>For application: CPU cores = 8, RAM = 8 GB, I/O = high, OS = Windows, App = 1000; Hour = 240 hours, Storage = 1000 GB, Month = 1 For DBMS: CPU cores = 8, RAM = 8 GB, I/O = high, Storage = 1000 GB, OS = Windows; Hour = 240 hours, Storage = 1000 GB, Month = 1</td>
<td></td>
</tr>
<tr>
<td><strong>Cloud Offerings</strong></td>
<td>VM for application server</td>
<td>VM for DBMS</td>
</tr>
<tr>
<td><strong>Total Cost Range</strong></td>
<td>$768 - $14900</td>
<td>$1196.8 - $7658.17</td>
</tr>
<tr>
<td><strong>Cost Range</strong></td>
<td>$153.6 - $2980.0</td>
<td>$153.6 - $2980.0</td>
</tr>
<tr>
<td><strong>Minimum Cost</strong></td>
<td>xlarge by Windows Azure</td>
<td>xlarge by Windows Azure</td>
</tr>
<tr>
<td><strong>Maximum Cost</strong></td>
<td>Server8-8 by Flexiscale</td>
<td>Server8-8 by Flexiscale</td>
</tr>
</tbody>
</table>

The results of these two migrations are listed in Table 5.5. There are more than 40 candidate offerings for each migration type. Only the minimum and maximum costs are
listed in this table with their offering’s names and providers. Because the details of this payroll processing migration are not known, a set of requirements is identified casually but must be kept the same all the time during both of these migration types. The requirements of both candidate offerings search and cost calculator are listed in Table 5.5. Comparing with Type III and Type IV, it can be seen that with the same budget user can choose either VMs services or corresponding Cloud services. Even he chooses a corresponding Cloud offering, the result could not be over budget. In addition, user can also know the total minimum cost of his whole project between different migration types from the table. But it should be not forgotten that with different migration types the risks and any other problems during each migration process are not considered in MiDSuS. All the results are presented from the effective cost’s respective that with the help of MiDSuS user can find the most effective cost and minimum cost of the whole project among different offerings combinations easily and directly. Furthermore, if user has a favorite provider or location of data center, MiDSuS is more helpful.
6 Conclusions

6.1 Summary

Some small or medium size enterprises who may have to face a shortage of funds or weak of against the loss during investment focus on Cloud services to reduce the cost and risks. In addition, more and more providers support various Cloud services in order to satisfy more customers’ requirements. A migration decision support system which focuses on finding a trade-off between cost evaluation and performances prediction among these different Cloud services is very necessary to help user to find an efficient Cloud service.

First of all, some related works and researches are discussed as foundation in order to get a general understanding of the state of decision support for Cloud migration. There are many factors from both user side and provider side that can influence the final decision. It is clear to see that the decision process consists of many decisions according to comparisons between user’s requirements and offerings’ information. All these related decision support systems which were mentioned in this work focus on the performances comparisons and effective costs comparisons.

More specifically, this work is based on a foundational system called MDSS which achieved the match offerings selection and cost calculations with an interaction between user and system. The system discussed in this work is aimed to implement these functionalities as web APIs which are exposed as web services. These decision support services which are collected in a system called Nefolog achieve tasks such as candidate offerings search and total cost calculations. Nefolog works in conjunction with a Cloud provider knowledge base. This Cloud provider knowledge base contains more providers and more service types than MDSS. The functions which were already done in MDSS are re-engineered as services in Nefolog. As a result, the offerings matcher function is transplanted to Nefolog with a little change while the costs calculator function is completely rewritten. Each service in Nefolog has two representations, XML and JSON data format with related information.

Furthermore, these APIs can be used in a migration decision support system to achieve the data handler part which was implemented in MDSS. In this work, a migration decision support system called MiDSuS is implemented to support the decision making process. The frontend of MiDSuS is a friendly user interface which is built on JSP with Servlet. The migration project starts from migration type’ selection. The migrated application is distributed by different migration types into application components or application layers. After that, the system collects the user’s requirements of candidate offerings’ search and a web service is used to achieve this search and lists all the candidate offerings with providers and performance information back to user. Next, the system summarized the user’s requirements of usage amount of certain variables. With the
help of the cost calculator web service, all the results which contains update dates, geographical areas of data center, total costs, upfront costs, service costs, data transfer costs and trends of costs in each month are calculated and summed for each configuration option. A user can use a type of ranking to decide which offering(s) is the appropriate one for him.

To validate the correctness and viability of Nefolog and MiDSuS, the results of Nefolog are compared with costs by PlanForCloud under the same input. To evaluate the MiDSuS, a use case in Appendix A.1 is implemented by different migration types and comparing with each type to demonstrate that with the help of MiDSuS user can find the effective cost for his migration project.

In the meantime, there are also some deficiencies and necessary improvements needed of MiDSuS. All these limitations and improvement suggestions are summarized in the following section and some of them may be useful and helpful for the future work.

### 6.2 Future Work

Firstly, the MiDSuS system lacks a dedicated knowledge base management system. It is known that technology is developing inconceivably fast. In order to meet customers’ needs and survive in the competition, Cloud providers would provide much more favorable service pricing to get more customers from the other providers. For example, some providers launch free try events in a short time, if user selects these providers in that time, the cost will be lower than usually. In this case, the pricings of these Cloud offerings are changed often. If this decision support system is hoped to get a valid and valuable result, all data in database must be updated to the latest.

Secondly, there are various Cloud offerings in the market now that cover almost all service types. In future work, with the growth of database, more and more offerings will be increased into the database, the service types should be refined to get a more detailed management. The meticulous classification of services is essential. It supports more comparability with each offering which belongs to the same service type and provides more choices to user to select.

Thirdly, this system collects 16 performance characteristics for each offering to support a comparison with user’s requirements. Actually, there are many other features which can be thought as a performance in this system. These features may not be abstracted directly, for example, trust, risk, etc. In addition, some of these features may play different roles with different users. For example, a user wants to choose a provider and he cares about what degree of risk the offerings have rather than what degree of trust reputation this provider is; but another one may be want to choose a good reputation provider in order to make sure that all his investments and stored data are in a safety statement. An additional design of this system should be considered that user’s desires can also influenced the result of the decision and sometimes it may be stronger than the numerical information comparisons. This design should be built on a lot of data collections and case studies which focus on finding a way to represent these abstract features.
Fourthly, it can be considered that the desired performance characteristics as expressed by the user as requirements can also be dynamic \[5\]. It means that the storage capacities could be increased or decreased in each month. In the candidate offerings’ requirements collection step of the further system, a requirement called also usage pattern can be added to expand this requirements collection into a dynamic situation.

Fifthly, MiDSuS system is now searching and calculating without login step. It means that the search results and calculation results of this user cannot be saved in this system. These results are refreshed and deleted after the migrated project is finished. It is better to add a login step in order to help user to compare with all his migrated projects with customized name and select the best one among them.

Finally, in the cost calculation step, MiDSuS suffers in performance time. It depends on the number of configurations. The calculation time for each normal configuration is 2 seconds (this configuration has only one data center) to 5 seconds (this configuration has more than one geographical areas of data center). It means the more configuration user selects the more slowly system runs. A solution is possible that each selected configurations can be assigned with one thread. It means that system could calculate each configuration’s cost parallelly. In MDSS, the trend of cost is shown in a diagram with spline. It is more clearly than a list in MiDSuS which contains costs of each month. It can be improved by the further work.
Appendix

A.1 Use Case of Applications on Cloud

Table A.1: Application Migration Use Cases

<table>
<thead>
<tr>
<th>Application</th>
<th>Use Case Documentation</th>
<th>Service Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payroll Processing</td>
<td>[8]</td>
<td>A A A A Y</td>
</tr>
<tr>
<td>Logistics &amp; Project Management</td>
<td>[8]</td>
<td>Y Y A A A</td>
</tr>
<tr>
<td>Central Government Services</td>
<td>[8]</td>
<td>A Y</td>
</tr>
<tr>
<td>Local Government Services</td>
<td>[8]</td>
<td>Y</td>
</tr>
<tr>
<td>Astronomic Data Processing</td>
<td>[8]</td>
<td>Y Y Y A</td>
</tr>
<tr>
<td>Application testing</td>
<td>[25]</td>
<td>Y</td>
</tr>
<tr>
<td>Application contingency</td>
<td>[25]</td>
<td>Y A</td>
</tr>
<tr>
<td>&quot;Big&quot; data processing</td>
<td>[25]</td>
<td>A Y</td>
</tr>
<tr>
<td>Abaca</td>
<td>[32]</td>
<td>Y</td>
</tr>
<tr>
<td>Banro</td>
<td>[33]</td>
<td>Y Y</td>
</tr>
</tbody>
</table>
### Table A.1 – continued from previous page

<table>
<thead>
<tr>
<th>Application</th>
<th>Use Case Documentation</th>
<th>Service Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Global Blue</td>
<td>[34]</td>
<td>Y</td>
</tr>
<tr>
<td>gumi</td>
<td>[20]</td>
<td>Y</td>
</tr>
<tr>
<td>HashCube</td>
<td>[35]</td>
<td>Y</td>
</tr>
<tr>
<td>Holiday Extras</td>
<td>[21]</td>
<td>Y</td>
</tr>
<tr>
<td>HostedFTP</td>
<td>[36]</td>
<td>Y</td>
</tr>
<tr>
<td>Hotelogix</td>
<td>[37]</td>
<td>A</td>
</tr>
<tr>
<td>ibay365</td>
<td>[38]</td>
<td>Y</td>
</tr>
<tr>
<td>Ice.com</td>
<td>[23]</td>
<td>Y</td>
</tr>
<tr>
<td>Infostrada Sports Group</td>
<td>[39]</td>
<td>Y</td>
</tr>
<tr>
<td>Data Mining</td>
<td>[40]</td>
<td>Y</td>
</tr>
<tr>
<td>Walsh Group</td>
<td>[41]</td>
<td>Y</td>
</tr>
<tr>
<td>Essex County Council</td>
<td>[22]</td>
<td>Y</td>
</tr>
<tr>
<td>Kaggle</td>
<td>[42]</td>
<td>Y</td>
</tr>
</tbody>
</table>

Table A.2: Mapping of Application Migration Use Cases to Migration Types

<table>
<thead>
<tr>
<th>Application</th>
<th>Initial Topology</th>
<th>Migrated Topology</th>
<th>Migration Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Payroll Processing</td>
<td>1 Application &amp; 1 DBMS</td>
<td>1 Application in 4VMs (concurrent); 1 DBMS in 1 VM</td>
<td>III</td>
</tr>
</tbody>
</table>
Table A.2 – continued from previous page

<table>
<thead>
<tr>
<th>Application</th>
<th>Initial Topology</th>
<th>Migrated Topology</th>
<th>Migration Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logistics &amp; Project Management</td>
<td>a combination of Quickbooks and spreadsheets</td>
<td>1 client-side application; data in GAE datastore</td>
<td>I</td>
</tr>
<tr>
<td>Central Government Services</td>
<td>back office systems &amp; front office systems</td>
<td>1 Cloud infrastructure built on a private network</td>
<td>I</td>
</tr>
<tr>
<td>Local Government Services</td>
<td>1800 local governments, each has its own servers and IT staff</td>
<td>internal tasks and some data in the private Cloud; other data locally</td>
<td>I</td>
</tr>
<tr>
<td>Astronomic Data Processing</td>
<td>AGIS software &amp; DBMS</td>
<td>1 DBMS in 1 VM; AGIS in VMs; Storage in 5 Cloud-based storage volumes</td>
<td>III</td>
</tr>
<tr>
<td>Application testing</td>
<td>VM locally</td>
<td>VM in Cloud</td>
<td>I/III</td>
</tr>
<tr>
<td>Application contingency</td>
<td>main site</td>
<td>contingency can be processed by cloud infrastructure</td>
<td>I</td>
</tr>
<tr>
<td>&quot;Big&quot; data processing</td>
<td>massive amounts of data</td>
<td>parallel processing can be done by the Cloud servers/services which based on tools for big data tasks</td>
<td>I/II</td>
</tr>
<tr>
<td>Abaca</td>
<td>servers locally</td>
<td>Amazon EC2 for computing power to solve the requests for highly-efficient spam filter</td>
<td>I</td>
</tr>
<tr>
<td>Banro</td>
<td>ERP &amp; workflow systems &amp; email systems &amp; data warehouse &amp; reporting tools</td>
<td>8 full-time instances ranging from m1.small to m2.2xlarge as an offshore data center solution; Amazon S3 as offsite backups</td>
<td>I</td>
</tr>
<tr>
<td>Global Blue</td>
<td>products system</td>
<td>web site and development environments are hosted on Cloud</td>
<td>I</td>
</tr>
<tr>
<td>Application</td>
<td>Initial Topology</td>
<td>Migrated Topology</td>
<td>Migration Type</td>
</tr>
<tr>
<td>---------------</td>
<td>---------------------------------------------------------------------------------</td>
<td>----------------------------------------------------------------------------------</td>
<td>----------------</td>
</tr>
<tr>
<td>gumi</td>
<td>Web server &amp; database server</td>
<td>all services on AWS; local static images store</td>
<td>I/II</td>
</tr>
<tr>
<td>HashCube</td>
<td>Website’s traffic on Amazon EC2; customer and user information on Amazon S3</td>
<td></td>
<td>I</td>
</tr>
<tr>
<td>Holiday Extras</td>
<td>10 quad core blade web servers &amp; 2 1-TB MySQL databases on the back-end</td>
<td>12 web servers; 1 Amazon RDS; a small number of memcached instances; 10 ELBs</td>
<td>I</td>
</tr>
<tr>
<td>HostedFTP</td>
<td>web-based file sharing system in several co-location facilities</td>
<td>customer files on S3; website and database on EC2; codebase on EBS</td>
<td>IV</td>
</tr>
<tr>
<td>Hotelogix</td>
<td></td>
<td>2 Amazon EC2 instances with Amazon CloudWatch; 1 EBS; local instance store</td>
<td>I/II</td>
</tr>
<tr>
<td>ibay365</td>
<td>database, web services and cache on EC2; users’ products and images on S3</td>
<td></td>
<td>I/II</td>
</tr>
<tr>
<td>Ice.com</td>
<td>1 e-commerce system including 1 new web store application &amp; 1 ERP system &amp; 1 CMS &amp; 1 BI platform</td>
<td>e-commerce site &amp; mobile site &amp; ERP &amp; CMS &amp; BI systems &amp; social loyalty program on EC2, backed up on EBS; static HTML &amp; images &amp; backups on S3; CloudWatch to deliver information on the company’s servers and data</td>
<td>I/II</td>
</tr>
</tbody>
</table>
### Table A.2 – continued from previous page

<table>
<thead>
<tr>
<th>Application</th>
<th>Initial Topology</th>
<th>Migrated Topology</th>
<th>Migration Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infostrada Sports Group</td>
<td>2 EC2 for admin machines &amp; web nodes; EBS for local data storage; S3 for static content; CloudWatch for delivering</td>
<td></td>
<td>I/II</td>
</tr>
<tr>
<td>Data Mining</td>
<td>substantial hardware</td>
<td>VM</td>
<td>III</td>
</tr>
<tr>
<td>Walsh Group</td>
<td>100 physical servers in Chicago data center &amp; 24 in regional offices &amp; 200 at job sites</td>
<td>150 VMs across 10 data center; server count from 324 to 262</td>
<td>III</td>
</tr>
<tr>
<td>Essex County Council</td>
<td>application on Microsoft Excel spreadsheet software</td>
<td>application on SQL Database</td>
<td>I</td>
</tr>
<tr>
<td>Kaggle</td>
<td>application code on cloud services; blob storage</td>
<td></td>
<td>I</td>
</tr>
</tbody>
</table>
Bibliography


82 Decision support for different migration types of applications to the Cloud
A Bibliography


Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

Ort, Datum, Unterschrift