Institute of Architecture of Application Systems
University of Stuttgart
Universittsstrae 38
D–70569 Stuttgart

Diplomarbeit Nr. 3381

A decision support system for application migration to the Cloud

Zhe Song

Course of Study:	Computer Science
Examiner:	Prof. Dr. Frank Leymann
Supervisor:	Dr. Vasilios Andrikopoulos
Commenced:	August.08, 2012
Completed:	February.08, 2013
CR-Classification:	D.2.0, D.2.11, D.2.12

Abstract

Cloud computing enables the computing resources to be available for purchase on demand. The benefits like elasticity, flexibility and expenditure reduction attract many enterprises to consider the migration of their applications to the Cloud. By the rapid expansion of Cloud computing market and the maturation of the offered solutions, the consumers have more and more choices in selecting Cloud offerings. However, when they face a large number of Cloud offerings which have similar features, an appropriate choice will be the key to guarantee a comparatively low operational cost. In this thesis, we propose a decision support system for application migration to the Cloud, by which consumers are able to obtain recommendations on selecting suitable Cloud services. We have evaluated this system with appropriate test cases and compared it with the existing tools and frameworks.

Contents

List of Figures

List of Tables

List of Algorithms

Listings

1 Introduction

This thesis is aimed at introducing a decision support system for application migration to the Cloud. The main contents involve the summary of related works, design and implement of a decision support system, as well as the evaluation of the system. In the first chapter the motivation is described, the scope of problem and the outlining of this thesis are explained.

1.1 Motivation

The development of IT technology helps people get closer to information which they need, whether they are end users or enterprise internal employees that want to obtain and use the required application information or some business information, at anytime from anywhere. How to meet these demands with the least amount of resources is a great challenge which system developers and stakeholders have to face. For the small and medium -sized enterprises, they may have to take the risk of upfront investment for IT infrastructure, and need to spend additional resources to manage a team which has responsibility for maintaining the whole in-house IT system. Besides that, the uncertainty of business growth by startups may lead to over-investment or duplicated construction. Suleiman et al. [1] have in their reserch summarized the contradiction between computing resource capacity and economics: enterprises plan their IT infrastructure based on maximum expected computing resource capacity which involves a large upfront capital investment; this however could reduce enterprises cash flow considerably and increase the period of payback. From all this, if the hardware and software resources can be flexibly and elastically consumed, the operating costs and risk faced by enterprises will be reduced dramatically.

Therefore, because of the advantages of Cloud computing such like scalability, flexibility, and expenditure reduction many enterprises are attracted and begin to consider the migration of applications into Cloud to resolve the contradiction between resource and demand. In other words, Cloud computing realized the delivery of sophisticated virtual services according to Service Level Agreements. Enterprises, as consumers, can obtain resources and services which are offered by Cloud providers through the network [2].

After several years of rapid expansion, the market provides a wide variety of Cloud service offerings. If the stakeholders have decided to migrate the application to Cloud, the choice of a suitable Cloud service plays a central role in reducing enterprise operational costs and improvement of work efficiency. But when stakeholders are faced with a large number of offerings with similar features and characteristics, a correct decision on choosing service is a great challenge, especially, even we consider that even for the same Cloud provider, the price or performance will be changed with different usage amount in various zones.

The comparison of Cloud offerings and a rational recommendation on choosing a suitable service according to consumers requirement is the motivation of this thesis.

1.2 Problem Definition

This work aims to develop a Migration Decision Support System (MDSS) so that it helps consumers to select a suitable service while migrating their application is migrating to the Cloud. We will use a formal definition to describe the problem of selection and decision support.

Table 1.1 presents the concepts required for the formal definition. This definition consists of i providers from p_1 to p_i, j offerings from o_1 to o_j and k configurations from c_1 to c_k. Provider is the vendor who can offer Cloud services, for example, Google and Amazon are the famous providers. The Offerings like VM and relational database service represent the Cloud service offered by Cloud provider, and each of them contains a set of configurations which can be identified with price and characteristic. For example, Google Compute Engine as a offering has 20 configurations which have different CPU cores, sizes of RAM or storage capacities. Besides that, set F contains the features of Cloud service and features are defined with service type like VM service or storage service. M is the set of parameters which are used to identify a configuration c_k in C. CPU cores and location are two of the parameters for identifying the configurations of Google Compute Engine. An offering will be interpreted as function $o_j \leftarrow f(p_i, f_l)$, and a configuration can be identified by $c_k \leftarrow f(\hat{M}, o_j)$ where $\hat{M} \subseteq M$.

Table 1.1: Key Concepts of the Formal Definition

Concept	**Desciption**
$P = \{p_1, ..., p_i\}$	a set of Cloud service Providers
$O = \{o_1, ..., o_j\}$	a set of service offerings
$C = \{c_1, ..., c_k\}$	a set of configurations
$F = \{f_1, ..., f_l\}$	a set of service features
$M = \{m_1, ..., m_m\}$	a set of service parameters
$R = \{r_1, ..., r_r\}$	a set of consumer requirements

Tak [3] and Walker [4] [5] in their research investigated the trade-off during the migration of the entire application or some components from economics and performance aspect respectively. From their works we can see that the enterprises have to analyze the pre-existing application architecture and determine expected computing capability to maximize the benefits of using Cloud services. Therefore, the requirements by service consumer will be taken into account as a prerequisite R which contains a set of requirements from r_1 to r_r. At last we can interpret the problem of decision:

Definition 1 *Decision support for the migration of applications with requirements R to the Cloud requires the identification of a function f such as $\hat{C} \leftarrow f(R)$ where $\hat{C} \subseteq C$*

The proposed system will be developed to help consumers select appropriate Cloud services based on their requirements without considering the combination of different services or components. In other words, the selection focuses on a set of particular services which have similar features rather than a set of deployment patterns, in which all necessary components are taken into account. Therefore, this thesis dose not focus on all the migration types which are defined in [6]. In order to complete MDSS, the first step of this work is to identify the collected data with the concepts which are described in the formal definition. A knowledge base is needed, it should deliver indispensable data during the process of decision support. Then according to the identified data we will design and implement a knowledge base which is the fundamental of MDSS. At last, the related comparison and calculation modules of MDSS will be developed to achieve the process of decision support during the migration of existing applications to the Cloud.

1.3 Outline

In this thesis, we apply the following research approach: survey of the literature to establish the State of the Art in fields decision support and application migration to the Cloud, requirement analysis, system design, prototypical implementation, validation and evaluation.

The rest of this thesis is structured as follows. In Chapter 2 the fundamentals of Cloud computing are introduced and we reflect some related works about decision support for Cloud migration. Some decision support approaches like singular value decomposition is described in this chapter and we discuss several migration frameworks and identify a set of parameters which are used by these frameworks. Chapter 3 presents the specification of the system and the conceptual design. In Chapter 4 the implementation of a knowledge base and the proposed migration decision support system is described. Chapter 5 shows the validation and evaluation of the system with some test cases. Finally we conclude this thesis in Chapter 6.

2 Background

In this chapter the basics concerning Cloud computing are established first, then some related work in the fields of decision support and Cloud migration are summarized and discussed.

2.1 Fundamentals

It is not by coincidence that Cloud computing has developed greatly in recent years. Back in the early 60s, a concept by John McCarthy was proposed that computing can be provided as a utility like water and electricity to the users. However, since the whole IT industry was still in its infancy at that time, many of the necessary technology did not appear, and the implementation of this concept got a number of constraints. The technological breakthrough in traditional fields of computer and network like distributed computing, parallel computing, utility computing, network storage technologies, virtualization and load balance led to the concept of Cloud computing to rise. Furthermore, the emergence of new business models also are key to enabling Cloud computing to become a reality and new application opportunities will promote momentum in the future [7].

A recent publication by NIST described Cloud computing as a model for enabling network access to a shared pool of computing resources that can be rapidly provisioned and released with minimal management effort or service provider interaction [8]. By this publication a Cloud model is composed of three service models.

- SAAS
  Cloud software as a service, compared to traditional software service, the application running on Cloud infrastructure and has been customized to consumer's demand, its install, upgrades and maintenance don't require the end user involvement.

- PAAS
  Cloud platform as a service, it provides basic computing capability which has business development environment supporting deployment of application to the Cloud infrastructure. A set of programming tools like framework, APIs or libraries is usually provided to support consumption of computational, network and storage resources.

- IAAS
  Cloud infrastructure as a service, at this level, a set of fundamental resource like

processing, storage and networks is provided by resource pool. Through virtualization consumers can run their own business systems and access these resources over network instead of construction of in-house infrastructure.

Consumers in principle cannot control and manage networks, servers, operating systems, storage and the other aspects of the underlying Cloud infrastructure on SAAS and PAAS. In IAAS the underlying Cloud infrastructure doesn't offer a manage or control permission for consumers, but they have control over operating systems, storage, deployed application and limited control of some networking components [8].

Table 2.1: Service model mapping to Hierarchical model of Cloud service

Hierarchical Model	**Service Model**
Application service	PAAS
Software service	SAAS
Data Service	
Storage service	
Infrastructure service	IAAS

Besides the service model, Cloud service can be divided into a Hierarchical Model based on its service type provided for clients. This service-oriented architecture has been discussed in work by Wenying et al [9]. Table 2.1 presents the components of the Cloud service hierarchical model. Application service indicates PAAS, a typical example is App Engine by Google that users can upload their application in configured running environment. Software service is mapping to SAAS, for that Web-based email service can be thought of as a classic case. IAAS is expressed as Infrastructure service in hierarchical model. Using each of the three services may generate additional networking traffic, caching and storage costs, these can be respectively considered data service and storage service. In our follow work, these two methods will be adopted for service partition.

From the above scenario it is clear that the IT Architecture at all levels has been subdivided into different components by already existed Cloud service, and there are corresponding solutions for combination and collocation of these components. In a research by Andrikopoulos et al. [6], four migration types have been advanced for distinguishing among the different migration approaches which contains one, several or all components in Cloud service offering:

- Type 1. Replace
  One or more components are replaced with offerings. Typically, this type of migration requires least overhead and takes the smallest risk.

- Type 2. Partially migrate
  Some of the application functionality is migrated to the Cloud. The functionality may need support from a collection of components which belong to different layers. Therefore, these components exist in some sort of interconnected relationships

- Type 3. Migrate the whole software stack
  As the most common type of migration, this type based on virtualization of application and VM environment by Cloud offerings.

- Type 4. Cloudify
  In this case, the application is migrated completely. The functionality of application is executed as a composition of Cloud services [6].

We will use these migration types in the following section to discuss related migration systems.

Table 2.2: Comparison of offerings for different provider

Provider	**Offering**	**CPU Speed**	**CPU Cores**	**Memory**	**SLA**
Google	App Engine	no	no	no	yes
Microsoft	Cloud Service	yes	yes	yes	yes

In recent years many providers have provided a rich variety of products which are aimed at different service models, and more and more products are becoming more sophisticated and reliable. On the one hand an amount of similar types of offering facilitates prosperity of Cloud computing market, simultaneously helps enterprises solve the problem of dynamic demand or contradiction between peek and average, allowing IT departments to focus on the provision of service and business operation. On the other hand once consumer has determined the requirement of offerings, and then a properly effective tradeoff is the first problem to be solved. For example, if a consumer intends to choose a configuration in PAAS service model, Google App Engine and Microsoft Cloud Service each have more than 4 options to meet conditions. Comparison of these options is difficult on account of mismatched parameters and non-functional requirements. In Table 2.2 the offerings with similar feature are presented. By Microsoft the details of instance is available but Google does not provide these information. Hence we have to consider multi-criteria decision for migration system to satisfy the requirements and correctly evaluate the cost.

2.2 Decision Support Systems

When the consumer consider migration of their in-house application into the Cloud, a decision support system is necessary to evaluate the Cloud service, pay close attention to the reduction of risk and trade-off between performance and cost. For a Cloud service there are many different factors that represent the overall performance and the main feature. An example of such factors is provided by the Service Measurement Index (SMI) [10]. The SMI framework summarizes the most importent attribute of QoS in a high level, such like Accountability, Agility, and Assurance of service, Cost, Performance, Security, Privacy and Usability. In order to measure and compare Cloud services, a high-level factor should consist of a collection of parameters which is defined as business-relevant Key Performance Indicators (KPIs) [10]. An example is Assurance containing KPIs like Availability, Service Stability and Service Ability.

Decision process is complicated by a large number of factors and parameters defined in different levels, Zeleny [11] defined this problem as multiple criteria decision making (MCDM). Because of the structured relationship between factor and parameter the Analytic Hierarchy Process (AHP) is a proposed approach for facilitating the problem of MCDM.

Hussain et al. has advanced a multi-criteria decision support method for Cloud service selection in literature [12]. This method uses similarity to make an assessment of Cloud services. By using requirement vectors and service descriptor vectors to construct a decision matrix, we can get the similarity among service provider's parameter and user's requirement. The measurement of difference in performance is enabled by Weighted Difference (WD) and Exponential Weighted Difference (EWD). WD may get a negative value when a Cloud service is below user's requirement. By EWD if a Cloud service value smaller than the requirement value, the final exponential value will be bigger than 1, therefore a small difference means higher similarity. There is another method in literature [13] that consider the similarity and suitability with a singular value decomposition (SVD) approach for ranking Cloud services. This method decomposes a QoS-Provider matrix (Table 2.3) into three matrixes: the QoS factor and provider are represented with left and right singular vectors respectively. The approach works as follows: Choose a few largest singular values from diagonal matrix as the dimension. Then calculate a pseudo-provider which is the centroid of its corresponding QoS attribute points. This centroid represents the average vector of the application required QoS vector. At last calculate the cosine value between pseudo-provider vector and the existing provider vectors, the most cosine value means the most suitalbe provider. Both of these two methods can generate the most suitable result from a candidate list, and especially SVD is very elastic since the precision of matching can be adjusted though modifying the threshold cosine value. However, such a calculation needs numerical value, and how to translate non-numerical value to numerical value is a problem that cannot be ignored.

Table 2.3: A Example for Factor-Provider Matrix

	Provider 1	**Provider 2**	**Provider 3**	**Provider 4**
Security	1	2	5	5
Integrity	2	1	3	4
Availability	3	2	3	3
Mean Time for Repair	3	3	4	5
First Attempt Fix Rate	2	3	3	3
Change Frequency	2	2	1	3
Average Server/VM Ratio	3	3	4	5
Average IO Response	2	3	4	5
Average CPU Utilization	5	3	4	2
Cost	5	3	2	1

2.3 Migration Support Systems

In a common migration process a significant part is mapping requirement from consumer to service factor. The specification of factors involves various fields and requires a uniform definition. The Choosing of factors which contain the main features of services and possible impact is the foundation for comparison and filtration in a decision making process. After that an effective and accurate ranking or selection method recommends best suited candidate. In this section we will study several migration framework in related work and summarize the factors and parameters that were specified by different frameworks.

In literature [14], Menzel et al. has introduced the $(MC^2)^2$ framework that specified requirement scale as binary and ordinal to filter out all infeasible alternatives for getting appropriate migration candidates. This framework defines criteria with questions and corresponding values which could be quantitative and qualitative for ranking the feasible criteria. As a recommendation multi-criteria decision making process the analytic network process (ANP) supports pair-wise comparison of non-measurable qualitative values whose normalization is still generated from subjective rating. Figure 2.1 shows the structure of $(MC^2)^2$ framework. The criteria with questions exams factors like QoS, security, cost etc, and the definition of alternatives, criteria and requirements needs a knowledgebase to deliver necessary internal and external information resources.

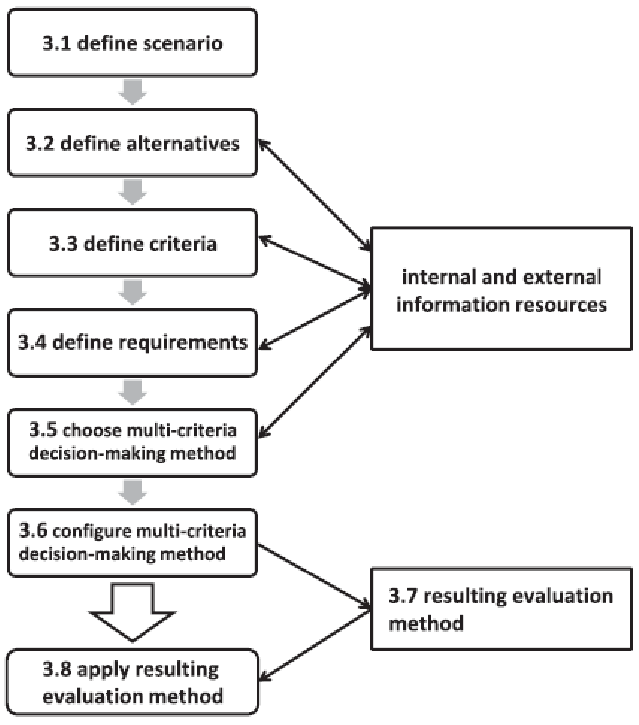

3.1 define scenario
3.2 define alternatives
3.3 define criteria
3.4 define requirements
3.5 choose multi-criteria decision-making method
3.6 configure multi-criteria decision-making method
3.7 resulting evaluation method
3.8 apply resulting evaluation method
internal and external information resources

Figure 2.1: Structure Of $(MC^2)^2$ Framework

The research by Khajeh-Hosseini et al. [15] proposed two decision support tools for Cloud migration within IAAS. The first tool is a spreadsheet that from consumer perspective summarizes the benefit and risk. Table 2.4 shows an example of the factors which have been considered as benefit or risk. Another tool modeling the requirements of application, data and infrastructure for cost estimate. It contains two main parts: a UML deployment diagrams helps modeling the infrastructure of enterprise's IT system and a baseline usage of resource with an elasticity pattern.

Table 2.4: Example of the Benefits and Risk

Benefit	Technical,Financial,Organizational
Risk	Technical,Financial,Organizational,Legal,Security

Three main problems in [16] have been considered by another framework named the Cloud Adoption Toolkit: cost modeling, technology suitability analysis and stakeholder analysis. Suitability analysis contains a simple checklist of questions which is used for evaluation and assessment of the suitability. Under this framework the consumer uses UML tool to describe a deployment diagrams and choose feasible VM, and Cloud storage from a database as candidate. To express the elasticity requirement, a simple notation is preferred to describe temporary and permanent patterns like the modeling tool in [15]. Beside these, the analysis of stakeholder impact refer to Benefit/Risk-list to identify the impact of stakeholder in non-technical area. This framework focuses on analysis of cost in a certain period and non-numerical factors about social and political impact.

CloudGenius [17] based on AHP and applies the $(MC^2)^2$ framework. It considers on VM and Cloud infrastructure service respectively. The VM and service are sorted with values which calculated by weighted parameters. The final ranking is from the feasible combinations of VM and service. For the selection and combination CloudGenius constructs a formal model to describe requirements, non- and numerical attributes.

Table 2.5: Cloud Migration Systems Summary

Cloud migration system	**Migration types**	**Factors taken into account**		**Applied decision support techniques and methods**
CloudGenius	Migrate the whole software stack	Cloud Service	Cost, Performance, provider, characteristic	AHP
		Cloud VM	Cost, Provider	
$(MC^2)^2$	Migrate the whole software stack	Cloud Service	Performance, Provider, Cost, characteristic	ANP
		Cloud VM	characteristic	
Cloud Adoption Toolkit	Migrate the whole software stack	Cost, Characteristic Social factors, Political factors Performance,Practicalities		Technology suitability analysis by checklist, Cost model by UML deployment diagram, Stakeholder impact analysis
SMICloud	Replace Component	Performance, Cost Characteristic		AHP

Another framework based on AHP is SMICloud [10]. AHP is used to compare parameters of different providers with value-based ranking method. The value of the parameter is classified with Boolean, numeric, unordered set and range type. It builds up an N×N Relative Service Ranking Matrix (RSRM) to compare the parameters pair-wise at lowest level , and then compute the Eigen vector as Relative Service Ranking Vector (RSRV) of these matrixes. The result of RSRV is to create a new RSRM for the parameters which is in a higher level and multiply corresponding weights to generate the RSRV which will be used in a higher level. The approach repeats this calculation from low level to high level until reaching the factors, i.e., all the RSRV of all parameters can be aggregated to generate a RSRM for all providers which we have considered. By the multiplication of the weights of factors we can get the final RSRV.

Table 2.5 gives an overview of existing migration system. It summarizes the factors, migration type and applied decision support method.

2.4 Discussion

From the above, it can be observed that an amount of parameters have been considered by these migration systems to describe and evaluate Cloud services. We use a hierarchical taxonomy to classify parameters into four factors, which are performance, cost, non-numerical and numerical characteristics. In Table 2.6 we use a column for each one of these factors. Among similar offerings and configurations offered by Cloud provider, consumer needs to choose a product which is appropriate for their requirement or expectation. For example, a research organization does not need large volume of storage or disk but a high computing capability is necessary. In each migration support system, the capabilities of CPU, RAM and disk are a common set of measurable, numerical parameters. The volume of RAM and disk and CPU speed can be regarded as a measure of the capability of instance. Because most of performance parameters can be expressed as concrete values, performance does not contain non-numerical parameters.

The parameter defined as numerical characteristic is presented in the second column in Table 2.6. Some parameters use unit of time to measure delay in operation, response time or maintain effort. The next column depicts non-numerical characteristics which include some non-functional parameters like location, OS version and QoS. Adoption Toolkit and $(MC^2)^2$ use these non-numerical parameters to ensure that an offering meets consumers' demands and help suitability analysis. Therefore some of them have a Boolean value or are evaluated as different levels and some have particular information to describe specific character such like location or format. The availability, reliability, stability, usability and other QoS quality can be represented with characteristics. A problem is that the official information about them are not offered by the providers. A third party usually provides these data, but the location of data center and size of the sample distribution as well as the statistical period may have an impact on results, so the accuracy and timeliness of the data are not guaranteed. Beside the impact of statistical methods to the result, some potential variables greatly affect QoS quality, for example, different application types running on a same Cloud environment exhibits high performance variability. Iosup et al. discussed performance variability of numerical characteristic in their work [18]. Additionally Li et al. [19] observed variations of instance, storage and network transfer to evaluate and compare the performance with framework Cloudcmp.

Cost as the last factor, normally contains usage duration, volume and read/write operation for storage as well as the size of in/out data. Offerings which have same functionality may have multi-dimension units which is leading to an increase in difficulty of cost comparison. For example, storage cost can be calculated with a unit defined as \$/TB per month or as \$/GB per day, hence a common unit is necessary for cost calculation and comparison. Besides the cost for using service $(MC^2)^2$ considers the additional expenses such as training, support and regulation.

Table 2.6: Parameters defined as Performance in different migration systems

Cloud Migration System	**Performance**		**Characteristic**			**Cost**	
			Numerical		**Non-numerical**		
CloudGenius	CPU	Flops	Max,AVG latency	ms	Location	Instance price	\$/h
	RAM	Flops	Uptime	Percent	OS,Version	License price	\$/h
	DISK	Flops	Popularity	Percent	Format		
Adoption Toolkit	Bandwidth				Elasticity,security	Instance price	\$/h
	CPU				access to hardware	I/O request	Request
	Energy Consumption				confidentiality	In/Out data	\$/GB
$(MC^2)^2$	Data Transfer	MB/S	Delay	ms	Location,Security	Investment, maintenance	
	CPU	Flops	Duration until opration	Hour	Maintenance quality	training,integration	
	RAM	TB	Maintance effort	Hour	Integration effort	regulation,support	
	Storage	TB	Training effort	Hour	Trustworthiness	license/contract	
SIMCloud	CPU		Upload time	Sec	Security	In/Out data	\$/GB
	Memory		Availability	Percent	support service	Storage	\$/GB
	DISK		Stability	Percent		VM	\$/h

Some online available toolkit such like Aotearoa [20] has applied migration support framework as we have discussed in Section 2.3 to achieve a multi-goal Cloud decision-making tool. Users have to define alternatives, goals and criteria themselves, that is to say, it doesn't contain a knowledge base to support user's selection. Furthermore this tool requires key-weight to achieve ranking for alternatives imported by user. The choosing of weight has a great influence on the final result, especially user normally has no experience on setting weights, hence it is difficult to get an accurate output for the initial use.

Cost saving is one of the main purpose that enterprise consider migrating the application to Cloud, for this reason, stakeholder should be able to make accurate estimates by decision support process.

Some Cloud service providers have already offered their own toolkit to support selection of offerings and cost calculation. Microsoft Azure has an online pricing calculator [21] that is able to make accurate estimates for the required services and TCO Comparison Calculator by Amazon [22] provides an estimate for the price difference between Web applications with On-premises and Amazon Web Service. According to variables inputted by consumer this toolkit calculates how many dollars could be saved per year running on Amazon Web Service vs. running a web application on the in-house infrastructure defined by consumer itself. It is an obvious limitation that through both toolkits they cannot compare the performance or price against different providers.

Therefore there exists a clear demand for a system that is able to do not only cost calculations but also considers the multi-criteria requirements supported by the various migration support systems discussed in the previous.

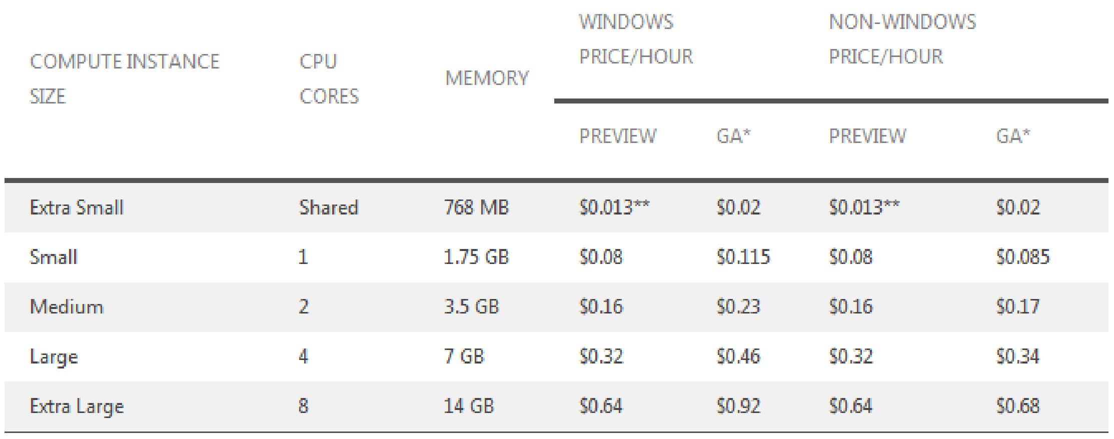
3 Specification & Design

In this chapter the specification of system is described and the concept of system design is explained.

3.1 Requirements

As we have identified the necessary parameters which should be considered while an application is migrating in the Cloud, a provider knowledge base is able to be designed to realize decision support process. And according to the formal definition in Section 1.1, we propose a migration decision support system, by which consumers can obtain the recommendations on choosing appropriate Cloud service offerings after they have explicitly stated their requirements during the migration of existing applications to the Cloud.

After the consumer determines a deployment scenario, a lot of offerings with similar service configurations may be available with diversity in factors. These differences in performance, characteristic and usage amount induce large price swings. For example, the lowest performance VM instance (Extra Small) by Windows Azure has a price at 0.02$ per hour with general availability state, but the price of highest performance instance (Extra Large) is 0.92$ per hour, the price differential is up to 46 times [23]. Figure 3.1 shows the price detail of Azure VM service. Therefore consumer should choose advisable value for factor variables to decrease expense.



COMPUTE INSTANCE SIZE	CPU CORES	MEMORY	WINDOWS PRICE/HOUR		NON-WINDOWS PRICE/HOUR	
			PREVIEW	GA*	PREVIEW	GA*
Extra Small	Shared	768 MB	$0.013**	$0.02	$0.013**	$0.02
Small	1	1.75 GB	$0.08	$0.115	$0.08	$0.085
Medium	2	3.5 GB	$0.16	$0.23	$0.16	$0.17
Large	4	7 GB	$0.32	$0.46	$0.32	$0.34
Extra Large	8	14 GB	$0.64	$0.92	$0.64	$0.68

Figure 3.1: Window Azure Pricing Details

Based on the major feature and limitations of existing tools that we discussed in Section 2.4, we identify the following requirements for a Cloud migration decision support system:

- Function
  The main function of our proposed system is to perform a comparison of Cloud services from different providers. The parameters of comparison focus on performance, characteristic and cost. This system enables offering matching and cost forecast with users requirement based on multi-criteria analysis. Cost calculation supports not only the price per unit time but also the details of usage pattern.

- Data
  A knowledge base should be created, it contains information about Cloud service offerings as well as their identified parameters. Given the requirement to be able to compare service among different providers, the knowledge base should consider the information from more than one Cloud service provider.

- Interaction
  The system provides a pleasant, intuitive, user-friendly interface. The user interface acts as the Frontend of the system where interaction takes place between user and system. Entering users requirements in system and displaying feedback results are designed to be a way to realize interaction.

- Others
  User can use this system with platform independence and it has the necessity of extensibility and scalability for the future development.

Table 3.1 shows an overview of the system requirements.

Table 3.1: Overview Of System Requirement

Requirement Type	**Requirements**
Function	Offering Matching, Cost Calculation, Usage Pattern Design, Service Comparison
Data	Knowledge Base, Multi-Provider Support
Interaction	User-friendly Interface
Others	Platform Independence, Extensibility

3.2 System Specification

Figure 3.2 gives an overview of the use cases extracted from the requirements discussed above. According to the requirements of the user the knowledge base can provide targeted options for parameters. User chooses options and provides numerical and non-numerical parameters based on requirement. Beside the basic usage user can also define usage pattern with discretional period, trend, and rate. After the system has collected the necessary data, the knowledge base should filter out infeasible configurations and generate estimated cost. Then the user can check the results which are generated from the Backend and should be displayed on the user interface. By these results the user selects preferred configuration and chooses one parameter as key-variable for ranking. Finally, the system outputs the final result s ranked by the chosen parameter.

According to the requirement of platform independence we propose a system running on Web environment. It consists of three main parts: user interface, data handler procedure and knowledge base.

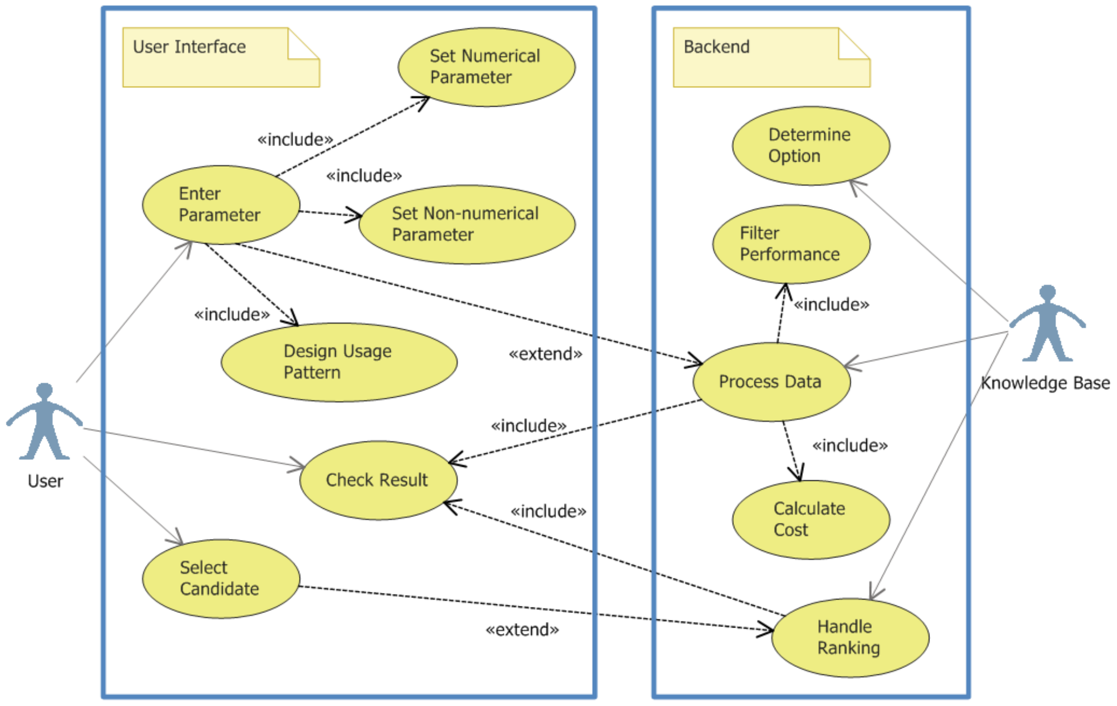


Figure 3.2: Use Case Diagram

Knowledge base contains a data model using relation database. Development of knowledge base should be aimed at Cloud service offerings which are market-oriented products. The Offerings from different providers and their parameters are identified, additionally finding out relationship among them to determine the classification for data modeling. An important requirement of this system is extensibility, when providers make changes to their services, the system should update the corresponding data in the knowledge base without having to make changes to the basic code, therefore, some parameters should be taken into account by knowledge base instead of realized by other modules.

Construction of knowledge base is groundwork for helping database define entities and relationship, and for supporting matching as well as cost calculation.

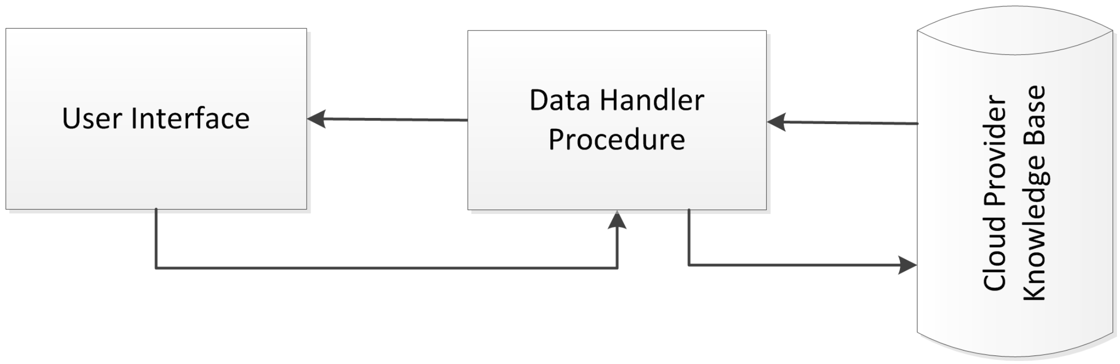


Figure 3.3: Modules of Decision Support System

Figure 3.3 shows the modules of decision support system. Data handler procedure is used to manipulate matching and calculation. This process is also the key to passing data between the modules whose implementation may depend on interactions with user interface and knowledge base. It enables to dynamically render options to user interface according to conditions defined by users and obtain necessary parameters retrieved from user interface, in certain cases, the lack of prior knowledge cause that user cannot determine a precise parameter value, system allows entering a value of NULL. Then the communication with database helps it query or retrieve the data from multiple tables based on various criteria. Typically, in order to complete the calculations that the query with different parameters will be executed more than once. For manipulating the matching and cost calculation, it is able to classify the final data or intermediate data and manage them in an ordered set, according to condition defined by user. Finally data handler should send results back to user interface.

3.3 System Design

Figure 3.4 gives an overview of the conceptual model of the proposed migration decision support system. The attributes of configuration are classified as hierarchical structure. Provider offerings can be identified with service level and service type as we have discussed in Section 2.1. One offering may contain more than one configuration, though they are used to achieve similar functionality, but performance and price may be very different. Hence the system must be able to distinguish between them. This model considers the principal parameters in performance, characteristics and cost. Currently it is aimed at the parameters which were official data released by service provider. However, it should be prepared to use a wider range of parameters in the future.

We found in our investigation that parameters affecting cost of configurations with same performance and characteristics are location of data-center and usage amount. Therefore they are the important parameter used for definition of requirement. In order to calculate cost this model applies formulas with multiple parameters to express basic cost. These

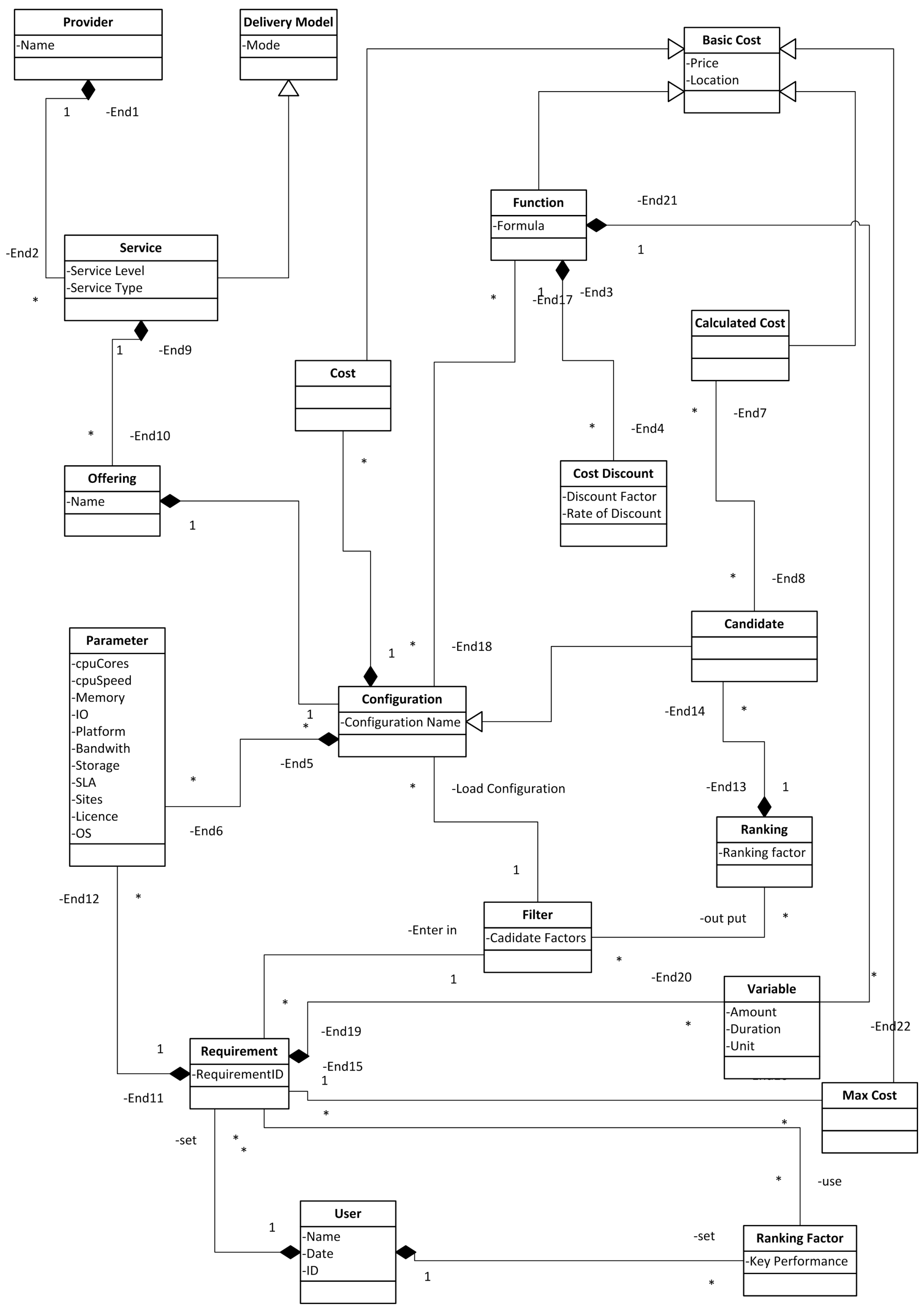


Figure 3.4: Conceptual Model of Decision Support System

parameters in the formula are numerical and obtained from users requirement. As a record that is stored in database, a remarkable point of this is that it can increase scalability and extensibility of the system. Each time a provider makes changes to pricing pattern, the system will only need to update the related entities in data model.

Another parameter affecting price of configuration is the location which is non-numerical, and it will be defined as an attribute in the basic cost entity.

In Figure 3.4 a requirement contains parameters and variables, the values of which will be used by filter and other components in the data handler to compare with the data stored in database and to calculate cost. According to the provided requirements, the filter will check out the data entity and choose the configuration that meets the multiple criteria. The selected configuration will be called as candidate. In order to be able to rank candidates, the system needs to calculate the cost of each candidate.

In this conceptual model, entities used to describe attributes of configuration will be presented as different tables in a relational database, and the components for parameter comparison, cost calculation and ranking will be implemented in data handler with backend code.

3.3.1 Cloud Provider Knowledge Base

A knowledge base is the fundamental part of implementing data handler procedure and it provides information resource for every step in decision support process. Through requirement analysis an Entity-Relation diagram is designed to support the implementation of the system's knowledge base. This ER-model contains 6 main entities and the relationships among them. Each principal property of a Cloud service will be abstracted as an entity which can be used to classify offerings and their configuration. Diagram 3.5 shows the Entity-Relation diagram.

Entity Provider contains the basic information of a Cloud service provider, it has a one-to-more relationship with Offering entity which can also be specified by entity Service type. One offering may have more configurations which are divided into groups with different performance or usage amount, hence each configuration has a Performance entity of its own which includes attributes about service capability or characteristic; this mapping can be represented by a one-to-one relationship. Additionally, cost is specified as another important factor in decision process. For one configuration, its cost is dynamic and depends on Location entity as well as Usage entity. These two parameters have a one-to-more relation with Cost entity. To make a accurate cost estimate, Variable entity will identify the necessary information about multiple parameters in each calculation.

3.3.2 Offerings Matcher

The role of offering matcher is as a component in data handler to recommend appropriate candidates who meet all of users multiple criteria. Most parameters of major factors are measurable, numerical like CPU, RAM, but there are some critical parameters like license, operation system expressed as non-numerical format. To compare configurations we need to distinguish between non-numerical and numerical parameters.

For numerical parameters, it is possible to analyze particular performance or characteristic with measurement of integer or floating-point types. For example, higher frequency of CPU or larger size of memory may signify better performance. Therefore we can simply

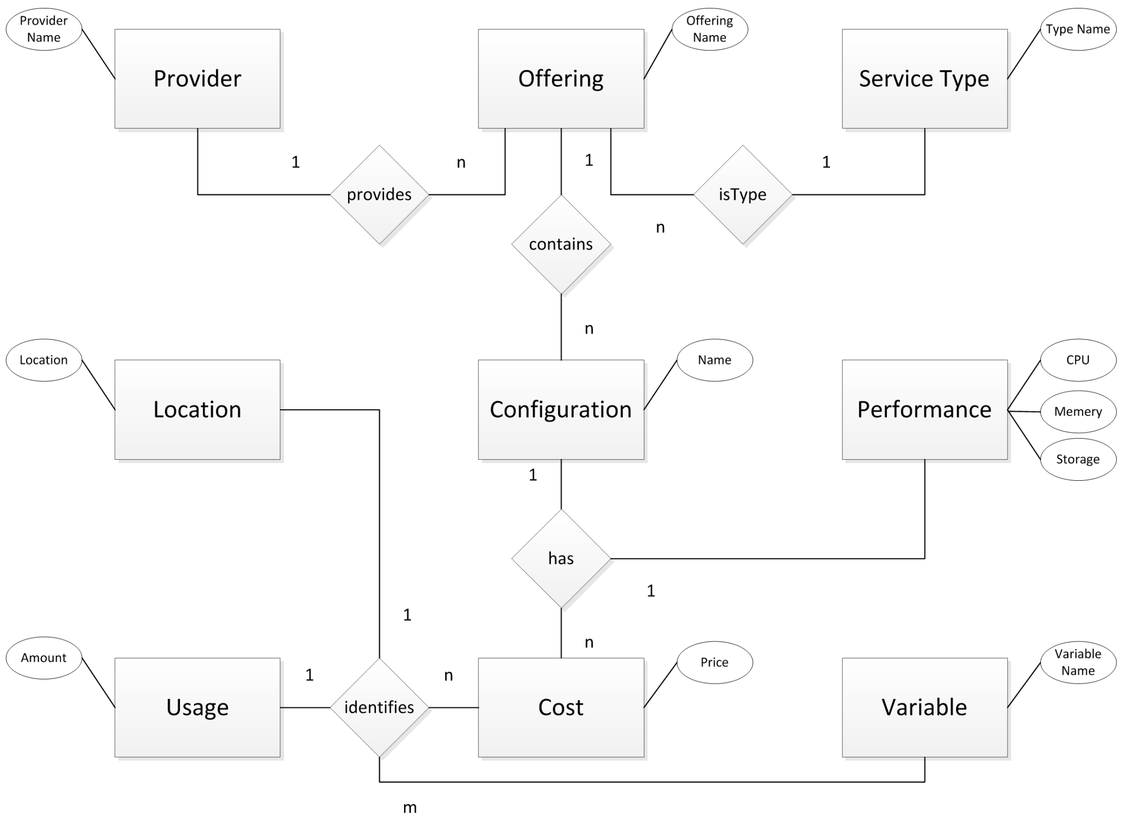


Figure 3.5: ER-Diagram of Cloud Knowledge Base

compare numerical values to determine the candidates, and the role of value inputted by user works as a threshold. A configuration will be filtered out if one of its parameter is less than the value given by user.

For non-numerical parameters, the format of expression of performance or characteristic is more diverse, and it is difficult to compare with a unified approach. Some values are discrete variable, and are incomparable like the type of database management system. In this case, the relationship between value defined by user and particular characteristic parameter is one to one, this means, it will be filtered out if the discrete parameter failed to exactly match the corresponding input value by user. Another possible format is sorted with hierarchical levels, for example, I/O performance is described by Microsoft Azure with low, moderate and high. Although these values are non-numerical, but based on their hierarchical level they can be converted to numerical value, in this case, the values of parameter on each level are able to be mapped to a constant depending on the grade. After converting they will be comparable and the relationship between user input and parameter value of offering is one-to-many. There could be more than one value satisfying users need. Additionally, Boolean is also a non-numerical format used to express some characteristic of offerings. This type is easier to check if the requested values are available.

We propose a method whose argument value is a kind of non-numerical parameter, although the passing value is varied, but return value should be expressed with unified

Algorithm 1 Offering Matcher

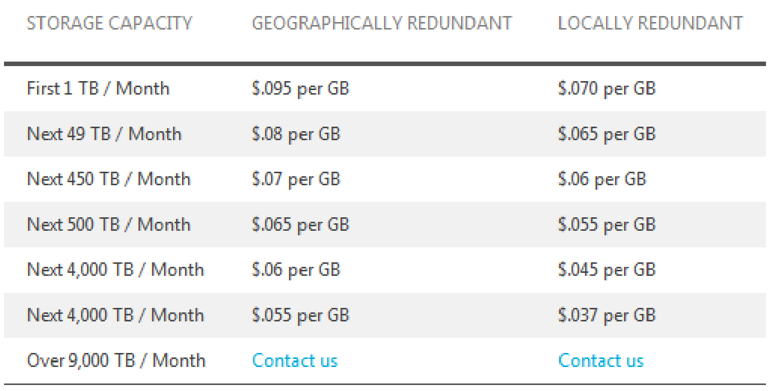
```
INPUT:
    $List : L_c,\ L_p;\ Var : baseValue, inputValue;$
    $(*\ L_c =$ configurations that belongs to selected offerings $*)$
    $(*\ L_p =$ required parameters $*)$
Var $condition$: Boolean
procedure $OfferingMatcher(L_c, L_p, baseValue, inputValue)$
Begin:
    for each $c \in L_c$ do
      for each $p \in L_p$ do
        if $p\ is\ numericalValue$ then
          if $baseValue < inputValue$ then
            remove $c$ from $L_c$;
            break;
          end if
        else
          $condition := matching(inputValue, baseValue)$;
          /*matching() implementation depends on
          /the type of inputValue and base Value*/
          if $\neg condition$ then
            remove $c$ from $L_c$;
            break;
          end if
        end if
      end for
    end for
endprocedure
```

format. We set a return value as Boolean type to determine if a service meets the requirement of user. Any parameter of a configuration returns false, then it will be filtered out.

Algorithm 1 shows a procedure used to match offerings. The initialization of the input data includes four variables. List L_c contains the configuration of user preferred offerings and the elements in $List_p$ are the types of necessary parameters. Variable $inputValue$ is assigned by user on user interface and every parameters value is represented with $baseValue$ which can be obtained from knowledge base. If any numerical value less than user input or any non-numerical value is failed to meet the condition, this configuration will be filtered out from L_c. At last, L_c contains all configurations which satisfy every users requirement and are ordered as candidates for the next step of decision support.

3.3.3 Costs Calculator

Cost calculator is used to estimate pricing with multiple parameters whose values are assigned by user. Almost every offering has its own pricing, and for most services, the relationship between unit-price and consumption is fixed, this means, the coefficient of calculation is a constant. But for some services, the price will have changed with the increase of consumption, the change of coefficient leads to a dynamic function. Figure 3.6 shows the pricing details of Azure Storage Service [23]. We can see that, the cost of this offering will decrease with increase of amount, but for every grades of storage capacity the unit-price is a constant. Therefore it is necessary to set a calculation respectively for each amount range.



STORAGE CAPACITY	GEOGRAPHICALLY REDUNDANT	LOCALLY REDUNDANT
First 1 TB / Month	$.095 per GB	$.070 per GB
Next 49 TB / Month	$.08 per GB	$.065 per GB
Next 450 TB / Month	$.07 per GB	$.06 per GB
Next 500 TB / Month	$.065 per GB	$.055 per GB
Next 4,000 TB / Month	$.06 per GB	$.045 per GB
Next 4,000 TB / Month	$.055 per GB	$.037 per GB
Over 9,000 TB / Month	Contact us	Contact us

Figure 3.6: Pricing Details of Microsoft Azure Storage Service

In order to effectively calculate cost for the offerings from different providers, each offerings price will be determined by a formula that contains the necessary parameters. It would have the major advantage that it would not require any changes to the backend code while provide have changed the pricing details and it is able to facilitate system extensibility and flexibility. The following formula is an example for calculating the cost of Azure Storage in different ranges which we have showed in Figure 3.6. A formula may have more than one parameters, in this case, a value given by user should correspond to a designated parameter in formula. The relationship between input value and the position of parameter in formula is pair-wise.

$$cost = \begin{cases} (0.125 \times para1) + para2 \times (0.01 \div 100000) & First1TB/Month \\ (125 + 0.11 \times (para1 - 1000)) + para2 \times (0.01 \div 100000) & Next49TB/Month \\ (5514 + 0.095 \times (para1 - 50000)) + para2 \times (0.01 \div 100000) & Next450TB/Month \end{cases}$$

In some cases, the user is unable to estimate the consumption for a particular offering, for that input may be null value. When this happens, for fixed unit-price we can set a initial value with smallest amount of usage to calculate cost but for dynamic unit-price user may want to know the cost in every amount range, hence each configuration will have a list expressing the cost with default maximum and minimum consumption in each amount range.

Algorithm 2 shows a procedure which is used to calculate cost. L_c contains a list of configuration whose cost should be calculated, and L_s contains the corresponding cost

Algorithm 2 Cost Calculation

```
INPUT:
    $List : L_c, L_s, L_v;$
    $(* L_c =$congfiguration$*)$
    $(* L_s =$cost string$*)$
    $(* L_v =$user input value$*)$
Var $variableList(value, position)$:ArrayList; $defaultMin, defaultMax$:Double
procedure $CostCalculation(L_c, L_s, L_v)$
Begin:
    for each $c \in L_c$ do
        $variableList \leftarrow \begin{cases} value := v \in L_v & \forall v \in L_v \neq NULL \\ value := v \in (initialValue \vee L_v) & \exists v \in L_s = NULL \\ & \wedge v \in fixedunit-price \\ value := NULL & \exists v \in L_s = NULL \\ & \wedge v \in dynamicunitprice \end{cases}$
        for each $s \in L_s$ do
            if $\neg\exists value = NULL \in variableList$ then
                $calculate(s, variableList);$
            else
                for each $value = NULL \in variableList$ do
                    $value := defaultMin;$
                end for
                $calculate(s, variableList);$
                for each $value = NULL \in variableList$ do
                    $value := defaultMax;$
                end for
                $calculate(s, variableList);$
            end if
        end for
    end for
endprocedure
```

formula expressed as string. The element in L_v is input value assigned by user. Additionally, ArrayList *variableList* consists of pair-wise element expressing the value of parameter and its position in formula. After procedure determined the input value, *variableList* will be assigned with value, and then cost can be calculated with user required or default value.

Based on their requirement analysis users should be able to design a pattern for the demand of elastic usage amount. A usage pattern covers four variants: type of parameter, type of change trend, period and change rate. Some offerings have multiple parameters for pricing. For example, in storage service whose parameter include the capacity of storage and number of transactions, it may happen that capacity increases but transaction declines at the same time or both of them have same change trend but rate is different. Therefore the elastic change is independent for each parameter, and the usage in months for every parameter needs to be taken into account respectively. Furthermore the change trend has three options: increase, decrease and invariant. By each of the trend option user can also define a period in months and rate in percentage. Table 3.2 gives an overview of the variants.

Table 3.2: Variants Of Usage Pattern

Variant	**Value**
Type of Parameter	GB, Hour ,Transaction...
Type of Trend	Increase, Decrease, Invariant
Period	Month
Rate	Percentage

Additionally, because fixed unit-price has a constant coefficient between usage amount and cost, we can directly calculate its cost by the change rate of usage. But for dynamic unit-price every change may lead to an overranging amount and the former price is no longer applicable, hence we need calculate the usage amount for every month in a period.

Algorithm 3 Cost Calculation for Usage Pattern

```
INPUT:
    $List$ : $L_m$,$L_u$,$usage$,$rate$;
    (* $L_m$ =pattern period*)
    (* $L_u$ =unit*)
    (* $usage$ =usage amount for last month*)
    (* $rate$ =change rate*)
Var $variableList(value, position)$:ArrayList;
procedure $PatternCalculation(L_m, L_u, usage, rate)$
Begin:
    for each $month \in L_m$ do
        for each $unit \in L_u$ do
            $usage := usage * rate$;
            $variableList \leftarrow usage$;
        end for
        $calculate(costString, variableList)$;
    end for
endprocedure
```

Algorithm 3 shows a procedure which is used for pattern calculation. L_m represents the length of period and L_u contains parameters which should be considered in calculation. For the difference of change trend by multiple parameters in one period the usage amount needs to be calculated on the monthly basis for every parameter respectively, and the result will be stored in a list for obtaining the final cost.

3.3.4 User Interface

By the use case described in Section 3.2, there are three steps for the interaction of decision support process. Because of the platform-neutrality requirement, Web Forms are applied as user interface. Controls are arranged in an intuitive manner on forms. For each step there will be a web page expressing particular options or calculated results.

- First step is collecting and processing users requirement. User can choose a service type to obtain a list of offerings which are available for required service type. Then user selects one or more offerings from this list for comparison. And user can enter the requirements which are able to be mapped to the necessary parameters like location, performance and usage amount. Especially, the options for requirement must be dynamically generated, since different types of offering do not have the same factors. Additionally, user is able to design an usage pattern for calculating the total cost.

- Second step is displaying results and setting key performance for candidates to rank. System shows the results from last step on user interface. Then user is able to choose some or all candidates for ranking based on a preferred key performance.

- Last step, the final results will be expressed as a table to display configurations recommended by the decision support process.

3.4 Summary

In this chapter the requirements of the proposed system is identified from several aspects. And this system is specified as three modules: user interface, data handler and Cloud provider knowledge base. In order to realize the knowledge base, we complete an ER-diagram to describe the relation between identified entities. Data handler involves two important procedures: offering matching and cost calculation. The algorithms are developed to achieve the goals of recommendation on selecting appropriate offerings and calculation corresponding cost. Finally, we give a specification for the design of user interface.

4 Implementation

This chapter describes the realization of a Cloud provider knowledge base and the implementation of the proposed migration decision support system.

4.1 Cloud Provider Knowledge Base

A relational database is developed for realizing Cloud provider knowledge base with the ER-model presented in Section 3.3.1. Through queries to the database the system can obtain necessary resources which are used to match offerings and calculate cost. The relational database is based on Microsoft SQL Server 2008 Express and contains 14 entities (Figure 4.1).

Attributes in entity Offering includes information about provider, service type and belongs to which level in service model. Configuration entity will inherit these attributes from Offering. Additionally there is a one-to-one relationship between Performance and Configuration, every configuration has different attributes in performance respectively.

For matching offerings, the system should first generate performance options and display them on Web forms. Different services have different performance characteristics; as a user has determined the service type he needs, the system should also be able to create corresponding performance options for this service. For all Cloud service providers, segmentation of the market allows the vendor to tailor its scenario to the consumers requirement, this means, every offering will have more than one configurations. As a precondition for comparing and ranking configurations, user need to submit their requirements to the system, therefore the generation of parameter options is dynamic and it depends on the service type selected by user. For this reason, we need an entity to express which performance parameters are contained in a particular service type. There is a mapping between performance and service type in knowledge base. This ensures that system can give the appropriate options depending on users choice.

The relationship between configuration and cost will be one-to-many. Parameters like usage amount or location can lead to generate a number of different costs for the same configuration. For some services like data transfer or storage, price has a direct relationship with amount of usage. The attribute costString in cost express formula with numerical parameter. In some cases, there are more than one parameter in one formula and every parameter has different units. For example, the parameters of Google Cloud SQL for cost calculation includes instance type, size of storage and usage period, a total of three parameters [24]. We need a mapping between costString and variables, in order to dynamically generate options for amount of usage and to let the system know how

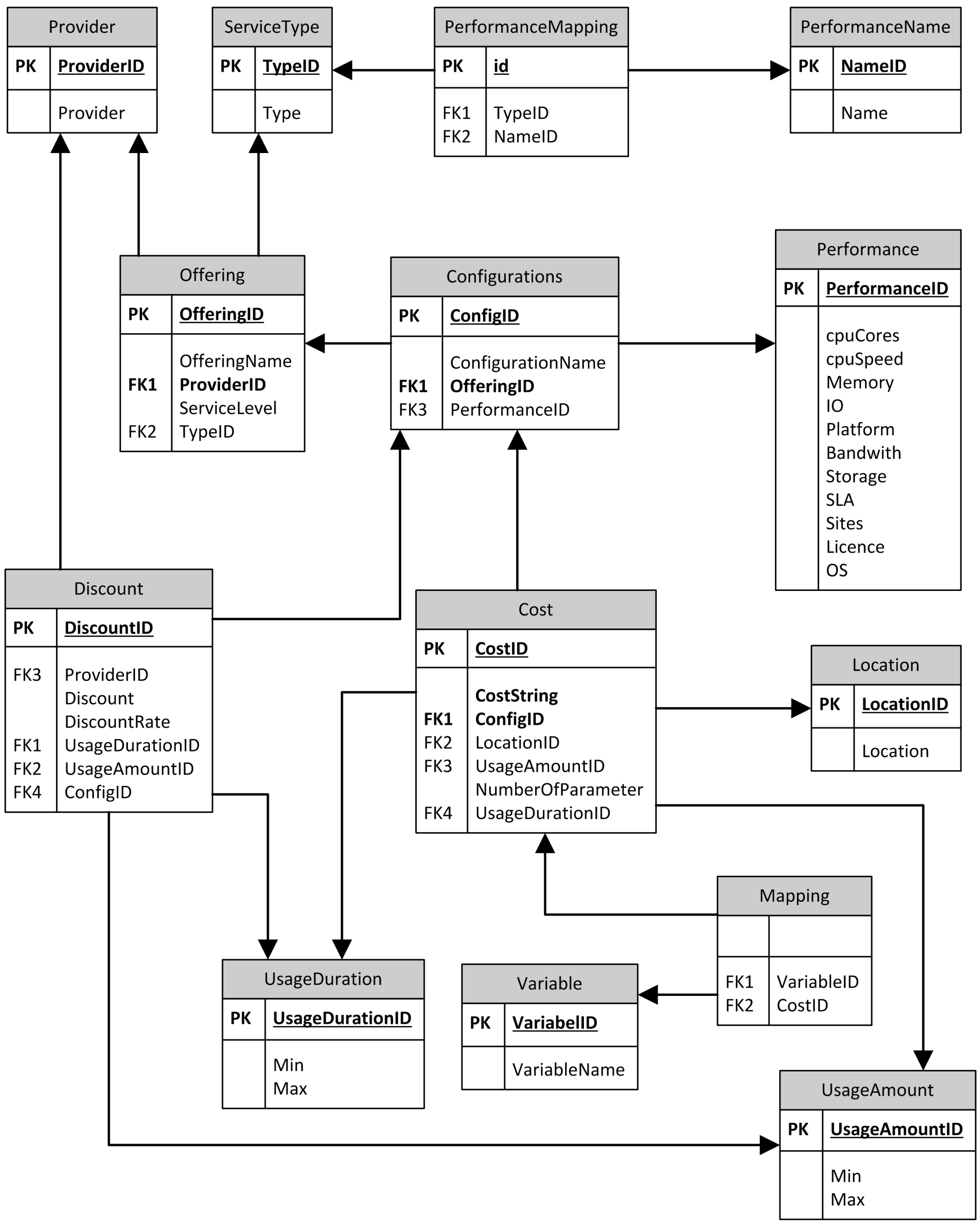


Figure 4.1: Data Model of Decision Support System

to set parameters in formula with corresponding variables. Location of datacenter and usage amount must be able to have respective entity and have a relationship with cost.

4.2 System Outline

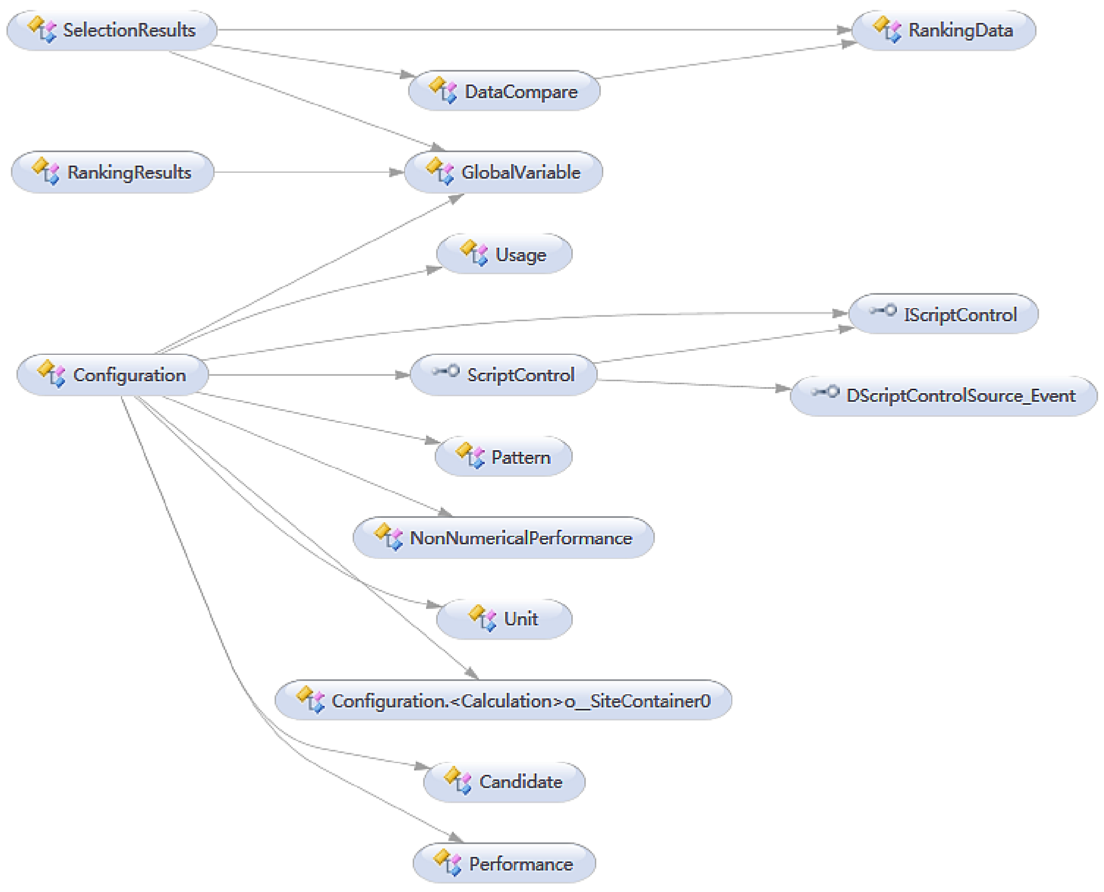


Figure 4.2: Overview Of Class Dependencies

The development tool is Microsoft Visual Studio 2010 based on Microsoft .NET Framework 4.0, development language is Microsoft Visual C# and ASP.NET.

In ASP.NET Web pages, user interface consists of two parts, the visual component and the logic. The variables and methods of the Web pages are defined in class *Configuration*, *SelectionResults* and *RankingResults* as logic part respectively. In these classes some methods are used to control the life cycle of Web page and some methods are used to handle the events from Controls on form like *Button*, *DropDownList*. When an event occurs on client, it will be automatically captured and trigger the corresponding method. The invoked method will be processed by handling the event with server-side code instead of client script.

To filter out infeasible configuration the corresponding values must be able to be compared and the cost calculation involves multiple variables. In other words, multiple type variables or data will be used in different procedures over and over. Hence a type-safe encapsulating method without committing to actual data types is required for using data which have the same properties. Generic is applied to construct several classes to enhance the reusability of code. Class *Performance* and *Unit* encapsulate the input data for offering matcher and cost calculator respectively. Class $NonNumericalPerformance$ is used to compare the non-functional parameters and Class *RankingData* includes the necessary properties to sort the candidates which are defined in class *Candidate*. For handling the usage pattern class *Pattern* and $Usage$ are created to record pattern defined by user and store the temporary intermediate variables.

Figure 4.2 shows the dependencies between classes. Page Configuration is responsible for matching and calculation, it has independencies with class *Unit*, $Performance$ and $NonNumericalPerformance$, furthermore it needs an instance of class *ScriptControl* to enable calculation with *CostString*. In page *SelectionResults* ranking is realized by reference to class *DataCompare* and *RankingData*.

4.3 Offerings Matcher

As an important component in data handler procedure, offering matcher plays a central role in decision support process. The main task of offering matcher is to obtain a collection of configurations whose performance and characteristic are enough to satisfy users requirement.

First of all, we have an initial collection including offerings selected by user, and each offering contains more than one configurations. In offering matcher each element in this collection must be able to have comparison process to filter out the infeasible configuration by iterating through them. On the beginning a query is created that joins the $Performance$ table and the $Configuration$ table on the $PerformanceID$ fields. The result of this query consists of the performance values which belong to the configuration in the collection at the current position of the loop. Typically, through the above query every configuration has a set of performance values. Hence by iterating query record the comparison will focus on each of the performance values for offering matching. Additionally, for this comparison we need another loop for iterating all values assigned by user. From all this, offering matcher uses a nested loop to generate a List which contains the identifiers of suitable configurations.

Table 4.1: Classification Of Values

	Numerical	**Non-Numerical**
Requirement	value1	value2
Knowledge Base	value3	value4

Table 4.1 shows that the performance value can be classified with four types. From the aspect of parameter type performance value can be divided into numerical value and non-numerical value, furthermore these values can be identified by the source of value, this means, a performance value may be assigned with users requirement($value\_require$) or be the property of a configuration stored in knowledge base($value\_provide$). There is fundamental work to be done that can compare the values between users requirement and knowledge base with two different methods. First the parameter type of a performance

value should be identified. There is a List *nonNumericalPerformanceName* that consists of identifier of all non-numerical performances. For one value at the current position of assigned value loop its parameter type will be checked. As a Boolean type, the result of this check is used to control the comparison method.

Listing 4.1: Offering Matcher

```

for(int i = 0; i < conigIDWithPerformanceList.Count; i++){
 String sql_performance_query = "select " + para_query +
           " from Performance join Configuration on " +
           "Configuration.PerformanceID=Performance.PerformanceID where" +
           "ConfigID="+conigIDWithPerformanceList[i].ToString();
 SqlCommand comm = new SqlCommand(sql_performance_query, conn);
 SqlDataReader reader = comm.ExecuteReader();
 while (reader.Read()){
  foreach (Performance p in inputPerformanceList){
   double value_require, value_provide;
   Boolean Condition;
   foreach (String s in nonNumericalPerformanceName){
    if(s.Equals(p.RequiredPerformance)){
        isNumerical = false;}
   }
   if (isNumerical){
    if (!(p.Input.Equals("") || p.Input.Equals(p.RequiredPerformance))){
     value_require = double.Parse(p.Input);
        }else{
         value_require = 0;
    }
    if (reader[p.RequiredPerformance].ToString().Equals("")){
     value_provide = value_require;
    }else{
     value_provide = double.Parse(reader[p.RequiredPerformance].ToString());
    }
    if (value_provide < value_require){
     conigIDWithPerformanceList.RemoveAt(i);
     i--;
     break;
    }
   }else{
    if (!(p.Input.Equals("") || p.Input.Equals(p.RequiredPerformance))){
     NonNumericalPerformance nonNumerical = new NonNumericalPerformance(
        p.Input,p.RequiredPerformance,reader[p.RequiredPerformance].ToString());
     nonNumerical.Comparsion();
     Condition = nonNumerical.Condition;
    }else{
     Condition = true;
    }
    if (!Condition){
     conigIDWithPerformanceList.RemoveAt(i);
     i--;
     isNumerical = true;
     break;
    }
   }
   isNumerical = true;
  }
 }
 reader.Close();
}
```

If the parameter type is numerical, the assigned value will be directly compared with the value obtained from knowledge base. If the assigned value is bigger, this configuration must be removed from the configuration list. Note that if the users requirement is default or corresponding value in data base is null, the performance will be considered as matched. This means that the system is optimistic with respect to fulfilling users' requirements in the absence of additional information.

If the parameter type is non-numerical, the two values with different sources will be the arguments to generate an instance of class *NonNumericalPerformance*; this instance is able to use a method to compare the non-numerical value.

Listing 4.1 shows the implementation details of offering matcher procedure.

4.4 Costs Calculator

After the candidates have been generated by offering matcher, as an indispensable step the corresponding pricing must be calculated. The cost calculator is responsible for providing an estimate cost by users requirement. Listing 4.2 gives the related code for implementing calculation.

Listing 4.2: Cost Calculator

```

foreach (Object o in configList){
 foreach (String[] s in variableList){
  if (s[0].ToString().Equals(o.ToString())){
   foreach (Unit inputValue in inputValueList){
    if (inputValue.UnitValue.Equals(s[2].ToString())){
         int i = int.Parse(s[3].ToString());
         if (!(inputValue.Input.Equals("") || inputValue.Input.Equals(
                inputValue.UnitValue)){position[i - 1] = inputValue.Input;
          if (inputValue.UnitValue.Equals("GB")||inputValue.UnitValue.Equals(
                "GB for Network")){input1 = inputValue.Input.ToString();
           para1 = "9";
      }
     }else{
      position[i - 1] = "1";
      if (inputValue.UnitValue.Equals("GB") || inputValue.UnitValue.Equals(
        "GB for Network")){
       locator = i - 1;
       para1 = "1";
       input1 = "0";
      }
  ... SQL Query
 while (reader.Read()){
  function = reader["CostString"].ToString();
  if (reader["UsageAmountID"].ToString().Equals("") || para1.Equals("9")){
   double cost = Calculation(function, position[0], position[1], position[2],
       position[3]);
  }else{
   position[locator] = reader["Min"].ToString();
   double min = Calculation(function, position[0], position[1], position[2],
       position[3]);
   position[locator] = reader["Max"].ToString();
   double max = Calculation(function, position[0], position[1], position[2],
       position[3]);
  }
 }
 reader.Close();
}
```

For every configuration stored in a list, at the first step the values entered by user and their corresponding positions have to be determined. For one configuration each parameter for cost calculation will have a unique identifier to associate the value assigned by user with its position in cost formula. The variable *variableList* contains a set of String array which consists of the data about unique identifier and its position. The elements in another variable *inputValueList* are instance of class *Unit* which encapsulates the information of input value and identifier. If a unique identifier in these two lists can be matched, then the input value will be associated with a position.

Another necessary part in this step is to check if the input value is null. We have to give every null input an initial value for the later calculation. As we have discussed in Section 3.3.3 the dynamic unit-price like GB has more than one usage range and for everyone in them there will be a different cost formula needs the initial value. Hence this kind of unit-price must be able to be handled separately. If the input for dynamic unit-price is null, a marker will be updated for expressing the state that will be used in a query for cost formula.

Listing 4.3 shows a SQL query for the next step. It compares fields by using one field as a criterion for the other. Through determining the state of marker the query can be aimed at achieving initial value for null input of dynamic unit. In this case, the query result may have a set of formulas. Additionally, the initial value for every usage range will also be included in the record set, if the marker has been updated at the previous step. We will use these formulas and the value assigned by user or by initialization to calculate cost.

Listing 4.3: SQL Query For Cost

```sql

Select * from Cost join Configuration on Configuration.ConfigID= Cost.ConfigID
left join UsageAmount on UsageAmount.UsageAmountID = Cost.UsageAmountID
join Location on Location.LocationID = Cost.LocationID
join Offering on Offering.OfferingID=Configuration.OfferingID
join Provider on Provider.ProviderID=Offering.ProviderID
where (Cost.UsageAmountID is null or Cost.UsageAmountID=
(case when (1=%usageParameter) then Cost.UsageAmountID else (
select distinct Cost.UsageAmountID from Cost
left join UsageAmount as t1 on t1.UsageAmountID = Cost.UsageAmountID
where t1.Min<=%input and t1.Max>=%input and Cost.ConfigID=%configID)end))
and (Location.LocationID=12 or Location.LocationID= case when (0=%
    locationParameter)then
Location.LocationID else 0 end)
and Cost.ConfigID=%configID
```

Listing 4.4: Calculation Method

```java

public double Calculation(String function, String para1, String para2, String
    para3, String para4){
  String resault = String.Format(function, para1, para2, para3, para4);
  MSScriptControl.ScriptControl sc = new MSScriptControl.ScriptControl();
  sc.Language = "JavaScript";
  double value = sc.Eval(resault);
  double cost = Math.Round(value, 2);
  return cost;}
```

Before the last step, either the users input or the initial value will be assigned to every parameter except dynamic unit. At the last step, if the unit is fixed then the formula and its parameter value in each position will be as arguments passed in calculation method, else the data in Max and Min field of query result will be assigned to the corresponding position respectively, and using this to calculate minimum and maximum cost for every usage range. Listing 4.4 show the code for String calculation.

If user defines a usage pattern for a certain period, system should calculate the total cost for every month in the pattern. Because the cost will tend to change as usage amount increase or decrease, especially usage amount has relationship with period. Therefore we need a mapping between the usage of every parameter to their period in months. For that, we have defined a Generic *Usage* to encapsulate the information about parameter, its usage amount and the corresponding month. In one month the change rate of different cost parameters may have different values, in order to obtain the total cost in a whole period the calculation will focus on a Cartesian product result set of months and usage amount of each parameter. This set will be stored in a list and can be used as the argument for the calculation process. Listing 4.5 shows the code of method *UsageConstructor* which can generate a list described above.

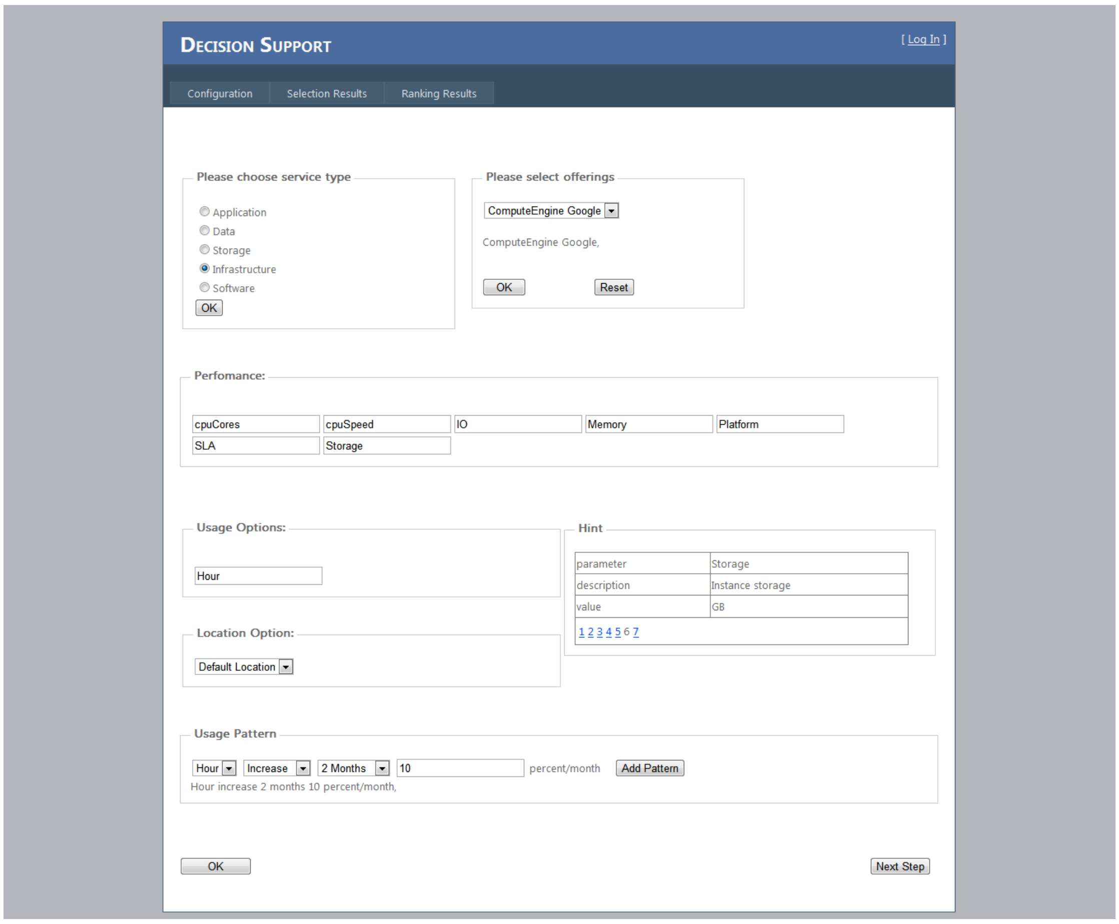
Listing 4.5: Usage Constructor

```

public List<Usage> UsageConstructor()
{
  double usage = 1;
  List<Usage> monthList = new List<Usage>();
  foreach (Unit unit in GlobalVariable.inputList)
  {
    int month = 1;
    if (!unit.Input.Equals(unit.UnitValue))
    {
      usage = double.Parse(unit.Input);
    }
    foreach (Pattern pattern in GlobalVariable.patternList)
    {
      if (pattern.Unit.Equals(unit.UnitValue))
      {
        for (int i = 0; i < Convert.ToInt32(pattern.Period); i++)
        {
          month = month + 1;
          usage = usage * (1 + double.Parse(pattern.Rate) / 100);
          monthList.Add(new Usage(unit.UnitValue, usage.ToString(), month.ToString
              ()));
        }
      }
    }
    if (month > GlobalVariable.monthCounter)
    {
      GlobalVariable.monthCounter = month;
    }
  }
  return monthList;
}
```

4.5 User Interface

A Web application form is created for achieving interaction between user and system. The Frontend runs in ASP.NET environment and contains three pages. Every page expresses a decision step separately. By three menu items on top of the page user can move more easily through sites.



Please choose service type
- ○ Application
- ○ Data
- ○ Storage
- ● Infrastructure
- ○ Software

Please select offerings: ComputeEngine Google

ComputeEngine Google,

Perfomance:

cpuCores	cpuSpeed	IO	Memory	Platform
SLA	Storage			

Usage Options: Hour

Location Option: Default Location

Hint

parameter	Storage
description	Instance storage
value	GB

1 2 3 4 5 6 7

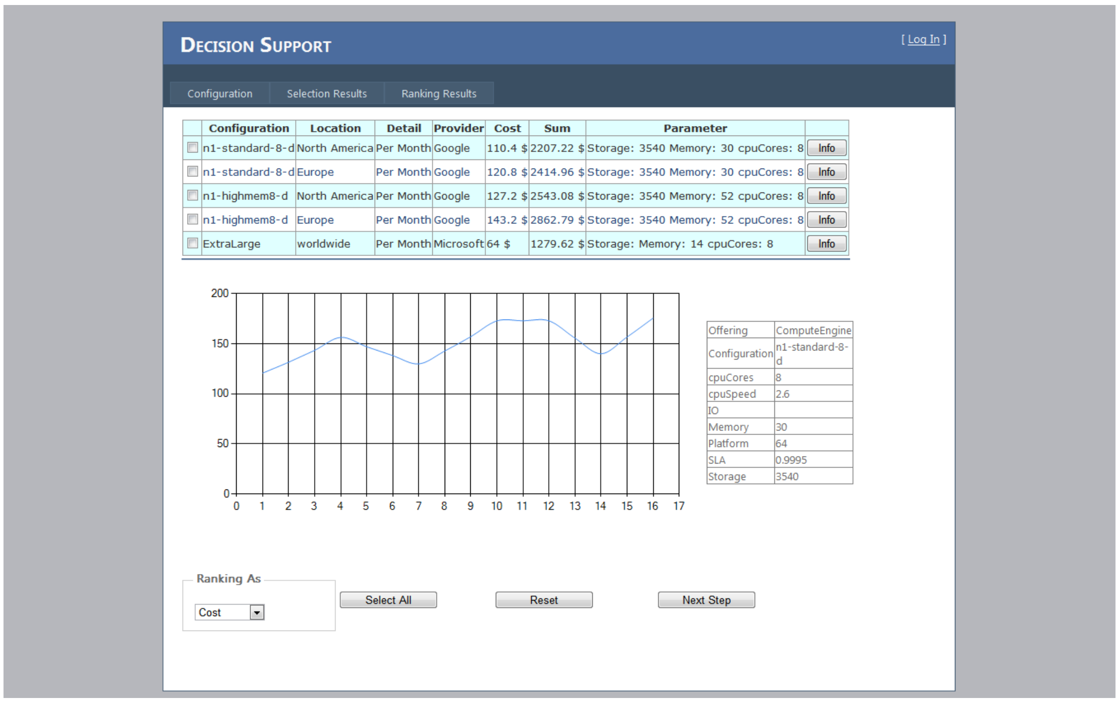
Usage Pattern: Hour | Increase | 2 Months | 10 | percent/month

Hour increase 2 months 10 percent/month,

Figure 4.3: User Interface: Configuration Page

Figure 4.3 shows the *Configuration* page which is used to collect requirements. Through some controls such like radio button, check list and text box users can choose offerings, input their requirements and design usage patterns.

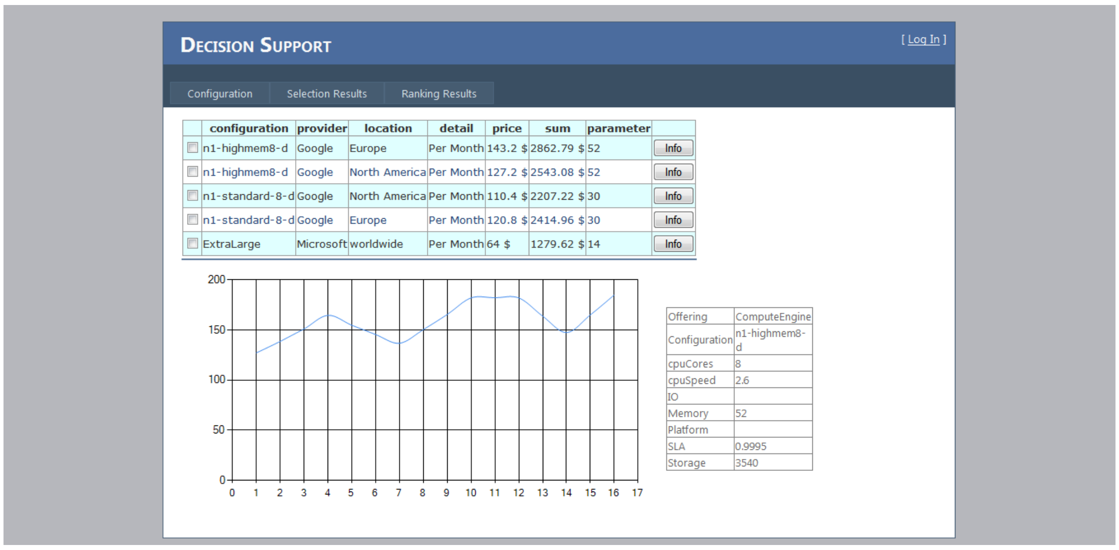
Figure 4.4 is an example for *SelectionResults* page. After collecting necessary data from the first page the system will output its result of calculation on *SelectionResults* page. A list view is applied to describe the details of candidates. Every row in the list expresses a configuration. Its cost and other information are displayed in a separate column. Under this list view we use a diagram with spline to demonstrate cost movements in usage pattern for a certain configuration. Beyond this user can also choose a key parameter and preferred candidates to complete the second step of decision process.



	Configuration	Location	Detail	Provider	Cost	Sum	Parameter	
☐	n1-standard-8-d	North America	Per Month	Google	110.4 $	2207.22 $	Storage: 3540 Memory: 30 cpuCores: 8	Info
☐	n1-standard-8-d	Europe	Per Month	Google	120.8 $	2414.96 $	Storage: 3540 Memory: 30 cpuCores: 8	Info
☐	n1-highmem8-d	North America	Per Month	Google	127.2 $	2543.08 $	Storage: 3540 Memory: 52 cpuCores: 8	Info
☐	n1-highmem8-d	Europe	Per Month	Google	143.2 $	2862.79 $	Storage: 3540 Memory: 52 cpuCores: 8	Info
☐	ExtraLarge	worldwide	Per Month	Microsoft	64 $	1279.62 $	Storage: Memory: 14 cpuCores: 8	Info

Offering	ComputeEngine
Configuration	n1-standard-8-d
cpuCores	8
cpuSpeed	2.6
IO	
Memory	30
Platform	64
SLA	0.9995
Storage	3540

Figure 4.4: User Interface: Selection Results Page



	configuration	provider	location	detail	price	sum	parameter	
☐	n1-highmem8-d	Google	Europe	Per Month	143.2 $	2862.79 $	52	Info
☐	n1-highmem8-d	Google	North America	Per Month	127.2 $	2543.08 $	52	Info
☐	n1-standard-8-d	Google	North America	Per Month	110.4 $	2207.22 $	30	Info
☐	n1-standard-8-d	Google	Europe	Per Month	120.8 $	2414.96 $	30	Info
☐	ExtraLarge	Microsoft	worldwide	Per Month	64 $	1279.62 $	14	Info

Offering	ComputeEngine
Configuration	n1-highmem8-d
cpuCores	8
cpuSpeed	2.6
IO	
Memory	52
Platform	
SLA	0.9995
Storage	3540

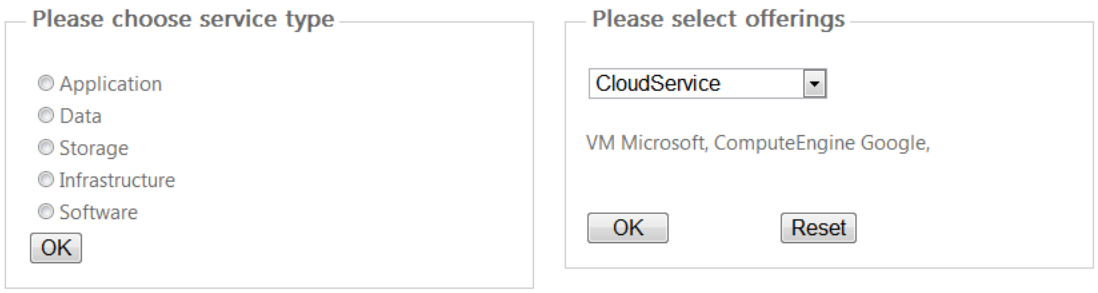
Figure 4.5: User Interface: Ranking Results Page

The last step of decision process is presented in Figure 4.5. Similar with *SelectionResults* page, this *RankingResults* page uses a list view to demonstrate the results from Back-end, but a significant difference is that the sequence of items in list is ranked by a key parameter.

4.6 User Guide

4.6.1 Selection Of Offerings

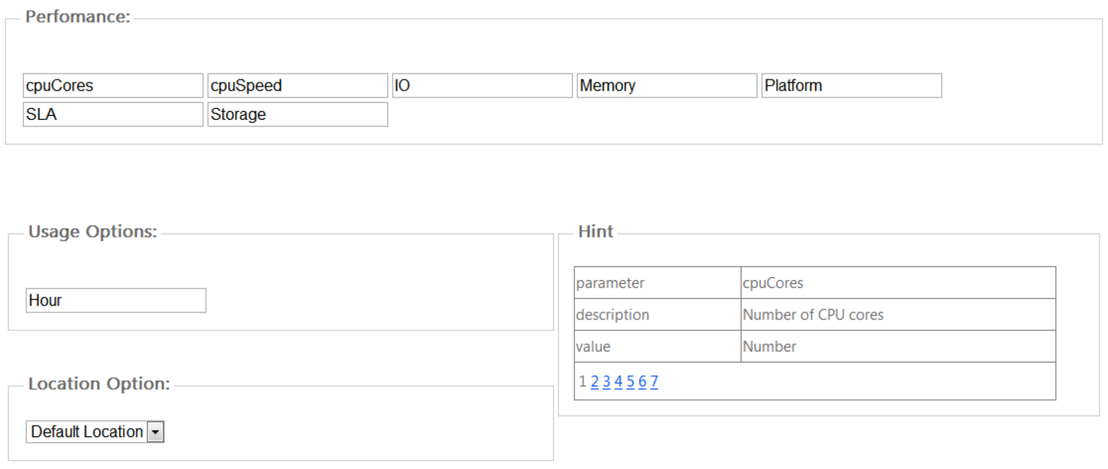
Through choosing one of the service type a list of corresponding offerings will be showed in the *DropDownList*. The selected offerings will be displayed on form. Service type of *Application* points to PAAS, *Data* represents data transfer during network access, *Storage* includes relation database service and data storage service, *Infrastructure* contains visual machine instance and *Software* is aimed at SAAS. Figure 4.6 shows the related controls for selecting offering.



VM Microsoft, ComputeEngine Google,

Figure 4.6: User Guide: Selection Of Offerings

4.6.2 Setting Of Parameters



parameter	cpuCores
description	Number of CPU cores
value	Number

Figure 4.7: User Guide: Setting Of Parameters

Figure 4.7 gives an overview of setting interface. This step includes three parts. First user assigns values to parameters displayed in panel *Performance*, these parameter will be used to match suitable service. If user does not define the value of a particular parameter, system will assign a default value to ignore the effect of this parameter in the matching process. Then the usage amount in a basic unit time can be defined in panel *UsageOptions*. If an entering is empty, system will define usage with smallest unit for fixed unit-price and in case of dynamic unit-price the minimum as well as maximum usage amount in each range will be listed automatically. At last, location as a characteristic parameter is able to be selected in panel *LocationOptions*. Option of *DefaultLocation* means avoiding restriction of location. Additionally, in panel *Hint* the information prompts the user to enter a specific type or correct format of requirements.

4.6.3 Usage Pattern

User can define usage patterns to forecast total cost in periods. Figure 4.8 shows the options for usage pattern. The first check list represents type of parameters which belongs to usage options. This means that patterns of every parameter will be created separately. Next one is used to determine a trend of change. Option *Period* contain months from 1 to 12. The change rate can be assigned by the text box. For any parameter, a period in subsequent pattern will be strictly connected with the period in previous defined pattern that they must be the same parameter. For example, the period in first pattern is 3 months with increase rate of 15% per month and the period in second pattern contains 5 months with decrease rate of 5% per month. Therefore, the total period is 9 month, peak appears in the fourth month and the usage from fifth month will start to decline and over the next four months. For one offering, if total periods of multiple parameters are different, from boundary point system will complete the missing months with invariant trend for the parameters whose total period is shorter. In a similar case, if there are no patterns defined for a particular parameter, its trend will be considered as invariant and the corresponding period is as same as the longest period of patterns which belongs to the same offering.

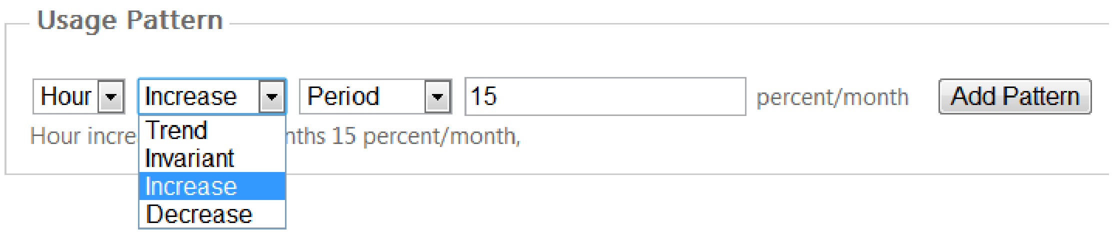
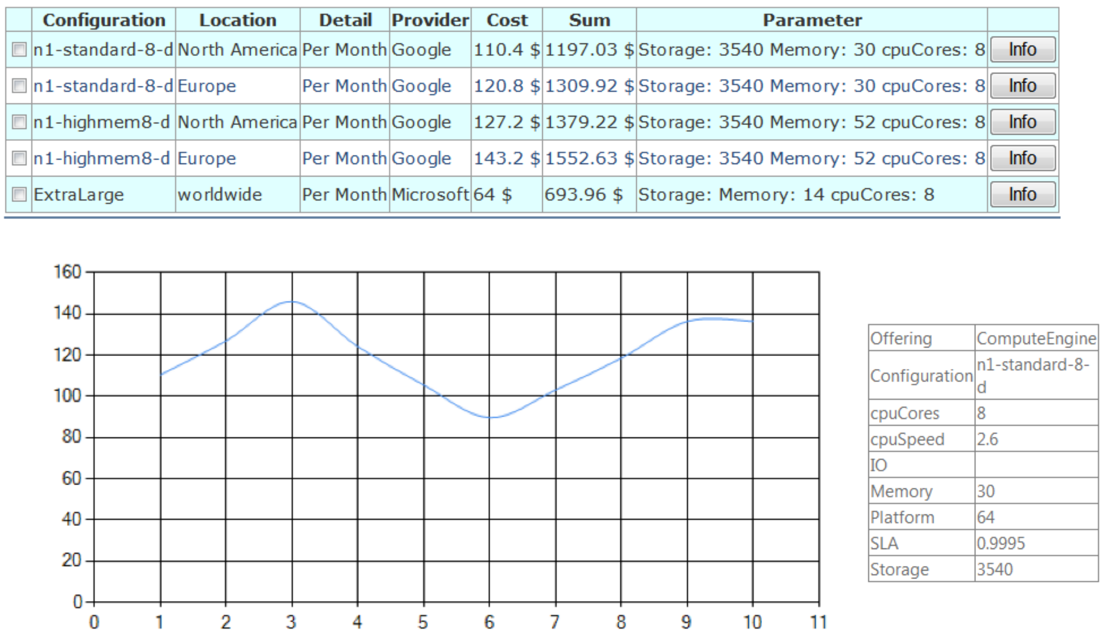


Figure 4.8: User Guide: Usage Pattern

4.6.4 Results Of Calculation And Selection For Ranking

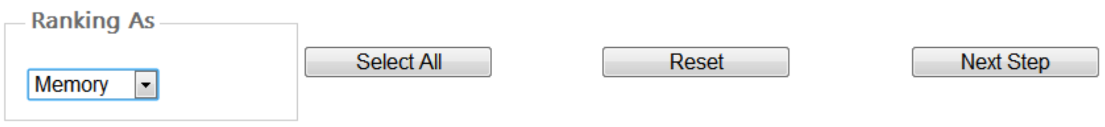
Figure 4.9 is the results of calculation which are listed in a table. Every row represents a configuration. The basic cost, total cost, specific parameters and other necessary information retrieved from knowledge base will be indicated in different columns. By clicking the *Info* button which is at the end of each row a diagram with spline and a table of details will be displayed under the results table. The x- and y-coordinates of the diagram contain values of months and cost respectively. This diagram enables users to watch the trend of estimate cost in the whole period. Additionally, there is a table to show all parameter details for a selected configuration. One or more configurations can be chosen for ranking by clicking the check box at the head of each row. User can also click the *All* button (Figure 4.10) to enable selecting all configurations. After user has chosen their preferred configurations as candidates, system needs a key parameter to start ranking process that can be completed with a check list at the bottom of page (Figure 4.10). This check list contains the options of parameter which can be used for ranking.



	Configuration	Location	Detail	Provider	Cost	Sum	Parameter	
☐	n1-standard-8-d	North America	Per Month	Google	110.4 $	1197.03 $	Storage: 3540 Memory: 30 cpuCores: 8	Info
☐	n1-standard-8-d	Europe	Per Month	Google	120.8 $	1309.92 $	Storage: 3540 Memory: 30 cpuCores: 8	Info
☐	n1-highmem8-d	North America	Per Month	Google	127.2 $	1379.22 $	Storage: 3540 Memory: 52 cpuCores: 8	Info
☐	n1-highmem8-d	Europe	Per Month	Google	143.2 $	1552.63 $	Storage: 3540 Memory: 52 cpuCores: 8	Info
☐	ExtraLarge	worldwide	Per Month	Microsoft	64 $	693.96 $	Storage: Memory: 14 cpuCores: 8	Info

Offering	ComputeEngine
Configuration	n1-standard-8-d
cpuCores	8
cpuSpeed	2.6
IO	
Memory	30
Platform	64
SLA	0.9995
Storage	3540

Figure 4.9: User Guide: Results Of Calculation



Ranking As: Memory

Figure 4.10: User Guide: Options For Ranking

4.6.5 Results Of Ranking

The configurations which are selected from the last step will have a sequence in the table. They are sorted by magnitude of key parameter. If key parameter is cost, the sequence is from smallest to largest, otherwise will be reverse. As same as the table with results of calculation (Figure 4.9), user can get cost and parameter details by clicking *Info* buttons at the end of each row.

5 Evaluation

In this chapter some test cases for offering matching and cost calculation will be used to validate the results of data handling. Then we will compare the functionality of MDSS with some online available decision support system and calculation toolkit. At last MDSS is compared with some migration support frameworks which we have discussed in Section 2.3.

5.1 Validation

As the first step in the decision process, offering matching should filter out unsuitable configurations and generate a list which contains the feasible candidates. MDSS can compare each requirement defined by user against all of the offerings data in data base. The implementation of matching component requires support of a provider knowledge base. We choose the results of *PlanForCloud* [25] as the comparison data, because its knowledge base supports more than one provider.

Table 5.1: Validation of Offering Matching

Migration Support System			**PlanForCloud**	
Google		Microsoft Azure	Google	Microsoft Azure
US	EU	Worldwide		
n1-standard-8-d	n1-standard-8-d	ExtraLarge	n1-standard-8-d	ExtraLarge
n1-standard-8	n1-standard-8		n1-standard-8	
n1-highmem-8-d	n1-highmem-8-d		n1-highmem-8-d	
n1-highmem-8	n1-highmem-8		n1-highmem-8	

In the test case user wants to select a VM service among Windows Azure and Google Computer Engine. The multi-criteria restriction conditions are that CPU must have at least 8 cores and RAM must reach at least 8 GB. Table 5.1 shows the results from these two systems. According to MDSS Google has 4 configurations that satisfy the test requirements and Window Azure has one feasible configuration. The output by *PlanForCloud* has the same candidates but a noticeable difference between them is that each configuration has a location information by MDSS. An instance pricing will depend on the location where it has been hosted by Google, although they have the same performance. Therefore location is an important parameter in characteristic factor and has been considered by MDSS. By the results of the two systems we can see that both of them support multi-criteria matching and more parameters can be taken into account by MDSS.

Through checking the official pricing details by Google [26] and Windows Azure [23] we can validate that both of these two systems can recommend correct candidate after matching requirements.

Because unit-price consists of fixed and dynamic case, we will handle the validation of cost calculation with both cases separately. First we focus on calculating with fixed unit-price. Suppose that a user wants to purchase VM service from Windows Azure or Google Compute Engine. Usage amount as an input parameter will be set to 720 hours per month. We will compare the calculated cost of each configuration which belongs to Google and Azure VM offering. Comparison data is from *Microsoft Price Calculator* (MPC) which is an official cost calculator and from *PlanForCloud* (PFC) as third-party data.

Table 5.2 shows the results of calculation from the three tools. For the instances by Google, MDSS contains cost distinction between US and EU, but PFC does not have the related location information, hence location parameter is a cause of differences which exist among their results. But when we compute the average value of US and EU cost from results of MDSS, we find that dissimilarity still exists , and the ratio of difference on average value to the results of PFC is floating between 2% and 3%. As we have no information about the PFC knowledge base, therefore, we assume that the basic unit-price of PFC may be different from MDSS. For each instance in Azure, the results of calculation from MDSS are identical to the data from MPC. But PFC has the differences, and the maximum ratio of difference reaches 59%. Therefore we still surmise that the basic unit-price is the main cause of the difference in calculation of cost. After the comparison of calculation results, we may reasonably come to the conclusion that MDSS is able to make accurate estimates for fixed unit-price.

Table 5.3 shows an overview of the comparison of calculation results with dynamic unit-price. In the test cases, the user has decided to choose a Cloud storage offering from Azure or Google. Because the unit-price and calculation formula will be different with variation of usage amount, the storage capacity is chosen in each usage range randomly as input value. The comparison data are obtained from PFC and MPC.

In the case with offering *Google Cloud Storage*, the difference between MDSS and PFC is less than 0.003% of cost in each usage range. For Azure *Storage Geographically Redundant*, the results by these three tools are exactly the same in the first usage range with values of 0.525TB, but from the second usage range a difference appears and begins to rise from the third range. Although the absolute value of the difference increases as the usage amount rises, the percentage is falling, and the ratio of difference on values from MDSS to values from the other two tools is floating between 0.001% and 0.003%. Except from the precision of calculation and rounding-off reasons, a probable cause of difference in aggregate total cost is that the basic unit-price in knowledge base may have different

Table 5.2: Validation of Cost Calculation: Fixed Unit Price

Provider	**Configuration**	**MDSS**		**PFC**	**MPC**	Unit Price	
		US	EU			US	EU
Google	n1-standard-1-d	$99.36	$108.72	$102.67		$0.138	$0.151
	n1-standard-2-d	$198.72	$217.44	$205.34		$0.276	$0.302
	n1-standard-4-d	$397.44	$434.88	$410.69		$0.552	$0.604
	n1-standard-8-d	$794.88	$869.76	$821.38		$1.104	$1.208
	n1-standard-1	$86.4	$95.04	$89.28		$0.12	$0.132
	n1-standard-2	$172.8	$190.08	$178.56		$0.24	$0.264
	n1-standard-4	$345.6	$380.16	$357.12		$0.48	$0.528
	n1-standard-8	$691.2	$760.32	$714.24		$0.96	$1.056
	n1-highmem-2-d	$228.96	$257.76	$236.59		$0.318	$0.358
	n1-highmem-4-d	$457.92	$515.52	$473.18		$0.636	$0.716
	n1-highmem-8-d	$915.84	$1031.04	$946.37		$1.272	$1.432
	n1-highmem-2	$182.88	$205.92	$188.98		$0.254	$0.286
	n1-highmem-4	$365.76	$411.84	$377.95		$0.508	$0.572
	n1-highmem-8	$731.52	$823.68	$755.9		$1.016	$1.144
	n1-highcpu-2-d	$122.4	$138.24	$126.48		$0.17	$0.192
	n1-highcpu-4-d	$244.8	$276.48	$252.96		$0.34	$0.384
	n1-highcpu-8-d	$489.6	$552.96	$505.92		$0.68	$0.768
	n1-highcpu-2	$97.92	$109.44	$101.18		$0.136	$0.152
	n1-highcpu-4	$195.84	$218.88	$202.37		$0.272	$0.304
	n1-highcpu-8	$391.68	$437.76	$404.74		$0.544	$0.608
Microsoft	ExtraSmall	$9.36		$14.88	$9.36	$0.013	
	Small	$57.6		$63.24	$57.6	$0.08	
	Medium	$115.2		$126.48	$115.2	$0.16	
	Large	$230.4		$252.96	$230.4	$0.32	
	ExtraLarge	$460.8		$505.92	$460.8	$0.64	

value. Taking into account of the validation data, we may reach the conclusion that in case of dynamic unit-price MDSS can output the calculated value with low deviations when compared to the official provider calculator and a third party calculator.

Table 5.3: Validation of Cost Calculation: Dynamic Unit Price

Usage	Offering	MDSS	PFC	MPC	Unit Price
0.525 TB	Azure Storage Geographically Redundant	$49.88	$49.88	$49.88	$0.095/GB
	Google Cloud Storage	$44.62	$44.63		$0.085/GB
5.125 TB	Azure Storage Geographically Redundant	$425	$425.36	$425.36	$0.08/GB
	Google Cloud Storage	$398.5	$398.72		$0.076/GB
80.075 TB	Azure Storage Geographically Redundant	$6120.25	$6132.61	$6132.53	$0.07/GB
	Google Cloud Storage	$5464.2	$5464.4		$0.067/GB
500 TB	Azure Storage Geographically Redundant	$35515	$35527.36		$0.065/GB
	Google Cloud Storage	$31999	$32010		$0.063/GB
2000 TB	Azure Storage Geographically Redundant	$128015	$128207.36		$0.06/GB
	Google Cloud Storage	$112999	$113118.98		$0.054/GB

5.2 Comparison with Existing Tools

We will compare the functionality of MDSS with that of the tools which are already on-line available. As we have discussed in Section 2.4, some providers have offered cost calculator tools like Azure Calculator or TCO by Amazon and some third-party companies put related decision support products into the market like PlanForCloud which plays a role as expert consultant in choosing Cloud services. Since a knowledge base is a cornerstone of decision support, and performance, characteristic as well as cost are the three most important factors in specification of offerings, the comparison with existing tools will start with following aspects: knowledge base for multiple providers, offering matching, cost calculation, design deployment and design usage pattern. Table 5.4 gives an overview of the features of the existing tools.

- Knowledge base for multiple providers

  There exists a large number of Cloud service offerings with similar functionalities on market, and a knowledge base is typically used to classify and identify these

Table 5.4: Comparison with Existing Tools

System	**Knowledge Base For Multiple Provider**	**Offering Matching**	**Cost Calculation**	**Design Deployment**	**Design Usage Pattern**
MDSS	yes	yes	yes	no	yes
PFC	yes	yes	yes	yes	yes
MPC	no	no	yes	no	no
TCO	no	no	yes	yes	yes

offerings with their parameters. PFC supports the latest cost and performance parameter for multiple providers. Similarly for MDSS, a knowledge base is designed and implemented by relation database whose data is the core of the match and calculation process. The knowledge base of TCO and MPC only include the information about their own offerings, therefore the comparison with different providers is impossible.

- Offering matching

  MDSS enables users to add multi-criteria requirements which cover the performance and characteristic parameters. The component of offering matcher is able to pack the requirements as arguments to retrieve the necessary data from knowledge base. Through querying the database and manipulating the retrieved data, users can obtain the output which is generated with their demands. PFC allows user to choose configurations according to the parameters, but a limitation of PFC is that only a few of the most basic parameters are covered. As a calculator tool TCO and MPC cannot recommend suitable configuration for user.

- Cost calculation

  By a given usage amount TCO, MPC and PFC can make an accurate cost estimate for the user preferred configuration. MDSS is not only able to calculate cost based on user defined usage amount, but also can output the expenses in each usage range, if no value of usage is entered by the user. Notably, when compared with other tools, more parameters in cost calculation are taken into account by MDSS, for example the location information as a non-numerical parameter has effect on the unit-price. Additionally, some offerings have more than one numerical parameter such like storage service whose cost consists of number of transaction and storage capacity. MDSS allows cost formulas to import multiple parameters and obtains total cost but MPC calculates and outputs the cost of these parameters separately.

- Design deployment

TCO and PFC enable users to design a Cloud deployment which contains server, storage, database and data transfer. But MDSS can only calculate the cost of these Cloud resources separately. Therefore, TCO and PFC can be used to support more types of migration as discussed in Section 2.1.

- Design usage pattern

  A significant advantage of Cloud computing is elastic consumption on demand. Hence usage pattern design is a practical feature to help users check out the costs in the long term. In TCO users can choose three kinds of patterns: spiky predictable, uncertain unpredictable and steady state. But the limitation is that user cannot set the change rate or period, if user has already determined when a peak will appear. PFC and MDSS enables user to attach patterns which defined with flexible change rate and period.

5.3 Comparison with Existing Research Frameworks

As we have discussed in Section 2.3, some related works introduced several frameworks which are aimed at the fields of decision support and application migration. We will compare MDSS with these frameworks. Table 5.5 gives an overview of the comparison.

Table 5.5: Comparison with Existing Research Frameworks

Frameworks and Systems	**Highlights**	**Factors Taken Into Account**	**Ranking Methods**
MDSS	Offering selection, Cost calculation, Ranking	Performance, Characteristic, Cost	Ranking by key parameter
CloudGenius	Offering selection, Combination between VM and infrastructure service, Ranking	Performance, Characteristic, Cost,Provider	AHP
$(MC^2)^2$	Alternative selection, Ranking	Performance, Characteristic, Cost,Provider	ANP
Cloud Adoption Toolkit	Suitability analysis, Cost modeling, Stakeholder impact analysis	Performance, Characteristic, Cost,Social and political factors	
SMICloud	Ranking	Performance, Characteristic, Cost	AHP

From the table we can see that every framework consider the performance, characteristic and cost factors. But a lot of parameters produced by these frameworks are not covered in MDSS, because the measurement and evaluation of these parameters are difficult and not accurate. For example, in $(MC^2)^2$ the factor cost involves more parameters like training costs, insurance costs or regulation costs, and in MDSS the factor cost only contains purchase. However, MDSS takes the most important parameters into account which are able to impact the results of offering selection.

The combination between VM and Cloud infrastructure service in CloudGenius is a marked difference from MDSS. According to user requirements, CloudGenius first focuses on the selection of image and infrastructure service separately and builds all possible pair, then evaluate the feasible pair to generate the best combination. In MDSS, when users select candidates for using infrastructure service, system offers all necessary options based on requirements. In other words, the compatibility is described as a parameter in MDSS and used to match appropriate offerings.

Cloud Adoption Toolkit (CAT) applies a suitability analysis to filter out infeasible service. The suitability analysis involves a check list which covers the main issues during migration. MDSS collects users requirements and compares with the service parameter stored in knowledge base to output the suitable services. Both of CAT and MDSS focus on cost calculation and can estimate the cost in the long term with usage pattern.

The key points in each existing research frameworks are the selection of appropriate offerings or cost calculation. However, they are not implemented as an on-line available system or do not have a provider knowledge base. Therefore we cannot compare the results of MDSS with the other frameworks.

As another highlight, the ranking method in SMICloud is based on AHP. CloudGenius applies also AHP and $(MC^2)^2$ uses ANP to achieve ranking. But how to select the weights for parameters is still a problem, therefore MDSS ranks the services by the value of a key parameter. This enables users to check out the list of offering candidates with a dynamic sequence which is specified based on the key parameter.

6 Conclusions

6.1 Summary

Cloud computing poses both an opportunity and a challenge for enterprises. It allows consumers to access computing resources through the network elastically; and it enables enterprises to put in more capital and human resources to develop their business. The Cloud computing market is booming, driven by the benefits of Cloud computing and the corresponding maturing technologies. But at the same time, a large number of Cloud service offerings with similar feature and different pricing make it difficult for consumers to select suitable solutions. A rational decision helps consumers to mitigate the risk while their existing applications are migrating to the Cloud. Therefore, we propose a system to realize the decision support process, by which consumers are able to obtain recommendations on selecting appropriate configurations.

For achieving the goal, we have first studied the related works in the fields of decision support and migration systems. The similarities and differences of the parameters considered by these works are summarized in this thesis. With the collected data and information a knowledge base for Cloud provider is created. The design of this knowledge base takes the extensibility and scalability into account. Furthermore, it has the official data captured from some Cloud providers and covers the parameters which we have identified. We implemented the proposed system as a Web application. The Frontend contains three Web forms which allow the user to enter their requirements and check the returned results; the Backend involves the offerings matching and cost calculation procedures, by which the infeasible configurations will be filtered out, then the estimate basic monthly cost and the forecast total cost with usage pattern will be calculated respectively.

In order to validate the outputs of implemented system, we set up the tests with some cases and analyze the results of them. We choose an official cost calculator MPC from Microsoft and a third-party migration support system PlanForCloud, their outputs will be used as the data in comparative analysis. As an important feature of MDSS, the results of cost calculation are validated by comparison with other systems. Finally the functionalities of our system are compared with several existing tools for the evaluation. MDSS have realized most of the functionalities and covered the most parameters which should be taken into account during application migration.

To conclude, this thesis introduces a decision support system for migration application in the Cloud, by which consumers can efficiently and accurately choose the suitable configuration based on their requirements.

6.2 Future Work

While writing this thesis, we summarized the limitations and the features which can be extended in the future work.

- Cover more migration types

  The cost calculation and ranking do not support the combination of different components in this thesis, not all migration types described in Section 2.1 are covered. Up to now, the migration types of *Replace* and *Migrate the whole software stack* are applicable. As a proposed solution to deal with this limitation, a procedure should be designed to store and manipulate the encapsulated cost of each service in a deployment pattern, and the deployment pattern can be defined as a Generic to achieve the comparison between different deployment patterns.

- Creation of the profiles for multiple services

  In this thesis, the implemented system can only consider one type of the offerings at each time. In other words, we can only focus on one of the components in a given deployment pattern. We suggest that, the system allows user to create several profiles which involve all components. For every profile, user can choose the feasible services for each component. Therefore, all profiles have different costs or performance, and the optimal one will be recommended. To achieve this, the cost of each component should be stored as an attribute in a new defined Generic *Profile*.

- Extension of the number of providers in the knowledge base

  By now the implemented knowledge base contains two providers. It should be extended with more data of other providers (like Amazon) to afford a further wide field of application.

- Horizontal elasticity

  Another point should be considered in the future is that the usage pattern should reach horizontal elasticity [6], this means users can not only define the change trend of computational resource, but also specify the change numbers of instance or storage. To achieve this, more parameters should be defined when the corresponding procedures generate the instance of Generic *Pattern*.

Bibliography

[1] B. Suleiman, S. Sakr, R. Jeffery, and A. Liu, "On understanding the economics and elasticity challenges of deploying business applications on public cloud infrastructure," *Journal of Internet Services and Applications*, pp. 1–21, 2011.

[2] R. Buyya, C. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.

[3] B. Tak, B. Urgaonkar, and A. Sivasubramaniam, "To move or not to move: The economics of cloud computing," in *Proceedings of the 3rd USENIX conference on Hot topics in cloud computing*, pp. 5–5, USENIX Association, 2011.

[4] E. Walker, W. Brisken, and J. Romney, "To lease or not to lease from storage clouds," *Computer*, vol. 43, no. 4, pp. 44–50, 2010.

[5] E. Walker, "The real cost of a cpu hour," *Computer*, vol. 42, no. 4, pp. 35–41, 2009.

[6] V. Andrikopoulos, T. Binz, F. Leymann, and S. Strauch, "How to adapt applications for the cloud environment - challenges and solutions in migrating applications to the cloud," *Computing (to appear)*, 2013.

[7] M. Armbrust, A. Fox, R. Griffith, A. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, *et al.*, "Above the clouds: A view of cloud computing," tech. rep., Technical report, 2010.

[8] L. Badger, T. Grance, R. Patt-Corner, and J. Voas, "Draft cloud computing synopsis and recommendations," *NIST Special Publication*, vol. 800, p. 146, 2011.

[9] W. Zeng, Y. Zhao, and J. Zeng, "Cloud service and service selection algorithm research," in *Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, pp. 1045–1048, ACM, 2009.

[10] S. Garg, S. Versteeg, and R. Buyya, "Smicloud: A framework for comparing and ranking cloud services," in *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pp. 210–218, IEEE, 2011.

[11] M. Zeleny, *Multiple criteria decision making*, vol. 25. McGraw-Hill New York, 1982.

[12] F. Hussain, O. Hussain, *et al.*, "Towards multi-criteria cloud service selection," in *Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS), 2011 Fifth International Conference on*, pp. 44–48, IEEE, 2011.

[13] H. Chan and T. Chieu, "Ranking and mapping of applications to cloud computing services by svd," in *Network Operations and Management Symposium Workshops (NOMS Wksps), 2010 IEEE/IFIP*, pp. 362–369, IEEE, 2010.

[14] M. Menzel, M. Schönherr, and S. Tai, "(mc2) 2: criteria, requirements and a software prototype for cloud infrastructure decisions," *Software: Practice and Experience*, 2011.

[15] A. Khajeh-Hosseini, I. Sommerville, J. Bogaerts, and P. Teregowda, "Decision support tools for cloud migration in the enterprise," in *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pp. 541–548, IEEE, 2011.

[16] A. Khajeh-Hosseini, D. Greenwood, J. Smith, and I. Sommerville, "The cloud adoption toolkit: supporting cloud adoption decisions in the enterprise," *Software: Practice and Experience*, vol. 42, no. 4, pp. 447–465, 2012.

[17] M. Menzel and R. Ranjan, "Cloudgenius: decision support for web server cloud migration," in *Proceedings of the 21st international conference on World Wide Web*, pp. 979–988, ACM, 2012.

[18] A. Iosup, N. Yigitbasi, and D. Epema, "On the performance variability of production cloud services," in *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pp. 104–113, IEEE, 2011.

[19] A. Li, X. Yang, S. Kandula, and M. Zhang, "Cloudcmp: comparing public cloud providers," in *Proceedings of the 10th annual conference on Internet measurement*, pp. 1–14, ACM, 2010.

[20] aotearoa, "aotearoa." `http://aotearoadecisions.appspot.com/`, November 2012.

[21] Microsoft, "Windows azure pricing calculator." `http://www.windowsazure.com/en-us/pricing/calculator/`, November 2012.

[22] Amazon, "Tco comparison calculator for web applications." `http://tco.2ndwatch.com/`, November 2012.

[23] Microsoft, "Pricing details." `http://www.windowsazure.com/en-us/pricing/details/?l=en-us`, November 2012.

[24] Google, "Google cloud sql pricing." `https://cloud.google.com/pricing/cloud-sql`, November 2012.

[25] R. Scale, "Cloud cost forecasting web site from right scale." `http://www.planforcloud.com/`, November 2012.

[26] Google, "Google compute engine." `https://cloud.google.com/pricing/compute-engine`, January 2013.

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, Unterschift