A Methodology and a Tool for the Design Time Exploration of Multi-Clouds Applications

Advisor: Prof. Danilo ARDAGNA
Co-Advisor: Dr. Michele CIAVOTTA
Co-Advisor: Ing. Giovanni Paolo GIBILISCO

Master Thesis by:
Riccardo Benito Desantis
Student ID 765106

Academic Year 2013-2014
Acknowledgments

I’d like to thank my family and all of my friends for all the support they offered me during my life and during my years at the Politecnico, and my girlfriend, Silvia, that helped me to go through all of the difficulties met on the road.

I’d also like to thank all the professors and assistants that helped me and taught me a number of concepts and skills that will surely help me in my whole life, not only at work.

I want also to show my gratitude to Prof. Danilo Ardagna, my advisor, and to Dr. Michele Ciavotta and Ing. Giovanni Paolo Gibilisco, my co-advisors, that supported and guided me with this thesis.

Riccardo
# Contents

**Sommario**  
12

**Abstract**  
13

1 **Introduction**  
15

2 **The Cloud Computing**  
21
   2.1 Introduction  
   2.2 Service Models  
   2.3 Deployment Models  
   2.4 Main Cloud Providers Considered  
      2.4.1 Amazon Web Services (AWS)  
      2.4.2 Google AppEngine  
      2.4.3 Microsoft Azure  

3 **State of the Art**  
43
   3.1 The Palladio Framework  
      3.1.1 Introduction  
      3.1.2 The Development Process  
      3.1.3 Simulation and System Review  
      3.1.4 Palladio and LQNs  
      3.1.5 Extending the Palladio Framework  
   3.2 The SPACE4Cloud Tool  
      3.2.1 Overview  
      3.2.2 The Multi-Provider Representation
## CONTENTS

3.3 The MILP Tool ........................................... 72  
3.4 Case Study: Apache Open For Business (OfBiz) . 74

4 The Design-Time Exploration Problem 81  
4.1 Problem Definition ................................. 81  
4.2 Problem Formulation .............................. 85  
4.3 Hybrid Local Search Metaheuristic Approach . . 96

5 Heuristic Solution for the Deployment Problem 101  
5.1 The Initial Solution Assignment ................. 101  
5.2 Bi-level Hybrid Tabu Search Algorithm ........... 103  
  5.2.1 The Make Feasible Procedure ................. 108  
  5.2.2 The Scale In Procedure ....................... 111  
  5.2.3 The Scramble Procedure ...................... 112  
5.3 The Workload Assignment ....................... 117

6 Conclusions ........................................... 127

Appendices .............................................. 129

A Structure of the Input and Output Files 131  
  A.1 Resource Model Extension ....................... 132  
  A.2 Usage Model Extension ......................... 133  
  A.3 Multi Cloud Extension ......................... 134  
  A.4 Solution ........................................ 136

[Bibliography] 143
List of Figures

2.1 The architecture of Google AppEngine. 33
2.2 All the services offered by Microsoft Azure. 37

3.1 The roles in the Palladio Component Model (see [1]). 45
3.2 The Repository Diagram of the Media Store Example. 46
3.3 The SEFF for HTTPDownload in the Media Store example. 48
3.4 The SEFF for HTTPUpload in the Media Store example. 49
3.5 The System Diagram for the Media Store Example. 51
3.6 The Resource Model for the Media Store Example. 53
3.7 The Allocation Diagram for the Media Store Example. 54
3.8 The Usage Model for the Media Store Example. 56
3.9 The mapping alternatives from Palladio Component Models to LQN (see [2]). 60
3.10 The mapping process from Palladio Component Models to LQN (see [3]). 61
3.11 An overview of the mapping performed by PCM2LQN (see [3]). 64
3.12 Overview of the main elements forming the SPACE4Cloud architecture. 67
3.13 The class diagram for the SolutionMulti class and some classes using it or being used by it. A detailed view is presented in figure 3.14. 70
3.14 The detailed class diagram for the SolutionMulti class and some classes using it or being used by it (showing only the useful methods and attributes). 71
3.15 How the file structure for the files of the solver changed. 73
3.16 The repository model of the OfBiz application. 75
3.17 The Service Effect Specification diagram of the process order functionality. 76
3.18 The system model of the OfBiz application. 77
3.19 The resource environment model of the OfBiz application. 77
3.20 The allocation model of the OfBiz application. 78
3.21 The usage model of the OfBiz application. 79
5.1 The sequence diagram of the SPACE4Cloud class using the MILP tool for finding an initial solution to the problem. 104
5.2 Main optimization procedure for the bi-level problem. 105
5.3 Make Feasible procedure. 109
5.4 Scale In procedure. 113
5.5 Scramble procedure. 116
5.6 The sequence diagram of the OptEngine class using the MILP tool for a workload assignment problem. 118
5.7 The Workload Assignment procedure in the full testing process. 121
5.8 The Maximize Utilization procedure. 122
## List of Tables

1. **Costs/hour per instance in the region EU (Ireland) for Amazon EC2.** | 30
2. **Costs/GB per moving data in the region EU (Ireland) for Amazon EC2.** | 30
3. **Costs for the EBS service for the region EU (Ireland).** | 31
4. **Amazon EC2 Instances Types.** | 32
5. **Costs per resource in the Google AppEngine system.** | 36
6. **Instance types for the Microsoft Virtual Server service.** | 39
7. **Costs/hour for each instance type in the Microsoft Azure platform, considering the EU West region.** | 40
8. **Optimization model parameters and decision variables.** | 100
# List of Algorithms

1. The full optimization algorithm for the bi-level problem. .................................................. 107
2. The algorithm for the Make Feasible procedure. .................................................. 110
3. The algorithm for the Scale In procedure. .................................................. 114
4. The algorithm for the Scramble procedure. .................................................. 115
5. Optimization algorithm with the computation of the workload percentages. .................................................. 123
6. The function assigning the workload percentages for a solution. .................................................. 124
Sommario

Negli ultimi anni, la comparsa sulla scena del paradigma del cloud computing ha cambiato radicalmente il mondo dell’informatica e delle telecomunicazioni, in un modo che si potrebbe definire rivoluzionario. Anche se le tecnologie che costituiscono il paradigma del cloud computing non sono nuove (come ad esempio le macchine virtuali, i dischi, le risorse di calcolo, le reti, etc.), l’idea di offrirle come se fossero un servizio on-demand con un metodo di pagamento a consumo è quello che è rivoluzionario.

Dato quindi questo set di nuovi strumenti, uno sviluppatore o progettista software vedrà materializzarsi nuove domande: dove fare il deployment dell’applicazione? Si deve preferire un’architettura server classica o è meglio usare una o più piattaforme cloud? Se l’unico problema è quello dei costi totali, allora la soluzione “su cloud” è molto spesso la preferibile, perché non ci sono costi iniziali e c’è la possibilità di cambiare le caratteristiche prestazionali dei servizi cloud a runtime (o quasi, ma comunque a basso costo), ma ci sono ovviamente dei nuovi problemi che nascono con la nuova piattaforma, e ne andremo a parlare nel testo.

In questa tesi considereremo i servizi disponibili nel modello cloud, e offriremo un metodo per valutare una soluzione e per trovarne una che ben si adatta al nostro problema, con costi minimi ma soddisfacendo i vincoli di QoS definiti dal programmatore in fase di progettazione.
Abstract

In recent years, the appearance on the scene of the cloud computing paradigm has radically changed the information and communications technology world, in a way that we could easily call revolutionary. Even if the technologies behind the cloud computing paradigm aren’t new (e.g., virtual machines, disks, computing resources, networks, and so on), the idea of offering them as on-demand services with a pay-per-use paradigm is what is breakthrough.

Given this new set of tools, new choices open up for a software developer or designer: where should he/she deploy his/her application? Is he/she going to prefer a classical servers architecture or one or more cloud platforms? If we consider the only problem that of the total costs, a solution “in the cloud” is quite often preferable, because there are no initial costs and there’s the possibility of scaling the performance characteristics of the cloud systems at runtime (or in an easy and cheap way), but of course some new problems arise, and we’ll talk later about them.

In this thesis we are going to consider the available services in the cloud model, and we’ll offer a method for evaluating a solution and for finding a well-suited solution which minimizes the costs while satisfying the QoS constraints defined by the developer at design time.
Chapter 1

Introduction

In recent years, the appearance on the scene of the cloud computing paradigm has radically changed the information and communications technology world, in a way that we could easily call revolutionary. Even if the technologies behind the cloud computing paradigm aren’t new (e.g., virtual machines, disks, computing resources, networks, and so on), the idea of offering them as on-demand services with a pay-per-use paradigm is what is breakthrough. As a matter of fact, most of the services offered on the Internet are now built with cloud computing resources, and more people every day consider cloud computing resources when designing a new application or a new service.

The possibility of having a highly scalable application or service is very interesting for a developer: he/she could create an application assigning very few resources at the beginning, and scale it up if the application gains popularity, or scale it in when less requests have to be served. The costs for the application depend on the resources used, so the less are the resources used, the less is the total cost of the application: knowing this, he/she will try to save as much money as possible by scaling in as soon as the resources aren’t needed. He/she will then save up money while offering a decent service, while an in-house management of the needed resources would be much more inconvenient and
much more difficult to scale up without changing the machines at all.

However, the thing is that a “decent” service isn’t often good enough, and that’s why the developers need a tool for deciding the best deployment configuration given the application, where that configuration could also consider more than one cloud provider for maximizing the availability of the application.

Being this a new and profitable market, a lot of service providers arose in these years, and the so-called cloud market is already highly diversified, with many technologies and services supplied by an ever increasing number of providers. Among the others, tools for fast prototyping, enterprise developing, testing and integration are offered, delegating to the cloud service providers all the intensive tasks of management and maintenance of the underlying infrastructure.

However, if on one hand cloud computing offers many advantages, on the other it introduces some important issues and new challenges, especially in application design and development. As a matter of fact, current available technologies and pricing models offered by cloud providers can be so complex and different among different cloud providers that determining the best deployment configuration for the services, meeting the requirements and minimizing the costs, may result in a tremendous task. To carry out such a labor, the QoS (Quality of Service) engineer should consider multiple architectures and multiple solutions at once, evaluating the costs and performances for each of them, and choose among these solutions the best one. The problem is that there are too many viable solutions, so doing this process by hand is almost impossible. Moreover, while information on architectures and costs are easily available on the Internet, it is actually quite more complex to consider the performances of those solutions. Indeed, cloud environments are usually multi-tenant
and their performance can vary with the time of day, according to the level of congestion and competition for resources among the applications.

It is clear, therefore, the need for analytical techniques and automatic tools to support performance and cost based design-time decisions, and this work attempts to fill this void. The problem is so urgent that there are already several performance models and their associated solvers that allow an approximate estimate of software performances and costs, but the downside of those models, however, is that they are hard to use since they leverage complex mathematical concepts that are outside of the average software engineer’s competence. There are higher level models that have been formalized, exactly for this reason, and those make actually use of concepts familiar to the software architect. But such models, however, do not support neither cloud-specific notions nor Quality of Service (QoS) requirements, and they still lack the proper level of automation envisioned by Model-Driven Software Development (see [4]).

In this thesis a solution for this problem has been proposed. An application or service designer will be able to use the models defined in the Palladio Framework (described in Section 3.1) to describe the application that he/she wants to design and build, defining all the QoS constraints he/she deems as useful, and the proposed tool will then compute the best solution (or actually a “cost optimized” solution, as we will see later on in Chapter 5), suggesting the providers to use and the services needed for satisfying the defined constraints and requests.

The work was done by extending the existing SPACE4Cloud tool (see Section 3.2) to make it consider applications and services deployed on multiple cloud providers, allowing the application designer to define the minimum number of providers to use or
explicitly which one of them he/she wants to use for the application (for example if he/she needs a particular service on some of these providers). Considering more than one provider when deploying an application is of vital importance if the designer wants a very reliable and fault-tolerant solution: the server farm of a cloud provider could go offline for some reason, and if all of the services offering the application were deployed there, then the application would be completely down. Having more than one provider assigned is then a failure-protection reason: it is in fact better for a cloud application designer to have part of the system handle always at least a minimum part of the workload instead of handling no requests at all. Of course, not every application needs such high availability but in some cases this is a must needed feature (e.g., applications controlling the air traffic in an airport, applications with medical uses, etc.). Using more than one provider at a time is of course more expensive than using just one, because considering a new cloud provider has a number of initial costs that don’t depends directly on the services requested. This is the beauty of this solution proposed: the developer could just choose to have one provider, and the tool will give him/her back a solution with just a provider.

The remainder of the thesis is structured as follows:

- Chapter 2 describes the cloud computing model and the services that are being considered for the problem definition and solution, talking about the main cloud providers treated and their most important services with their prices;
- Chapter 3 offers an overview of the tools used and the concepts behind them, going from the Palladio models to the SPACE4Cloud and MILP tools;
- Chapter 4 describes the optimization problem and how it is simplified to make it solvable in an heuristic way;
• Chapter 5 fully describes the solution to the problem;

• Chapter 6 wraps up the work done, underlying the achieved results and presenting future research directions.
Chapter 6

Conclusions

In this thesis we talked about the problems that an application designer or developer has to face when deciding of developing its application or service on the cloud, talking especially about the minimization of the costs while guaranteeing QoS constraints expressed in terms of average response times.

This is a classic $\mathcal{NP}$-hard optimization problem, and we have proposed a formulation and an heuristic solution algorithm focused on the selection of the beneficial set of cloud providers among those available, the services (that in our example are virtual machines of some kind) offered by those cloud providers to be used for satisfying the requirements and the structure of the application to be developed, and the workload distribution among the providers for satisfying the availability requests of the application developed. This is done on an hourly basis over the full set of providers and services, minimizing the costs while guaranteeing the respect of the QoS constraints.

The solution proposed exploits the SPACE4Cloud and the MILP tool, extended for the sake of the problem for supporting the multi-provider case and the requirements of availability, this tool can now be considered by an application developer in his/her designing process.
An extensive analysis of our proposed solution has been performed, considering multiple workloads with a fixed system configuration, with an increasing number of cloud providers, analyzing in this way the scalability of our solution. The tool takes an average of 20 minutes for evaluating an application and for giving its best solution, but this highly depends on the characteristics of the problem, of course, as described in Chapter ?? in which the experimental results are reported.

The tool is highly configurable, making it possible to select a number of characteristics of the search, and making it work quickly while still approaching the best solution.

Future work will consider also the cloud bursting problem and thus the possibility of using private clouds deployed on local resources for reducing the costs even more. Also the tool could be improved by considering even more cloud providers and different kinds of services (e.g., PaaS systems other than IaaS systems). Lastly, it could be interesting testing the proposed solution by actually deploying the application considered in the services proposed and check if the costs and the performances were predicted correctly.
Bibliography


