Resource Provisioning in Clouds via Non-Functional Requirements

By

Diana Carolina Barreto Arias

Under the supervision of

Professor Rajkumar Buyya

and

Dr. Rodrigo N. Calheiros

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

Department of Computing and Information Systems
The University of Melbourne

November, 2013

Resource Provisioning in Clouds via Non-Functional Requirements

Diana Carolina Barreto Arias

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

---

ABSTRACT

Cloud computing, in particular its Infrastructure as a Services model, allows system administrators to obtain resources to deploy their applications and to pay just for the resources that are consumed. Therefore, local data centers are replaced by remote infrastructures. As a consequence, the specific hardware expertise required from system administrators is reduced. However, cloud computing requires new skills. Estimating resources that the applications require to run in the cloud is a complex task due to system administrators frequently do not know the low-level technical details of what they really need. Furthermore, the estimations can vary depending on the cloud provider characteristics. Selecting the more adequate provider for a particular application is also a laborious task because there are a huge number of services offered by a considerable number of providers that are not directly comparable. In order to assist system administrators in this hard labour, we propose an architecture that supports the process of deploying applications in cloud providers, using high level information based on non-functional requirements. Experiments with a small scale prototype compared our selection process based on non-functional requirements with one based in price. The results demonstrate that our architecture, considerably improves the process.

Declaration

I certify that

– this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person where due reference is not made in the text.

– where necessary I have received clearance for this research from the University's Ethics Committee and have submitted all required data to the Department.

– the thesis is less than 9000 words in length (excluding text in images, table, bibliographies and appendices).

Diana Carolina Barreto Arias

Nomvember, 2013

Acknowledgement

My thankfulness is first for the Heavenly Father because it was his willing to give me the opportunity, the knowledge and the understanding to join this master and to develop the project presented in this thesis. I also want to express my gratefulness to Professor Rajkumar Buyya, who give the opportunity to work in this project, and with his wide knowledge introduce me to the field of cloud computing, giving me inspiration and motivation. My special thanks are for Dr. Rodrigo N. Calheiros who was my constant support and guide. He generously shared his expertise to provide me the right direction to address the issues that I face in the development of this project. Thanks to his support, I finished successfully the work that I am presenting today.

I would like also thank my husband, that with patient and love encourage me to pursuit my goals and my family that from the distance were supporting me to make possible this dream.

Contents

1 Introduction **1**

1.1 Cloud Computing Overview 2

1.2 Automated Decision System for Efficient Resource Selection and Allocation in Inter-Clouds. 3

1.3 Public Cloud Providers 4

1.4 Problem Statement 5

2 Related Work **7**

2.1 Cloud Monitoring Services 7

2.2 Techniques for Application Resources Estimation 7

2.3 Tools to Estimate and Compare Resources in Cloud Providers 9

3 System Design **13**

3.1 Non Functional Requirements 13

3.2 Cloud Resource Estimator 16

3.2.1 Database Tier 16

3.2.2 Business Tier 18

3.2.3 Presentation Tier 20

3.3 Resource Selection Decision Maker 21

4 Implementation **23**

4.1 Technical Details 23

4.2 Cloud Resource Estimator 23

4.3 Candidate Providers Prioritization 25

5 Evaluation **28**

5.1 Cloud Resource Estimator 28

5.2 Resource Selection Decision Maker 29

6 Conclusion and Future Directions **33**

List of Figures

1 System Context. 2

2 Issues faced by system administrators, addressed via non-functional requirements. 6

3 Software system provisioning via non-functional requirements. 14

4 Workflow to evaluate non-functional requirements. 16

5 Entity relation model of non-functional requirements. 17

6 Example of the data stored. 17

7 Entity relation model of application profile. 18

8 Class diagram business tier. 19

9 Steps executed by the evaluator *NFEvaluatorEfficiency*. 21

10 Example of constraints to filter the candidate cloud providers. 21

11 Extension of the database schema of the Resource Selection Decision Maker. 22

12 Picklist for adding, removing and ordering non-functional requirements. . . 24

13 Relation App_nf_requirement. 24

14 Resource Estimation Request. 25

15 Resource Estimation Result. 26

16 Relations Provider and NFR_Evaluation. 26

17 Candidate providers order by non-functional requirements. 27

18 Set of candidate providers to be priorized. 29

19 Set of candidate provider order by non-functional requirements. 30

List of Tables

1 Differences in the computing services offered by Windows Azure, Amazon EC2 and GoGrid. 5
2 Monitoring services. 8
3 Resource estimation techniques. 9
4 Tools to estimate and compare resources in cloud providers. 12
5 Initial non-functional requirements to consider in the system. 14
6 Synthetic evaluation of non-functional requirements. 30
7 Non-functional requirements prioritized by order. 30
8 Set of candidate providers to prioritized. 31
9 Set of candidate providers order by price. 31
10 Set of candidate providers order by % uptime. 32
11 Set of candidate providers order by response time. 32

1 Introduction

Cloud computing is a revolutionary technology that changed the way as computing and storage resources are acquired. If a company needs resources to deploy their applications, it is no longer necessary to make a great investment in infrastructure and resources management, because there are significant number of cloud providers that can offer a solution for specific requirements [3].

The fact that computing resources are not maintained in local infrastructures but in cloud provider data centers suggests a considerable reduction in the number of tasks that need specific hardware expertise [9]. However, system administrators, working in cloud computing require new skills to be able to perform different tasks that include the estimation of application resources, the selection of cloud providers, and the acquisition of computing and storage resources in the selected clouds.

The tasks mention before are complex due to the large number of variables that have to be considered to achieve the right balance between the cost and the performance, expected from the users when they acquire resources in cloud providers. Furthermore, frequently system administrators do not know the low-level technical details of the resources that they need. For example, administrators who use a local data center in many cases have extra resources to avoid lack of infrastructure and to allow their applications to work correctly. Moreover, the resources required to deploy applications in a local data center can differ from the ones required to deploy the applications in the cloud.

In this thesis, we propose the design of an automated solution that supports system administrators in the execution of these tasks, reducing requirements of specific knowledge and helping companies to take the decision of moving their applications to public cloud providers.

We introduce this thesis providing a description of the system context, that as is shown in Figure 1, includes a cloud computing overview, the description and the limitations of an existent solution to select and allocate resources in cloud providers and a general description of the services offered by public cloud providers. After presenting the system context, we state the problem to be solved with the architecture proposed.

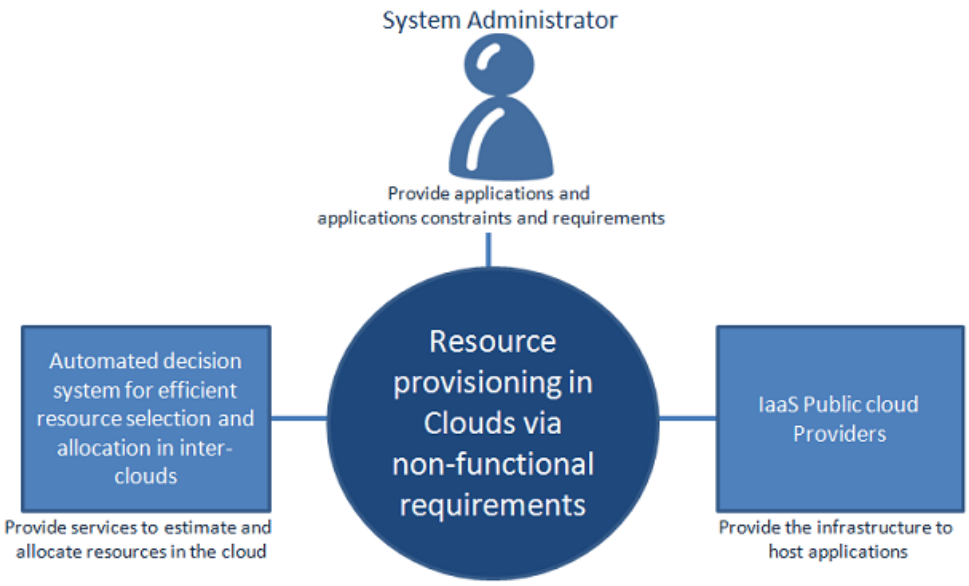


Figure 1: System Context.

1.1 Cloud Computing Overview

Cloud computing has a set of attractive features that promise to change the way that IT services are accessed [3]. Among many features, the most outstanding are the possibility of using computing services as utility, which means accessing services by demand and paying as you go; the promise of rapid elasticity that enables users to increase the infrastructure when it is necessary, having the idea of unlimited resources, and the possibility of saving money in infrastructure and resources management. Additionally, cloud computing also offers broad network access, so required resources can be obtained through networks such as Internet, and the possibility of measure the quality of the services provided [3, 21].

The services that can be offered through the cloud are diverse; therefore they have been classified in the following services models [21]: Software as a Service (SaaS), that corresponds to services that offer diverse kind of applications to be accessed through the cloud, Platform as a Service (PaaS), that means to provide platforms with complete tools to develop applications, and Infrastructure as a Service (IaaS), that is related to provide computing resources such as storage, processing, and network.

The main objective of this thesis is to raise a solution to support system administrators when they acquire resources from cloud providers that follow the IaaS model. By using this model, they have control at operating system level to deploy and configure applications according to their needs. This is possible thanks to virtualization, which allows the execution of individual virtual machines over shared hardware. This technology, along

with its benefits also comes with impact in the performance that should be considered [39].

This idea of virtualized environments to deploy applications has been highly used in what is called public clouds, which can be accessed for anyone who pays for the resources. Others models are private clouds, that correspond to the ones that are used in private organization, community clouds where infrastructures are shared for a group of organizations and hybrid clouds that are associated with the combination of resources from public and private clouds [21].

1.2 Automated Decision System for Efficient Resource Selection and Allocation in Inter-Clouds.

Some earlier approaches have presented solutions to support system administrators in the deploying applications in the cloud. The Automated Decision System [31] is one of such approaches. It supports system administrators in the tasks of selecting and allocating resources from IaaS public cloud providers. It is composed of two main sub-systems. The first one, the Resource Selection Decision Maker, is supported by a database that unifies the services features offering for different cloud providers, allowing comparisons among them. The goal of this system is to receive requirements from system administrators to find cloud providers configurations that can fulfil these requirements. Then it delivers the possible options ordered from the one that offers the cheapest price to the one that offers the most expensive price.

The second sub-system is the Automated Resource Allocator that is able to take the list of possible cloud provider configurations and tries to allocate the resources in the provider that offers the best price. If the best option is not available it tries the next ones, according to the provided order. One of the main features of this system is that it follows the inter-cloud model, being able to allocate resources in different cloud providers to fulfil completely the requests.

The Automated decision system is a solution that tackles the issues that administrators have to face to select and deploy resources in cloud providers. However, it assumes that system administrators know the resources that their applications need, so the estimation of resources is not considered. Moreover, the selection of cloud provider is based specifically in the best price and other important aspects related with the performance are not evaluated. Therefore further research is needed to cover these important matters.

1.3 Public Cloud Providers

In the market, it is possible to find a great number of options to choose services in public cloud providers. As example, More than 65 public clouds are registered by the monitoring service CloudHarmony [6]. Additionally, each public cloud provider offers diverse kind of services that cannot be directly compared with the services offered for other providers.

Many public cloud providers offer default configurations of virtual machines that assign provided-specific amounts of resources, such as cores of CPU, size of memory, and size of disk, to the machines that user can choose. However, the configurations differ for one provider to other. For Instance, an GoGrid [13] offers just one type of machine x-Large, that is configured with 8 cores of CPU and 8GB of memory, while Amazon [1] offer configurations m1.xlarge, m2.xlarge, m3.xlarge, and c1.xlarge, all of them with different amount of resources. Moreover, other clouds, such as CloudSigma [7], do not have default configuration but allow a flexible configuration of resources.

The name of the resources and the data center locations are other aspects that differs. A virtual machine in Amazon is called instance, in GoGrid is called cloud server and in CloudSigma server. GoGrid has their data center locations in United States while Amazon is located in five continents.

Other example of the difference among cloud providers are the billing models. For instance, spot price is a specific model that is offered just by Amazon [2] and that allows users saving money by sacrificing reliability. Using this model, users can bid for resources that will be delivered just if the bid exceeds the price of the service and similarly when the bid is less than the price, the computing service can be removed without any advice.

Table 1 shows three of the more popular cloud providers, Windows Azure [22], Amazon EC2 [1] and GoGrid [13], and illustrates some differences. For example, looking at the table, if a company needs few instances to be reserved monthly, the best option is hiring machines in GoGrid, however if the customers of the service are located in South Asia, because the performance, a better options are Windows Azure or Amazon EC2.

Moreover, if the company needs resources around the world, the best option is not to hire the machines in one cloud provider but to use several of them, following the model that has been called inter-cloud [4].

As a result, system administrators have the advantage of having many options to find what is more suitable for their needs. However, to find an answer considering these huge number of options is still a great challenge.

Table 1: Differences in the computing services offered by Windows Azure, Amazon EC2 and GoGrid.

Cloud Provider	Number of VM configurations	Geographic Availability	Billing Methods	Data transfer out per GB
Windows Azure [22]	22	-East US -West US -North Europe -West Europe -East Asia -South Asia	-Hourly -Semiannual -Annual +No price variation according to the location	First 5GB (free) 0.12 to 0.19 (amount)
Amazon EC2 [2]	1	-East US -West US -North Europe -South Asia -East Asia -Oceania -South America	-Hourly -Annual -3 Years -Discount (%) for number of reserved instances. -Spot Instances +Price variation according to the location	0.030 to 0.110 (locations)
GoGrid [14]	13	-West US -East US	-Hourly -Monthly -Semiannual -Annual +Price variation according to the location	First 1 GB (Free) 0.08 to 012 (amount)

1.4 Problem Statement

There are three issues that system administrators have to address when they decide to deploy their applications in IaaS public cloud providers:

- *Estimating the resources* that are required, considering that this estimation can vary from an external environment, such as a local data center, to a cloud environment and from one public cloud provider to other.
- *Selecting the cloud provider* that is more adequate for a given application.
- *Allocating the resources* estimated in the selected cloud providers.

This thesis proposes an architecture that supports system administrators in addressing the two initial issues (Figure 2), via the use of non-functional requirements. The third issue is addressed by the work presented by son [31], which is used as started point for the architecture presented in this thesis.

The first objective of this thesis is to use non-functional requirements to solve the initial problem of resource estimation. Non-functional requirements are used to classify

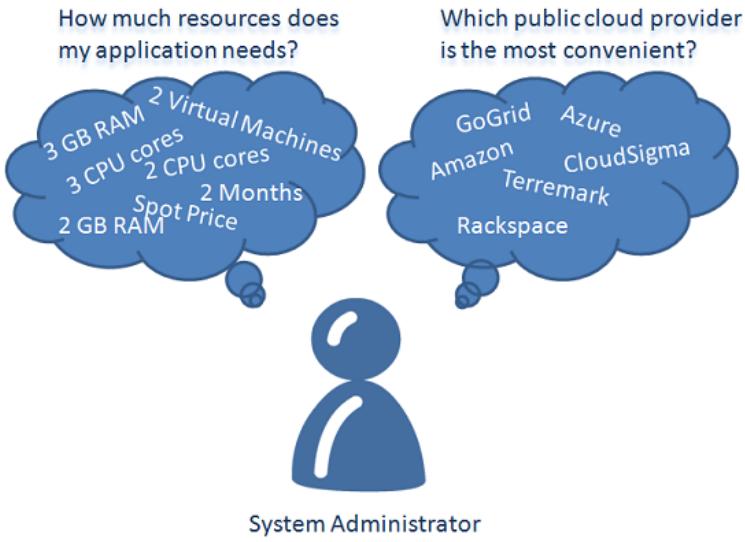


Figure 2: Issues faced by system administrators, addressed via non-functional requirements.

and to ask administrators the constraints and the performance expected from the applications. Then, considering these requirements, it is possible to perform an estimation of the resources needed, without asking system administrator for technical details.

Having the resource estimation, the second issue to address, using non-functional requirements, is the selection of the most suitable cloud provider. The work presented by Son [31] is an initial approach to help system administrators in the arduous task of selecting resources, however the prioritization of the possible configuration that fulfil the requirement is based only in the price and price is not the only aspect that should be considered. Depending of the project this is not the most important factor. As an example, in an application to sell flowers for the mother's day, availability is more important than cost because if the applications does not work the clients would buy their products in other place.

Therefore, the second objective of this thesis is to use prioritized non-functional requirements to select cloud providers. In this way, the providers can be selected, not just according with the price but according to the importance that each non-functional requirement represents for the application.

2 Related Work

2.1 Cloud Monitoring Services

The importance of knowing the performance of different public cloud providers has encouraged the development of monitoring services that report metrics to support a better picture of real behaviour of the different services. Three recognized monitoring tools are described in the following paragraphs and the main features are summarized in Table 2.

CloudHarmony [6]: This service reports updated benchmarks that endorse comparisons related with performance, network and uptime for a wide set of public cloud providers. Thus, uptime measures are available for the last 90 days and are provided for more than 120 providers, while other metrics such as web performance, disk IO performance, or multi-thread CPU performance are available for around 50 to 60 providers. To collect this metrics, monitoring services are located inside and outside of the cloud provider and additionally some benchmark applications are executed. The information can be accessed directly in the web page, in tabular or graphical format, or using SOA or RESTFUL web services.

CloudSleuth [8]: In this service, it is possible to find a tool called Cloud Provider View. This tool monitors the perceived response time and the percentage of availability of cloud providers in different cities located around the world. This information is collected by monitoring a set of hosted applications, and the measures can be found for around 90 cloud providers in time frames of 6 hours, 24 hours, 7 days or 30 days. In addition, CloudSleuth offers tools that help users to measure the performance of their applications when they are deployed inside or outside of the cloud.

CloudStatus [14]: This tool is included in the monitoring application called Hyperic and it collects, in real-time, observations of cloud provider metrics such as availability, response time, latency and throughput. Different types of applications inside and outside of the cloud are used to collect these metrics. Currently this tool has been developed for two providers: Amazon and Google.

2.2 Techniques for Application Resources Estimation

This subsection present a set of techniques based in mathematical models to estimate resources. Table 3 shows a summary of this techniques together with the model, they use.

Stewart and Shen [32] propose a model to estimate the performance of on-line ap-

Table 2: Monitoring services.

Monitoring Service	Time Frame	No. Cloud providers	Type Metrics
CloudHarmony [6]	- Last 90 % (Uptime) - Historic data Other metrics	- 120 (Availability) - 50 to 60 (Other metrics)	-Availability -CPU -Multi-Threaded -IO -Application
CloudSleuth [8]	- Last 6 - 24 hours - Last 7 or 30 days	- 90	-Availability -Response time
CloudStatus [14]	- Real Time	- 2	-Availability -Response time -Latency -Throughput

plications, which usually are composed of several layers deployed in different nodes. The general idea is to obtain a profile of each individual component, communication channels, and overhead of remote invocation and then use this profile to feed a linear model which, given the workload, estimates throughput and response time to predict the best placement strategy for components.

Shimizu et al. [30] present an approach to predict the amount of resources that an application requires to be executed. The aim of their work is to present a solution that works independently of the platform where applications are deployed. To this purpose, they propose to make a profile of the application with a specific workload in different platforms, taking measures of computing, communication, and storage parameters. Those parameters are plugged in a linear mathematical model that allows estimations of consumption of resources, and response and execution time for static workload in different conditions of hardware.

The issue of estimating resources is also developed by Wood et al. [36]. They address this issue considering the applications that are deployed in virtualized environments. The authors propose to perform the estimation in two steps. The first one is to execute a profile of several benchmarks in virtualized and no-virtualized environments in a given platform. In the second step, these results are used in a regression model to calculate the resource usage of any application deployed over this platform.

Another interesting work focuses in estimating application performance in virtualized environments. The authors, Kundu et al. [16], propose to build a model that is iteratively trained to achieve a level of accuracy. The model is fed with the percentage of CPU allocation, the memory allocation, disk and network IO, and with performance metrics, such as throughput or response time obtained with the resources configuration given.

They suggest using this metrics in a Neural Network model that, according to the evaluation could work better than linear model to estimate resources in virtualized environments.

Table 3: Resource estimation techniques.

Author	Proposal's Main Goal	Type of Model
Stewart and Shen [32]	Estimate resources of a Multi-tier application	Linear model
Shimizu et al. [30]	Estimate resources for an application that is migrated of hardware platform.	Linear model
Wood et al. [36]	Estimate resources of any application in a virtualized environment.	Linear regression model
Kundu et al. [16]	Create an iterative model that incrementally improve the accuracy of the Model.	Neural networks model

2.3 Tools to Estimate and Compare Resources in Cloud Providers

In the literature, it is possible to find some approaches related with the tasks of estimating resources and selecting the best cloud provider for a given application; however, according with the knowledge acquired, none of them suggest a complete system able to translate non-functional requirements in a final allocation of resources in cloud providers. In this section, known approaches, which deal with similar goals to the one pursuit by this thesis, are described. Table 4 presents a summary of these approaches.

The initial work to present is DEVA [33], which is a module created to be added in a cloud data center toolkit manager such as Open Nebula [26], Eucalyptus [10] or any internal resource manager. This module has the ability to translate user resource requirements in cloud physical resources that satisfy, mainly, network QoS requirements given by the user. To achieve this goals, the authors propose to add a resource manager in the cluster manager and an agent in each physical host that composes the data center. This two components work together to monitor and to setup network connections in order to keep the levels of QoS.

In the initial work of DEVA, specific requirements such as CPU power, quantity of RAM, and bandwidth are expected to be given by the user; however, in a second work [34], the proposal is to add a module able to receive non-functional requirements that later are translated in the specific requirements. These non-functional requirements are also used for application monitoring and to perform changes in the cloud deployment to

keep the fulfilment of these non-functional requirements. It is important to notice that this application focuses in improving the service delivered from cloud providers, so it is a provider decision to implement this kind of solution. Therefore, it is not an option for end-users that want to choose a public cloud provider.

Another tool designed to estimate applications resources and performance in the cloud is CloudProphet [20, 19]. It performs the estimation in two phases. The first phase is called tracing and it is used to collect data related with the application workload in an environment outside of the cloud provider. This workload is collected considering multi-threaded applications, so time spent in synchronization is excluded to consider just the real execution time of the threads. In a second phase, called replaying, an agent is set up with the application workload to simulate the execution of the application in the cloud provider and to estimate the cloud performance. The main advantage that the authors present is that the application does not have to run in all providers to know their performance, which could be more secure option.

In an initial work, Li et al. [17] presented CloudCmp, which allows the comparison of different cloud providers by using a tool to perform systematic benchmarking. This tool evaluates mainly the services of elastic computing, persistent storage, intra cloud and wide area networking and to do this, for each kind of service, different metrics are chosen and measured by performing the same tasks in different cloud providers.

In a second work by Li et al. [18], the benchmarking tool of CloudCmp is included in a more ambitious application that allows comparisons of cost and performance of a web application when this is deployed in different cloud providers. To achieve this task, the proposal is first to perform the benchmarking and then to profile the application workload by measuring the different trace requests in order to find the most expensive path. Finally, the result of the workload is combined with the benchmarking values to estimate which cloud provider could offer the best result.

An additional approach is described by Rak et al. [29]. The authors propose an evaluation by using simulation that consists in first executing a benchmarking according with the application features in the cloud provider where an application will be deployed. Then with the results of this benchmarking, the parameters of a simulation model that allows estimating of resources, cost and response time that the application could consume for given workload are set up. Finally, an additional step evaluates different configurations considering the prices given for the cloud provider helping users to visualise which

performance features could be sacrifice to get a better price. The model was developed considering specific characteristic of applications developed in mOSAIC [23], which is an environment for the development of cloud applications. It is important to notice that, in futures works, the authors propose to use non-functional requirements to create a system that suggests the best option among a set of cloud providers.

An approach that differs from the ones presented before, because does not use any computational profile of the application, is SMICloud [11]. This is a software framework to rank cloud providers for a given application considering the Service Measure Indexes : accountability, agility, assurance of services, cost, performance, security and privacy and usability. The proposal is assigning different Key Performance Indicators (KPI) to evaluate these indexes in different cloud providers. Them these measures can be used to compare, rank, and suggest a cloud provider. The framework is compose of a SMICloud broker where the user can specify the QoS attributes that his application needs, together with a weight that means the importance of each attribute. Additionally, the framework has a SLA manager that keeps the historical compliance of the SLA promised by the providers; a monitoring service that is in charge of keeping information about cloud providers' functionality and a Service Catalogue that keeps the features offered by cloud providers. Having this information, the module SMI calculator gives a value to each one of the KPI and a final module called the Ranking system use this information to resolve the Multiple Criteria Decision Making (MCMD) using an Analytic Hierarchy Process (AHP).

The final approach presented by Zhang et al. [38], supports system administrators in the difficult task of reading cloud providers specifications to find the parameters to compare and finally determined the best option at the best price. The authors propose a tool that, after being populated with information from different cloud providers, is able to unify concepts and measure units. Having this information unified enables the creation of an interface that, given the application resource requirements, calculates the different cloud provider that can fulfil the requirements and estimate cost. The authors suggest modifying the decision making algorithm to include user requirements. To achieve this goal they propose to use a combination of genetic algorithms with Analytic Hierarchy Process (AHP) [37].

Table 4: Tools to estimate and compare resources in cloud providers.

Tools	Focus Applications	Main goal	Techniques
DEVA [33] [34]	Any	Monitoring QoS applications inside a cloud provider	-Application profile -NF-Requirements
CloudProphet [19] [20]	Web	Estimate Resources	-Application profile
CloudCmp [17] [18]	Web	Compare providers	-Application profile -Benchmarks
Evaluation by Using Simulation [29]	mOSAIC	Estimate resources and best configuration	-Application profile -Benchmarks
SMICloud [11]	Any	Compare Providers	-NF-Requirements -Benchmarks
Declarative Recommender System [38]	Any	Compare Providers	-Resources Required -Database with cloud providers specifications

3 System Design

In this section we present an overall view of the proposed architecture for provisioning resources in the cloud via non-functional requirements. The architecture is composed of three main modules: the Cloud Resources Estimator, the Resource Selection Decision Maker, and the Automated Resource Allocator.

The Cloud Resources Estimator is a module able to estimate resources that an application could consume when it is deployed in a cloud provider. It receives non-functional requirements, such as efficiency, availability and reliability of a specific application and translates them in an estimation of resources, such as number of CPU cores, Megabytes of memory, and Gigabytes of Disk.

The second module, called Resource Selection Decision Maker, is responsible for suggesting the cloud providers that are more adequate to deploy an application. For this purpose, it receives the resource estimation, the constraints given by non-functional requirements, and the prioritization of these non-functional requirements.

Finally, the Automated Resource Allocator is a module able to allocate resources directly in cloud providers. For this task, it uses the output given by the Resource Selection Decision Maker, trying to allocate first the most convenient providers. Figure 3 shows the general view of the system.

As presented in the section 1.3, the two last modules have been already proposed by Son [31]. However, the Resource Selection Decision Maker architecture has been enhanced, including non-functional requirements in the decision process. The present work is focused in the design and implementation of the first module the Cloud Resources Estimator and in the improvements of the second module, the Resource Selection Decision Maker.

3.1 Non Functional Requirements

We propose a Cloud Resources Estimator and a Resource Selection Decision Maker that work based on non-functional requirements. To select the initial set, a list containing common non-functional requirements [5] was used, and those that were related with the deployment of applications and those that allow cloud providers comparisons were chosen. Therefore, requirements such as usability or testability were not considered because they are more related with the development of applications itself.

The details of the non-functional requirements selected and the initial features that

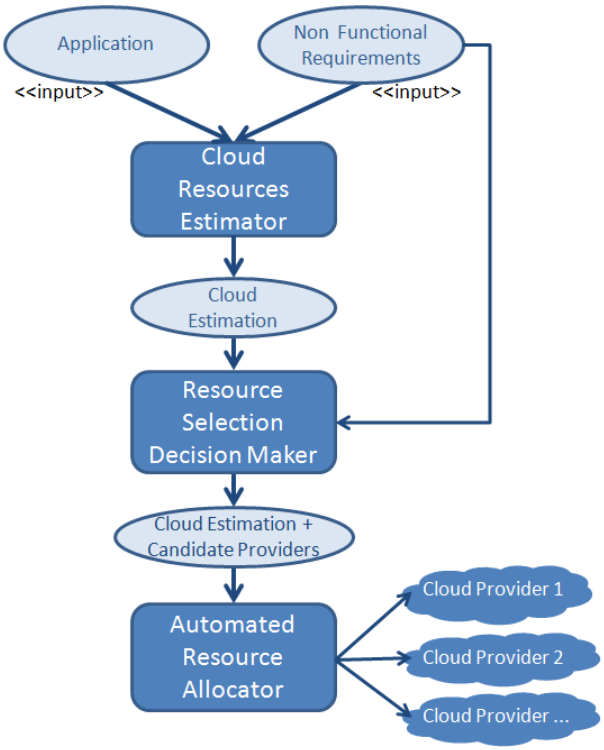


Figure 3: Software system provisioning via non-functional requirements.

should be provided by system administrators, in order of describing the level needed by the requirement, are described below. A summary of these requirements and the possible features is presented in Table 5.

Table 5: Initial non-functional requirements to consider in the system.

Non Functional Requirements	Features ask to the users
Portability	OS Images OS Hardware 64 or 32 bits
Reliability	Mean Time To Repair (MMTF) Mean Time To Fail (MMTR) Requires resources for application backup Type of environment Allows spot price
Availability	Uptime percentage Users locations Requires load balance Contract time
Efficiency	Throughput Response time Workload
Cost	Maximum price to pay for a window of time

Portability: This requirement is related with the easiness of an application to be

executed in different platforms. Therefore, related features are: operating system of the images installed in the virtual machines, operating system required in hardware and if it is preferred to run on platforms of 32 or 64 bits.

Reliability: In the context of this work, we target reliability related with the hardware offering by cloud providers. Therefore, some possible features that can be asked to administrators are the Mean Time To Failure (MTTF) and the Mean Time To Repair (MMTR). Moreover, reliability can be also related with resources required to run an application backup; with the number of machines to run a multilayer application, and with the type of environment. The last one means whether resources are required for test, development, or production environment. In addition, it can be included if it is acceptable to use the spot price [2], supported by Amazon.

Availability: This is related with the ability of the system to respond to user requests. Therefore, it is important to ask system administrators the minimum percentage of uptime required from cloud providers, the different locations where the application will be accessed or if the application supports load balance to estimate resources for this functionality. Other aspect to be considered is if there is a defined time contract or if the service is continuing over the time.

Efficiency: The efficiency is associated with the application performance. Therefore, it is necessary to ask administrators values of response time, throughput, and workload expected for the application.

Cost: This requirement is related with how much the customer is willing to pay in order to be provided with a service that supports all the additional non-functional requirements. It is possible to ask the price expected to pay for a window of time.

The requirements mentioned above are probably sufficient to make a satisfactory estimation of cloud resources and to choose an adequate cloud provider. However, these requirements could be not enough for some applications and for other some of this could be irrelevant. Therefore, it is important to allow system administrators to add and remove no functional requirements in a dynamic way.

Apart from the dynamic management of non-functional requirements, other aspect to consider is the way that they should be evaluated. The idea is to use a workflow to evaluate requirement by requirement according to a priority given by system administrators. Understanding, that non-functional requirements are generally contradictories, so if a user wants to achieve one level in one requirement, probably he/she has to sacrifice other

requirements. Thus, the more important requirements are guarantee even though others that have less priority could not be satisfied. Figure 4 represents the workflow to evaluate non-functional requirements.

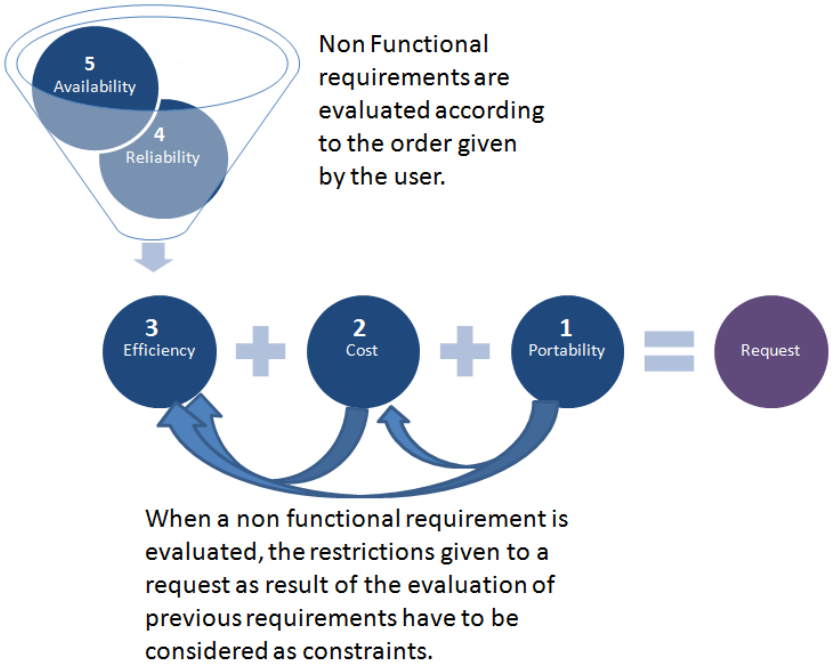


Figure 4: Workflow to evaluate non-functional requirements.

3.2 Cloud Resource Estimator

The architecture proposed for the Cloud Resources Estimator follows a client-server model of three tiers composes of database, business, and presentation tier. The next sections explain the functionality and design of each tier.

3.2.1 Database Tier

In the database tier, two different entity relation models are proposed and implemented. The first one keeps non-functional requirements, their features, and the values that these features can assume. The second one, which is used to estimate resources, stores data with the results of application profiles executed in different infrastructures.

The first entity relation model, related to non-functional requirements, is presented in Figure 5 together with an example of how it is populated in Figure 6. Next, the main entities are described.

Est_nf_requirement: This relation enables storage of as many non-functional requirements as needed. Each new requirement should specify the name of the class that will

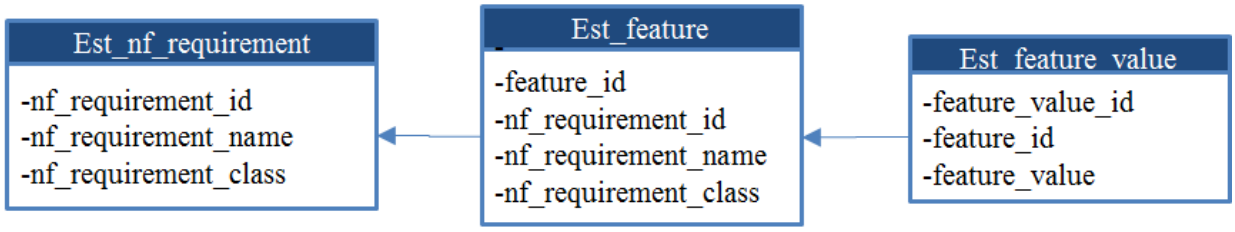


Figure 5: Entity relation model of non-functional requirements.

est_nf_requirement

NF_Requirement_ID	NF_Requirement_Name	NF_Requirement_Class
1	Portability	NFEvaluatorPortability
2	Reliability	NFEvaluatorReliability
3	Availability	NFEvaluatorAvailability

est_feature

Feature_ID	NF_Requirement_ID	Feature_Name	Feature_Type
1	1	OS Image (Name - Version)	Close
2	1	OS Image bits	Close
3	1	Os Hardware	Close
4	1	Required single provider	Close
5	2	Type of environment	Close
6	2	Application layers	Close
7	2	Deploying in different data center locations	Close
8	3	Percentage uptime	Close
9	3	Support load balance	Close
10	3	Regions of access	Close
11	3	Contract period	Open
12	3	Total hours	Open

est_feature_value

Feature_Value_ID	Feature_ID	Feature_Value
1	1	Ubuntu 12.04
2	1	CentOS 6.4
3	1	Debian 6
4	2	32 bits
5	2	64 bits
6	3	Linux
7	3	Windows
8	4	Si
9	4	No
19	5	Development
20	5	Production
21	5	Test User
22	5	Test Load
15	7	Si
16	7	No
17	9	Si
18	9	No

Figure 6: Example of the data stored.

evaluate it in the business logic, thus this new requirement can be added dynamically.

Est_feature: This relation is created to specify the features associated to each requirement. A field to notice in this relation is *feature_type*, because this identify whether a feature receives close values, such Yes or No, or accepts open values, such as number of days of a contract or response time.

Est_feature_value: This relation contains the values that a user can chose for features of type close. For example, in Figure 6, the rows with *Feature_ID* 2 (OS image bits) contains the two possible answers, stored in the *Feature_Value_ID* 4 and 5 with the *Feature_value* 32 bits and 64 bits respectively.

A second entity relation model, showed in Figure 7, represents the data related with the application profile and the performance estimation in different infrastructures. This was designed considering the work presented by Stewart and Shen [32] that proposes prediction of throughput of an application based in the performance of individual components and the cost of communication among them. This approach has the advantage of allowing

the estimation of the application performance with components deployed in different infrastructures and possibly in different cloud providers. A description of entities suggested to storage this information is provided below.

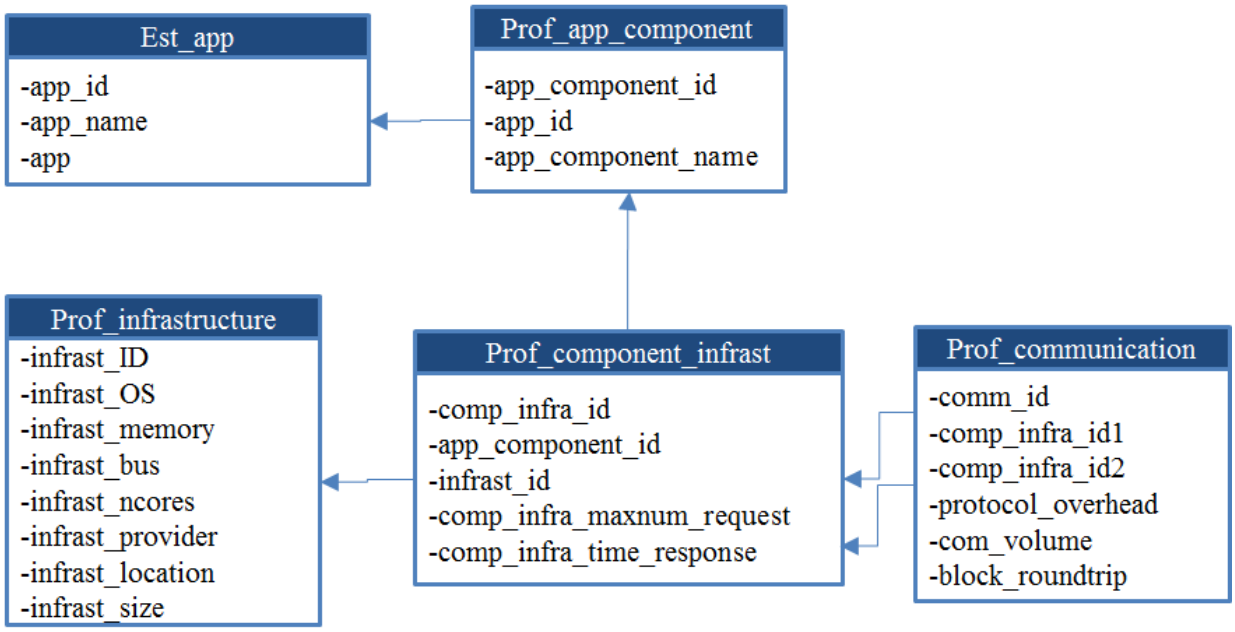


Figure 7: Entity relation model of application profile.

Est_app: This relation represents the application, which is composed of one or more components.

Prof_app_component: This relation stores the different components that constitutes an application.

Prof_infrastructure: This relation represents the characteristic of different infrastructures where the profiles are executed.

Prof_component_infrast: From this relation it is possible to obtain the performance of each component in each infrastructure.

Prof_communication: This relation stores the cost and overhead of communication between a component *c1* installed in a infrastructure *i1* and a component *c2* installed in a infrastructure *i2*.

3.2.2 Business Tier

The business tier of the Cloud Resource Estimator encapsulates the functionality to recovery data from the database and the logic to create a workflow to evaluate non-functional requirements and to perform the estimation of resources needed from cloud providers. In

order to present the design, a general description of the classes that compose the class diagram (Figure 8) is given here.

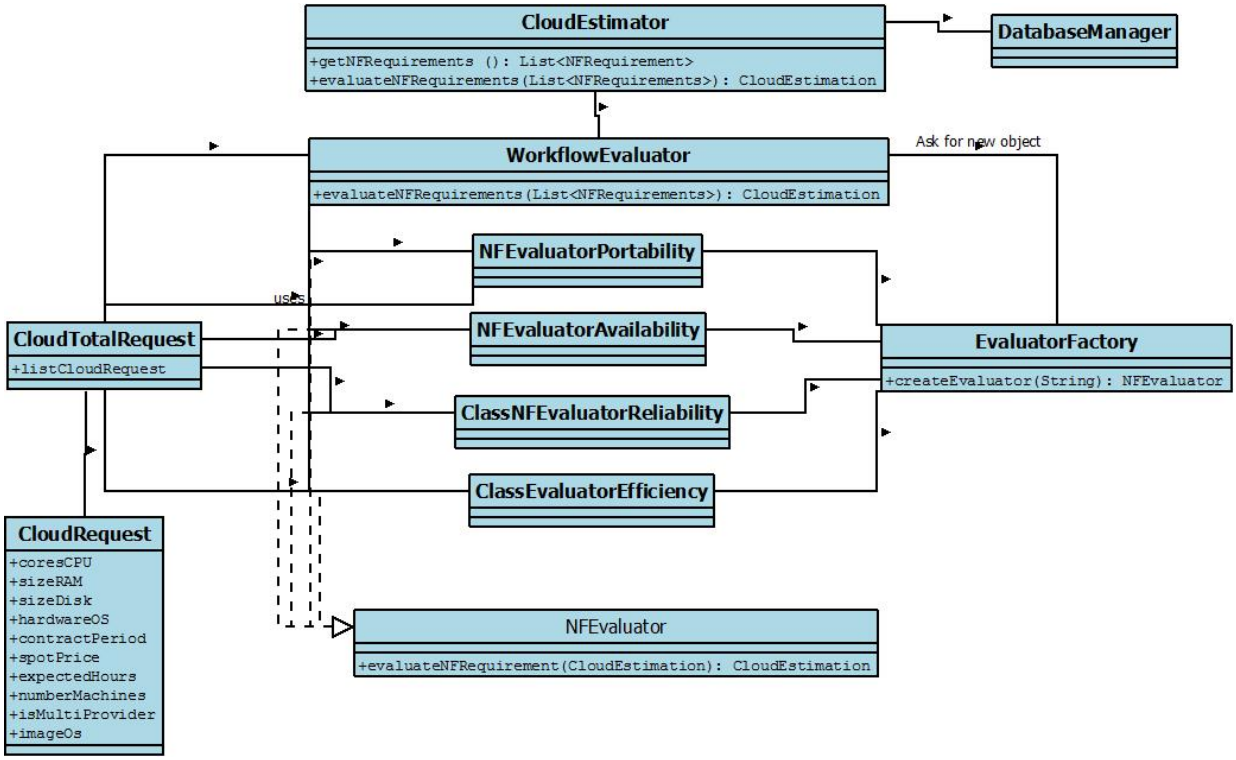


Figure 8: Class diagram business tier.

CloudEstimator: This class was designed following the Facade pattern [35], so this is the only interface to access the functionality of the Resource Estimator. It has two methods *getNFRequirements*, to obtain the values from the database, and *evaluateNFRequirements*, which begins the workflow to evaluate non-functional requirements.

WorkflowEvaluator: This class manages the workflow. Therefore, its process consists in obtaining each requirement according with its priority, from a list given by the system administrator, to then evaluate it. For each non-functional requirement it uses the associated evaluator class (this association is stored in the database in the relation *est_nf_requirement* shown in Figure 5).

CloudRequest: This class represents a cloud provider request. This request can include one or several configurations if they have the same specifications. However, if different configurations are needed, different instance of *CloudRequest* are created. A *CloudRequest* is created with default field values that are changed for the evaluators considering the features provider by the user.

CloudTotalRequest: This class keeps all the different requests, *CloudRequest*, that can result after running the evaluators of each non-functional requirement.

NFEvaluator: This is the abstract class that is inherited for each one of the evaluators that are created to assess each non-functional requirement. The only method that it implements is *evaluateNFRequirement*, which receives an object *CloudTotalRequest* and returns a *CloudTotalRequest* modified for the workflow of non-functional requirements.

EvaluatorFactory: This class is used by *WorkflowEvaluator* to obtain the implementation of each evaluator. This class follows the Factory pattern [35], thus it receives the name of the evaluator and according with this creates and returns a implementation of a class that inherits from the abstract class *NFEvaluator*.

The classes that were not described, *NFEvaluatorPortability*, *NFEvaluatorAvailability*, *NFEvaluatorReliability*,*NFEvaluatorEfficiency* and *NFEvaluatorCost*, correspond to evaluators, that assess each non-functional requirement and that inherits from the class *NFEvaluator*. In this document, we describe, as an example, the functionality of the evaluator *NFEvaluatorEfficiency*.

NFEvaluatorEfficiency: To evaluate efficiency and estimate resources, we propose, in an initial step, to use a profile [33, 19, 17, 30] that runs the application components with a given workload. The results of the execution, which are resources consumed and performance achieved, can be store in the entity relation model presented in Figure 7. The idea is to profile components in different infrastructures [30] to have better estimations.

Having the profile, the second step is to use this information to feed an estimator model, for example a neural network as suggested by Kundu et.al [16]. Then, in a third step this model can be used to estimate resources, that an application could need when it is deployed with different workload and when it needs to achieve a different performance. After this estimation, the new data obtained, using the model, should be also stored in the database and possibly updated with the real values that could result of running the application in the cloud provider. This new data is stored to update the model and to make it more accurate. After, profiling, modeling the application and estimating resources, the output is generated, updating the existent *CloudTotalRequest* with the result of the estimation. Figure 9 shows the steps executed.

3.2.3 Presentation Tier

The presentation tier corresponds to the graphic interface that is used by system administrators in order to access the application functionality. An important design requirement of this tier is to provide a friendly interface to allow users adding and removing non-functional

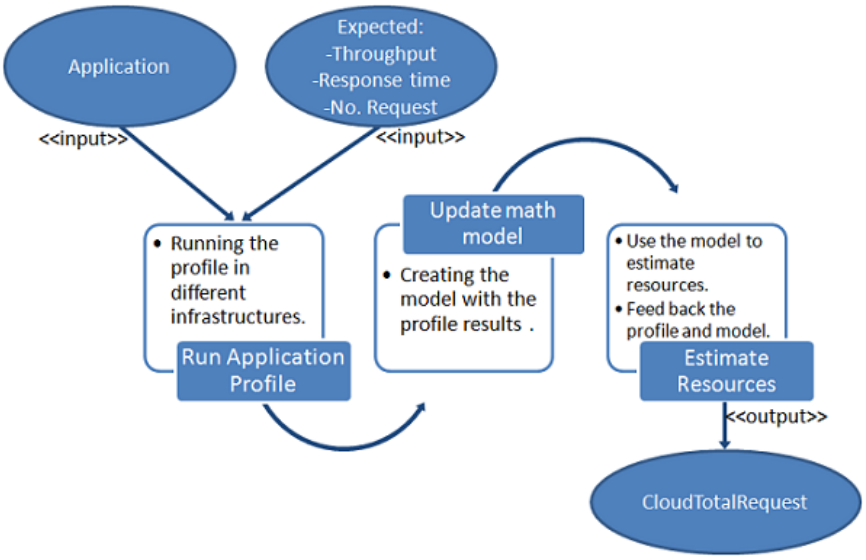


Figure 9: Steps executed by the evaluator *NFEvaluatorEfficiency*.

requirements, and supply the information of the features that describe them. It also should support the entry of the non-functional priority that is given by order.

3.3 Resource Selection Decision Maker

The Resource Selection Decision Maker receives cloud resources requirements to suggest a set of possible configurations prioritized by price. Non-functional requirements features can include new constraints that along with the priority given by system administrators to non-functional requirements, should be considered to select the most adequate providers. Our architecture of Resource Selection Decision Maker is based on the model proposed by Son [31], but it has been substantially improved.

First we propose to extend the data base schema adding new relations to store the values that cloud providers have for new features (see Figure 11). For example, a relation *Uptime* could be added to store the information related with the availability of the different cloud providers, in order to include a new constraint of minimum percentage of uptime to filter the candidate providers. Other examples of additional constraints are shown In Figure 10.

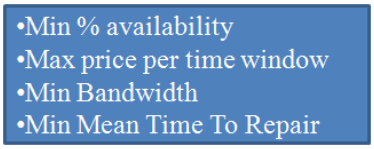


Figure 10: Example of constraints to filter the candidate cloud providers.

The output of the Resource Selection Decision Maker is a set of possible configurations organized by the best price. We add a new functionality that receives this configurations and order them according with the priority given to non-functional requirements. This is achieved with a relation *NFR_Evaluation* (see Figure 11). It keeps the assessment that the different providers receive for each non-functional requirement [11]. This data could be maintained by using the information from cloud monitoring services and by collecting the experience of users when they deploy their applications in different cloud providers.

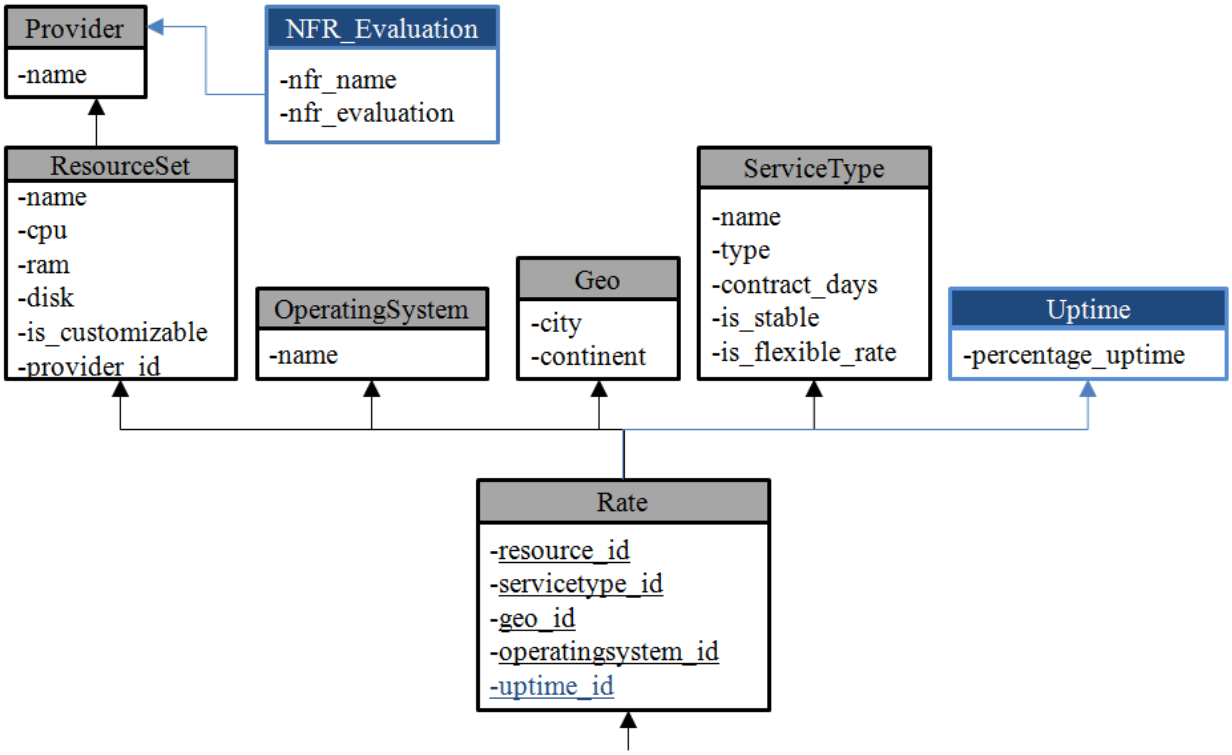


Figure 11: Extension of the database schema of the Resource Selection Decision Maker.

The proposed architecture allows system administrators to allocate resources in the cloud without providing low-level technical details, but providing a high level knowledge, represented with prioritized non-functional requirements. They are used in the components responsible for estimating resources and selecting cloud providers. The overall architecture shows that to provide an automated system it is necessary the integration of inputs and outputs of the different components. To facilitate the integration of the Cloud Resource Estimator, the architecture was designed using a multitier model. Therefore, the business component can be invoked from a web base interface or for an external application that needs to use its services. The next section details a prototype implementation of the proposed architecture.

4 Implementation

In order to illustrate and evaluate the functionality of the proposed architecture, we implement a web base prototype that includes the Cloud Resource Estimator and the functionality to prioritize candidate providers by non-functional requirements included in the Resource Selection Decision Maker. The present section provides the details of this implementation.

4.1 Technical Details

The prototype was developed using the Java Platform Enterprise Edition 7 (Java EE 7) [27], which is the last version of a middleware software platform that gives complete support to develop multitier architectures, such as the one proposed in the design of the Cloud Resource Estimator. To deploy the prototype was used the application web server for Java EE GlassFish 4.0 [12].

The database selected was My SQL 5.5 [24] and to access the database following the JEE standard, the Java Persistence API included in Java EE 7, was used. This API allows Object-Relation Mapping. It means that the database relations are mapped to Java objects, making simpler to deal with database instances. The presentation tier was developed using Java Server Faces 2.0 [28] using components from JBoss RichFaces 4.3.3 [15].

The use of this technologies allow us to implement an architecture of three tier where presentation and business components are well defined and independent developed, so it is possible to build a web interface or other type of interfaces without performing any changes in the business and database tier.

4.2 Cloud Resource Estimator

The prototype of the Cloud Resource Estimator includes the implementation of the two entity relation models presented in the section 3.3.3, that were created for keeping non-functional requirements data (see Figure 5) and for storing the profile information (see Figure 7). The implementation of the Resource Estimator assumes that the profile of the applications was performed off-line. Therefore, this information is available when the workflow of non-functional requirements is executed.

The Resource Estimator was configured with four non-functional requirements: porta-

bility, availability, reliability and efficiency. They were configured with the features necessary to create a resource estimation request able to supply the information required for the existent Selection Decision Maker.

One interesting feature of the implemented availability requirement is that it is able to add more than one location, so a cloud request estimation is created for each selected place. Moreover, the prototype is also able to work with application components. Therefore, It is required to store the results of the profile in n components, if the requirement is running the application in n virtual machines. Having this information, the prototype is able to generate individual estimations of the resources required for run each component in different virtual machines.

To facilitate the user tasks of adding, removing, and prioritizing requirements by order, a pick list, as is shown in Figure 12, was used. The order given by the user is stored in the relation *App_nf_requirement*, which have the structure shown in Figure 13. This relation was used to consider non-functional requirements in the prioritization of cloud providers explained in the next subsection.

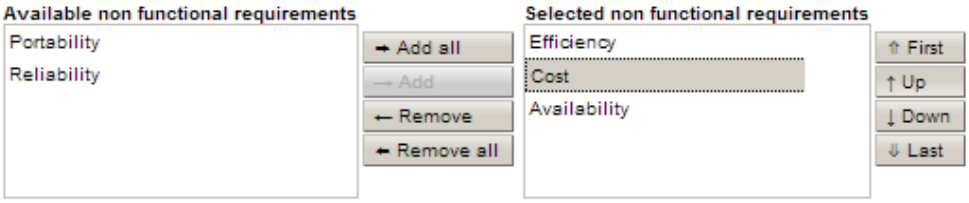


Figure 12: Picklist for adding, removing and ordering non-functional requirements.

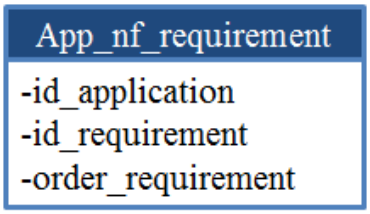


Figure 13: Relation App_nf_requirement.

Figure 14 shows an example of the screen used for user to supply the features and the priority by order of the non-functional requirements and Figure 15 shows the output screen provided by the prototype.

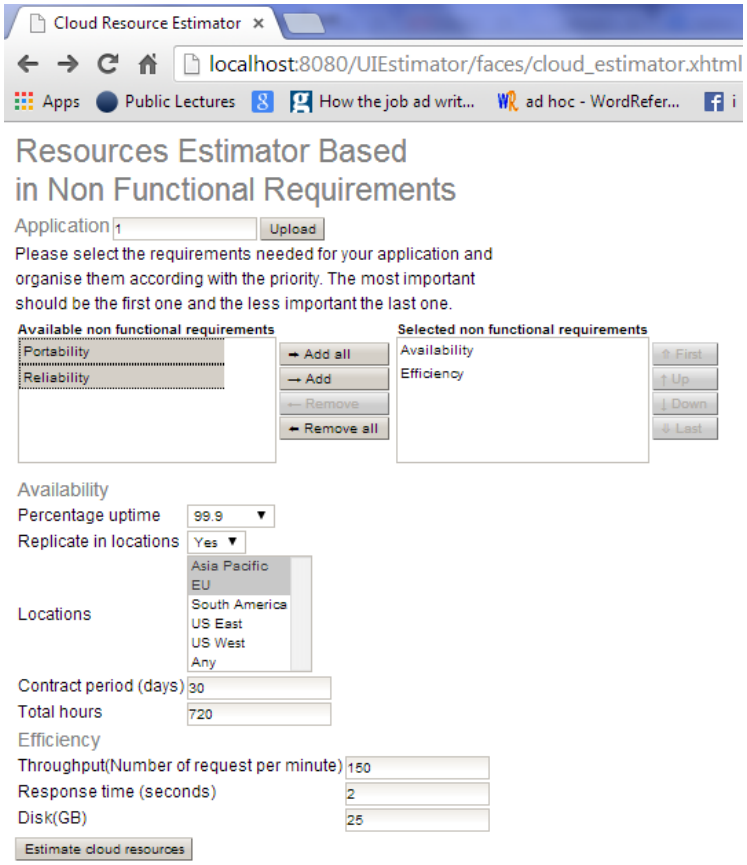
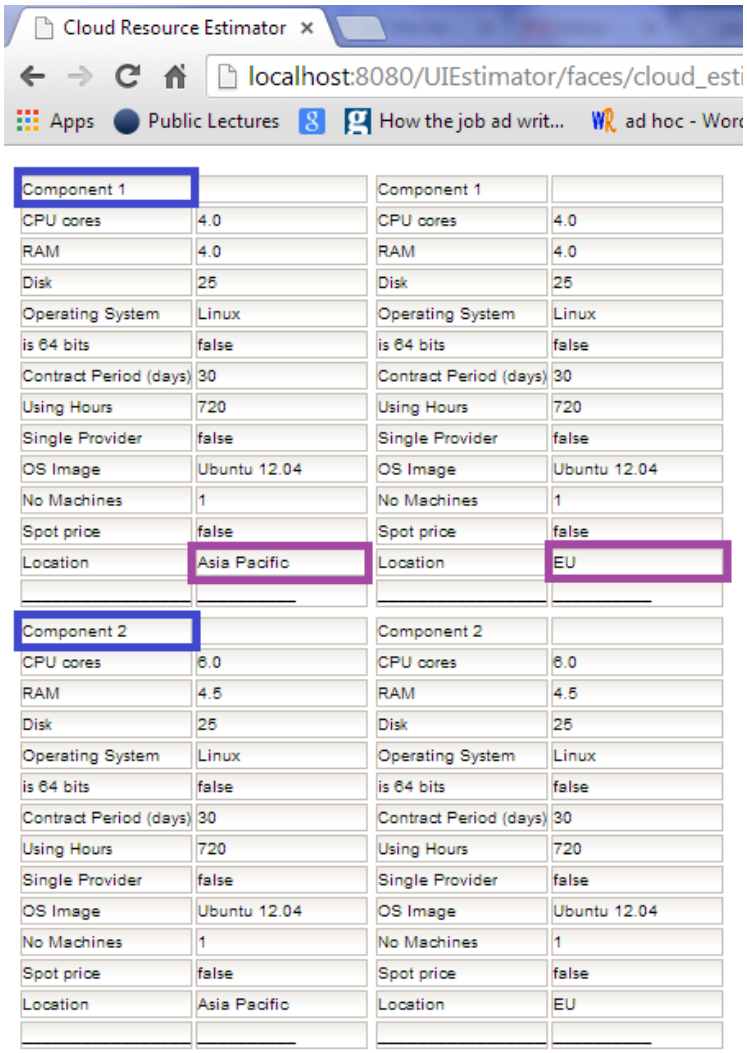


Figure 14: Resource Estimation Request.

4.3 Candidate Providers Prioritization

The functionality to prioritize candidate providers by non-functional requirements was implemented including two relations of the design proposed to improve the Resource Selection Decision Maker: *Provider* and *NFR_Evaluation* (see Figure 16). The information contained in these two relations along with the order of the non-functional requirement stored in the relation *App_nf_requirement* (see Figure 13) are used to reorganize the output list of candidate cloud providers generated for the existent Resource Selection Decision Maker. After organized, the list conserves the same format of the input, so it can be used as the input of the next component of the architecture which is the automated resource allocator. Figure 17 shows the screen of this functionality.



Component 1		Component 1	
CPU cores	4.0	CPU cores	4.0
RAM	4.0	RAM	4.0
Disk	25	Disk	25
Operating System	Linux	Operating System	Linux
is 64 bits	false	is 64 bits	false
Contract Period (days)	30	Contract Period (days)	30
Using Hours	720	Using Hours	720
Single Provider	false	Single Provider	false
OS Image	Ubuntu 12.04	OS Image	Ubuntu 12.04
No Machines	1	No Machines	1
Spot price	false	Spot price	false
Location	Asia Pacific	Location	EU
Component 2		Component 2	
CPU cores	6.0	CPU cores	6.0
RAM	4.5	RAM	4.5
Disk	25	Disk	25
Operating System	Linux	Operating System	Linux
is 64 bits	false	is 64 bits	false
Contract Period (days)	30	Contract Period (days)	30
Using Hours	720	Using Hours	720
Single Provider	false	Single Provider	false
OS Image	Ubuntu 12.04	OS Image	Ubuntu 12.04
No Machines	1	No Machines	1
Spot price	false	Spot price	false
Location	Asia Pacific	Location	EU

Figure 15: Resource Estimation Result.

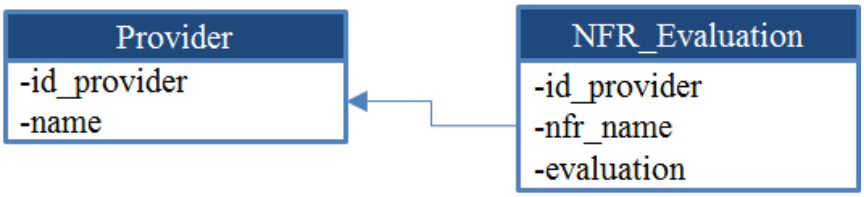


Figure 16: Relations Provider and NFR_Evaluation.

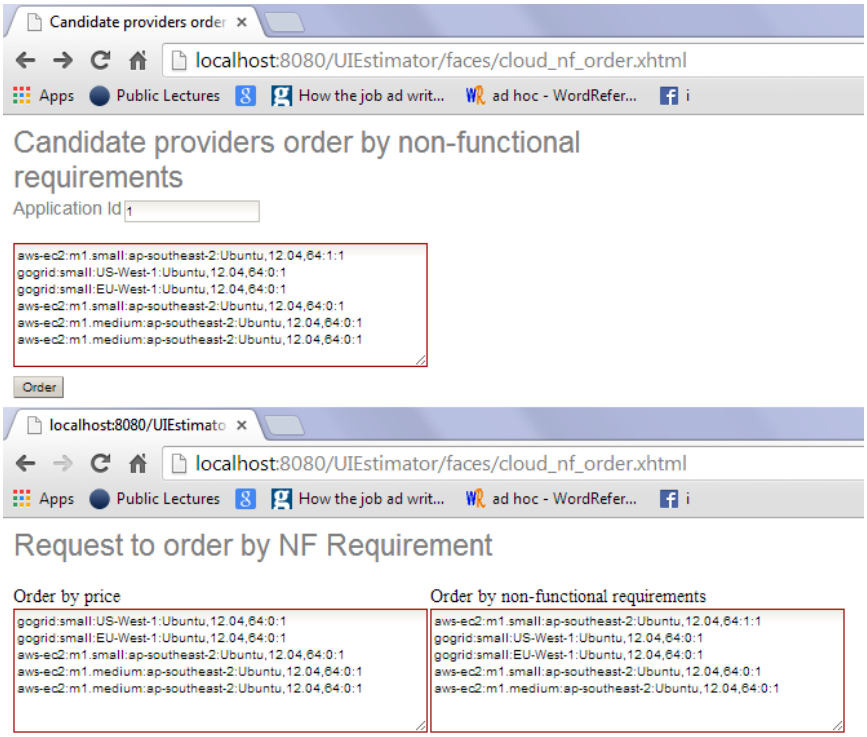


Figure 17: Candidate providers order by non-functional requirements.

5 Evaluation

In this section, we present the experiments carried out to evaluate the proposed architecture. The Resource Estimator was evaluated in order to have its performance measure. The Selection Decision Maker was assessed, in order to evaluate the benefits of including non-functional requirements in the prioritization of cloud provider candidates.

5.1 Cloud Resource Estimator

We evaluate the Cloud Resource Estimator by showing that the proposed architecture is able to scale dynamically when the number of requirements and requirement's features increase. For this purpose, it was created and evaluator class, which inherits from *NFEvaluator*. To enable the evaluation, this evaluator modifies 12 properties of the class *CloudRequest*. This class was associated in the relation *est_nf_requirement* as the dynamic class that evaluated the non-functional requirements.

The platform used to test the application was the Australia Research Cloud NeCTAR [25] and for the experiments, was utilized an instance of type m1.small. Instances of this type have 4GB of RAM, 1 CPU core and 10 GB of Disk. The Operating System used in the virtual instance was Ubuntu 13.04.

To increase the number of requirements, the relation *est_nf_requirement* was populated with different workloads, beginning with 10 requirements and increasing the number by 10 until reaching 300. In the same way, the number of features in the relation *est_feature* were incremented. For each increment of requirements and requirement's features, the time spending in performing the non-functional requirements evaluation was measured. The results of the experiment are shown in Figure 18

The conclusion of this experiment is that the application prototype scales satisfactorily when the number of requirements increase but the performance is more affected when the number of features grows. However, moving to a real environment it is very unlikely that the number of features achieve a number higher than 70, which is the number were the performance begin to decrease. Also, looking to the extreme case the time consume to process 300 requirements with 300 features does not take more than 4 seconds, which is an acceptable time to process this amount of information.

Another observation in relation to the graph is that the modification of the number of requirements and features were performed in objects in memory that map the database

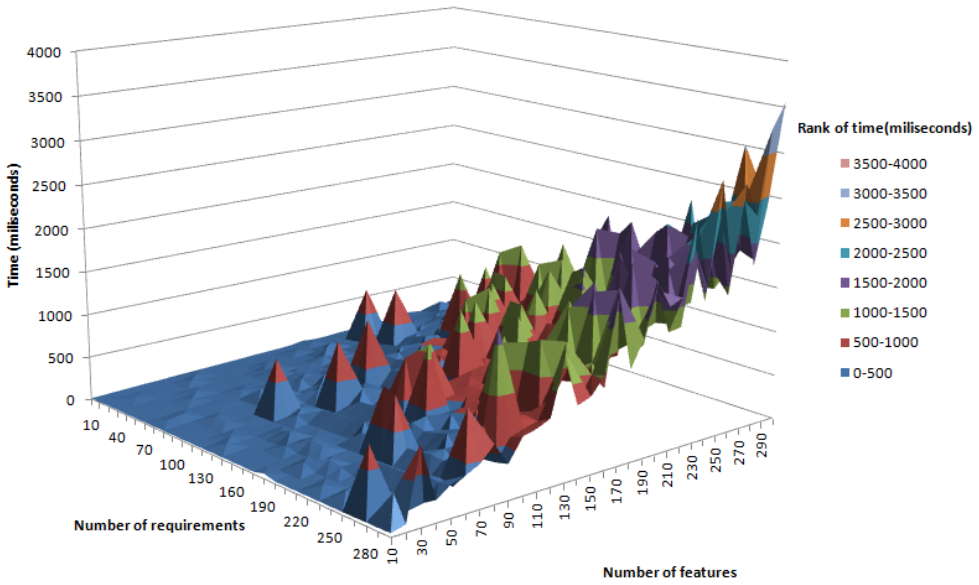


Figure 18: Set of candidate providers to be priorized.

relations, therefore allocation and deallocation of objects increase the work of the Java garbage collector. In the real environment objects will be loaded just one time in memory without modifications, so it is expected to have better performance.

5.2 Resource Selection Decision Maker

Experiments performed for the Resource Selection Decision Maker have the main objective of assessing the benefits of including non-functional requirements in the Selection Decision Maker.

First, the functionality of the system was tested with synthetic information, to show the correct process of the prototype when it organize candidate cloud providers by non-functional requirements. The relations *Provider*, *NFR_evaluation* were populated with the information given in the Table 6 and the non-functional requirements of a given application were prioritized according to the order shown in Table 7. The input of initial set of candidate cloud providers prioritized by price and the output of this set prioritized by non-functional requirements are shown in Figure 19.

The new prioritization of the candidates providers shows how the *provider2:location1* is considered the best option, because it has bigger evaluation in relation to portability than the *provider 1:location1*, which by price was the best option. Moreover, *provider2:location1* has same portability evaluation than the *provider2:location2*; however, thanks to the consideration of the next requirements, it is possible to find that *provider2:location1* is the

Table 6: Synthetic evaluation of non-functional requirements.

name_provider	nfr_name	evaluation
provider1:location1	portability	6
provider1:location1	availabiity	9
provider1:location1	reliability	9
provider1:location1	efficiency	8
provider2:location1	portability	7
provider2:location1	availabiity	8
provider2:location1	reliability	8
provider2:location1	efficiency	9
provider2:location2	portability	7
provider2:location2	availabiity	6
provider2:location2	reliability	6
provider2:location2	efficiency	6

Table 7: Non-functional requirements prioritized by order.

application	nfr_name	order
1	portability	4
1	availabiity	2
1	reliability	3
1	efficiency	1

Data-Input

```
provider1:m1.small:location1:Ubuntu,12.04,64:1:1
provider2:small:location1:Ubuntu,12.04,64:0:1
provider2:small:location2:Ubuntu,12.04,64:0:1
provider1:m1.small:location1:Ubuntu,12.04,64:0:1
provider1:m1.medium:location1:Ubuntu,12.04,64:0:1
provider1:m1.medium:location1:Ubuntu,12.04,64:0:1
```

Data-Output

```
provider2:small:location1:Ubuntu,12.04,64:0:1
provider2:small:location2:Ubuntu,12.04,64:0:1
provider1:m1.small:location1:Ubuntu,12.04,64:1:1
provider1:m1.small:location1:Ubuntu,12.04,64:0:1
provider1:m1.medium:location1:Ubuntu,12.04,64:0:1
provider1:m1.medium:location1:Ubuntu,12.04,64:0:1
```

Figure 19: Set of candidate provider order by non-functional requirements.

best configuration. After perform this experiment we conclude that more non-functional requirements mean more refined prioritization of the set of candidate providers. Therefore, even if price is considered the best option, adding more requirements allows the system to make better decisions.

A second experiment for assessing the benefits of non-functional requirements in the selection of the cloud providers uses information of monitoring service for evaluating cloud provider using non-functional requirements. Availability was measured using % Uptime of the last 90 days, published by CloudHarmony [6] and efficiency was measured using the world response time of the last 30 days, published by CloudSleuth [8]. The candidate

configurations considered for the experiment are the ones provided in Table 8.

For the experiments we assume that a user requests a virtual machine with the following features: 1 core(CPU), 1 GB(RAM) , 50 GB, 1 hour execution and with operating system Linux. Therefore, all the providers offer configurations that fulfil the requirements. Then, the configurations were prioritized by price, availability, and efficiency.

Table 8: Set of candidate providers to prioritized.

Provider	Resource	CPU (Cores)	RAM (MB)	Disk (GB)	OS	Location	Contract Period	Price	% Uptime	Response Time
Amazon EC2	Small	1	1.7	160	Linux	USA-Virginia	Hourly	$0.060	99.917%	11.98 Sec
Amazon EC2	Small	1	1.7	160	Linux	USA-Califomia	Hourly	$0.065	97.762%	12.64 Sec
Windows Azure Platform	Small	1	1.75	127	Linux	USA-Illinois	Hourly	$0.060	100.000%	10.54 Sec
GoGrid	Small	1	1	50	Linux	USA-Virginia	Hourly	$0.080	100.000%	11.78 Sec
GoGrid	Small	1	1	50	Linux	USA-Califomia	Hourly	$0.080	99.998%	12.34 Sec

Table 9: Set of candidate providers order by price.

Rank by Price	Provider	Location	Price	% Uptime	Response Time
1	**Amazon EC2**	USA-Virginia	$0.06	99.92%	11.98 Sec
2	**Windows Azure Platform**	USA-Illinois	$0.06	100.00 %	10.54 Sec
3	**Amazon EC2**	USA-California	$0.07	97.76%	12.64 Sec
4	**GoGrid**	USA-Virginia	$0.08	100.00 %	11.78 Sec
5	**GoGrid**	USA-California	$0.08	100.00 %	12.34 Sec

The suggestion that provides a decision based in price can be observed in the Table 9. This organization of candidate providers by price is adequate if this is the most important attribute for users. However, if the organizations using the cloud are able to make profit from this configuration, they will be willing to pay more, if this represents a better experience for their customers. Table 10 and Table 11 show recommendations that could be better for users that are more concerned about efficiency and availability. It is interesting to notice that GoGrid is not a good option if candidates are prioritized by price. Nevertheless, GoGrid is more relevant when availability and efficiency become more important.

Table 10: Set of candidate providers order by % uptime.

Rank by price	Provider	Location	Price	% Uptime	Response Time
2	**Windows Azure Platform**	USA-Illinois	$0.06	100.00%	10.54 Sec
4	**GoGrid**	USA-Virginia	$0.08	100.00%	11.76 Sec
5	**GoGrid**	USA-California	$0.08	100.00%	12.34 Sec
1	**Amazon EC2**	USA-Virginia	$0.06	99.92%	11.97 Sec
3	**Amazon EC2**	USA-California	$0.07	97.76%	12.63 Sec

Table 11: Set of candidate providers order by response time.

Rank by Price	Provider	Location	Price Hourly	% Uptime	Response Time
2	**Windows Azure Platform**	USA-Illinois	$0.06	100.00%	10.54 Sec
4	**GoGrid**	USA-Virginia	$0.08	100.00%	11.78 Sec
1	**Amazon EC2**	USA-Virginia	$0.06	99.92%	11.98 Sec
5	**GoGrid**	USA-California	$0.08	100.00%	12.34 Sec
3	**Amazon EC2**	USA-California	$0.07	97.76%	12.64 Sec

6 Conclusion and Future Directions

Cloud computing following the Infrastructure as a service model, promises to change the way that computing and storage resources are acquired. It gives the possibility of paying just for the resources consumed, and it allows the IT infrastructure to be hosted in external data centers. Thus, less specific hardware knowledge is required by system administrators.

This features make cloud computing and interesting option for companies. However, deploying applications in the cloud is still a complex task. It requires from system administrators skills for estimating resources required by their applications, what is difficult because system administrators frequently do not know what they really need and because estimation in the cloud and outside of the cloud can be different. Besides, administrators also need to select adequate cloud providers among of many options and type of services, and to allocate applications in one or more providers.

The goal of the project presenting in this thesis was to propose an architecture supporting system administrators in the arduous task of deploying applications in the cloud. Two main tasks were developed. First, non-functional requirements were used to estimate the resources that a given application could consume in public cloud providers. In a second task, non-functional requirements were used to improve the decision process of selecting the most convenient providers to deploy applications.

In the development of this project, as part of the system context, we explored the main concepts of cloud computing, the differences that make difficult the selection of public cloud provider among existent options, and the previous work to select and allocate resources in public cloud providers [31]. Then, we presented bibliographic reviews related to cloud monitoring services, techniques for estimation of computing resources, and existing tools to compare cloud providers.

The architecture proposed is composed of three main modules The Cloud Resource Estimator, The Resource Selection Decision Maker and the Automated Resource Allocator. We focused in the design of two first modules. In the design of The Resource Estimator, we propose an approach that uses a model of three tiers database, business and presentation, and allows estimating the resources required by applications and evaluating a set of non-functional requirements according to the priority given by the user. For Resource Selection Decision Maker, we propose to improve the selection based in price including non-functional requirements. The modifications required in the existent architecture to support the new functionality were implemented in a prototype.

Finally, experiments were performed. The first one was used to validate the performance and scalability of the architecture proposed for the Resources Estimator, which showed that it scales well regarding to non-functional requirements and its features. Additional experiments were implemented to demonstrate how the order of the candidate cloud providers changes when non-functional requirements are included; benefiting the applications where the price is not the most important issue to be considered.

Future works related to this project can focus in different directions. First, in the area of resource estimation, techniques proposed for different authors [32, 30, 36, 16] could be compared and tested, to find the one that is more accurate to estimate resources in public cloud providers.

Other efforts could address the design of monitoring services that include additional metrics to measure the level in which different cloud providers achieve non-functional requirements others than efficiency and availability. For this purpose, it is important to explore time and amount of data that a metric should consider to generate fairly comparisons. In relation to this topic, the proposed Resource Selection Decision Maker component of our architecture could be expanded, considering the best way to automatically include measurements collected the by selected monitoring services.

Moreover, The Resource Estimator could also be improved including new models to prioritize non-functional requirements, for example the model based in weight proposed by Garg et al. [11] could be an interesting option.

Finally a really interesting future challenge is to modify the architecture proposed to make it able to estimate, select and allocate resources in a dynamic way. Therefore, it would be possible for the applications to acquire more resources when it needs it or when the cloud provider does not have the behavior expected.

References

[1] Amazon. Amazon. http://aws.amazon.com/. Accessed: 2013-09-29.

[2] Amazon. Amazon EC2 Spot Instances. http://aws.amazon.com/ec2/spot-instances/. Accessed: 2013-09-29.

[3] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, 2010.

[4] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo N Calheiros. InterCloud: utility-oriented federation of cloud computing environments for scaling of application services. In *Algorithms and architectures for parallel processing*, pages 13–31. Springer, 2010.

[5] Lawrence Chung, B Nixon, E Yu, and J Mylopoulos. Non-functional requirements. *Software Engineering*.

[6] CloudHarmony. CloudHarmony. http://cloudharmony.com/. Accessed: 2013-09-23.

[7] CloudSigma. CloudSigma. http://www.cloudsigma.com/. Accessed: 2013-09-29.

[8] CloudSleuth. CloudSleuth. https://cloudsleuth.net/global-provider-view. Accessed: 2013-09-23.

[9] Dave Durkee. Why cloud computing will never be free. *Queue*, 8(4):20, 2010.

[10] Eucalyptus. Eucalyptus. http://www.eucalyptus.com//. Accessed: 2013-10-27.

[11] Saurabh Kumar Garg, Steven Versteeg, and Rajkumar Buyya. SMICloud: A framework for comparing and ranking cloud services. In *Utility and Cloud Computing (UCC), 2011 Fourth IEEE International Conference on*, pages 210–218. IEEE, 2011.

[12] GlassFish. GlassFish. https://glassfish.java.net. Accessed: 2013-10-30.

[13] GoGrid. GoGrid. http://www.gogrid.com/. Accessed: 2013-09-29.

[14] Hyperic. CloudStatus. http://www.hyperic.com/products/cloud-status-monitoring. Accessed: 2013-09-23.

[15] JBoss. RichFaces. http://www.jboss.org/richfaces. Accessed: 2013-10-30.

[16] Sajib Kundu, Raju Rangaswami, Kaushik Dutta, and Ming Zhao. Application performance modeling in a virtualized environment. In *High Performance Computer Architecture (HPCA), 2010 IEEE 16th International Symposium on*, pages 1–10. IEEE, 2010.

[17] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. Cloudcmp: comparing public cloud providers. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*, pages 1–14. ACM, 2010.

[18] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. CloudCmp: shopping for a cloud made easy. *USENIX HotCloud*, 2010.

[19] Ang Li, Xuanran Zong, Srikanth Kandula, Xiaowei Yang, and Ming Zhang. CloudProphet: predicting web application performance in the cloud. http://www.cs.duke.edu/ angl/papers/cloudprophet$_t$*r.pdf*, 2011.

[20] Ang Li, Xuanran Zong, Srikanth Kandula, Xiaowei Yang, and Ming Zhang. CloudProphet: towards application performance prediction in cloud. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 426–427. ACM, 2011.

[21] Peter Mell and Timothy Grance. The NIST definition of cloud computing (draft). *NIST special publication*, 800(145):7, 2011.

[22] Microsoft. Azure. http://www.windowsazure.com/. Accessed: 2013-09-29.

[23] mOSAIC. mOSAIC. http://developers.mosaic-cloud.eu/confluence/display/MOSAIC/mOSAIC+Java+API+-+ProgrammingGuide. Accessed: 2013-11-03.

[24] MySQL. MySQL. http://www.mysql.com/. Accessed: 2013-10-30.

[25] Nectar. Nectar Research Cloud. http://www.nectar.org.au/research-cloud. Accessed: 2013-09-29.

[26] OpenNebula. OpenNebula. http://www.opennebula.org/. Accessed: 2013-10-27.

[27] Oracle. Java EE 7. http://www.oracle.com/technetwork/java/javaee/overview/index.html. Accessed: 2013-10-30.

[28] Oracle. JavaServer Faces Technology. http://www.oracle.com/technetwork/java/javaee/javaserverf 139869.html. Accessed: 2013-10-30.

[29] Massimiliano Rak, Antonio Cuomo, and Umberto Villano. Cost/performance evaluation for cloud applications using simulation. In *Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), 2013 IEEE 22nd International Workshop on*, pages 152–157. IEEE, 2013.

[30] Shuichi Shimizu, Raju Rangaswami, Hector A Duran-Limon, and Manuel Corona-Perez. Platform-independent modeling and prediction of application resource usage characteristics. *Journal of Systems and Software*, 82(12):2117–2127, 2009.

[31] Jungmin Son. Automated decision system for efficient resource selection and allocation in inter-clouds. Master's thesis, The University of Melbourne, Australia, 2013.

[32] Christopher Stewart and Kai Shen. Performance modeling and system management for multi-component online services. In *Proceedings of the 2nd conference on Symposium on Networked Systems Design & Implementation-Volume 2*, pages 71–84. USENIX Association, 2005.

[33] David Villegas and Seyed Masoud Sadjadi. DEVA: distributed ensembles of virtual appliances in the cloud. In *Euro-Par 2011 Parallel Processing*, pages 467–478. Springer, 2011.

[34] David Villegas and Seyed Masoud Sadjadi. Mapping non-functional requirements to cloud applications. In *International Conference on Software Engineering and Knowledge Engineering*, pages 527–532, 2011.

[35] John Vlissides, R Helm, R Johnson, and E Gamma. Design patterns: Elements of reusable object-oriented software. *Reading: Addison-Wesley*, 49:120, 1995.

[36] Timothy Wood, Ludmila Cherkasova, Kivanc Ozonat, and Prashant Shenoy. Profiling and modeling resource usage of virtualized applications. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*, Middleware '08, pages 366–387, New York, NY, USA, 2008. Springer-Verlag New York, Inc.

[37] Miranda Zhang, Rajiv Ranjan, Armin Haller, Dimitrios Georgakopoulos, and Peter Strazdins. Investigating decision support techniques for automating cloud service se-

lection. In *Cloud Computing Technology and Science (CloudCom), 2012 IEEE 4th International Conference on*, pages 759–764. IEEE, 2012.

[38] Miranda Zhang, Rajiv Ranjan, Surya Nepal, Michael Menzel, and Armin Haller. A declarative recommender system for cloud infrastructure services selection. In *Economics of Grids, Clouds, Systems, and Services*, pages 102–113. Springer, 2012.

[39] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of Internet Services and Applications*, 1(1):7–18, 2010.