FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Quality of Service Control Mechanisms in Cloud Computing Environments

DISSERTATION

zur Erlangung des akademischen Grades

Doktorin der Technischen Wissenschaften

eingereicht von

Soodeh Farokhi MSc.
Matrikelnummer 1228800

an der Fakultät für Informatik
der Technischen Universität Wien

Betreuung: Priv.-Doz. Dr. Ivona Brandić

Diese Dissertation haben begutachtet:

Priv.-Doz. Dr. Ivona Brandić	Univ.-Prof. Dr. Erich Schikuta

Wien, 3. Dezember 2015

Soodeh Farokhi

FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Quality of Service Control Mechanisms in Cloud Computing Environments

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktorin der Technischen Wissenschaften

by

Soodeh Farokhi MSc.
Registration Number 1228800

to the Faculty of Informatics
at the Vienna University of Technology

Advisor: Priv.-Doz. Dr. Ivona Brandić

The dissertation has been reviewed by:

Priv.-Doz. Dr. Ivona Brandić	Univ.-Prof. Dr. Erich Schikuta

Vienna, 3rd December, 2015

Soodeh Farokhi

Erklärung zur Verfassung der Arbeit

Soodeh Farokhi MSc.
Taborstrasse 27/3/21, 1020 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 3. Dezember 2015

Soodeh Farokhi

Acknowledgements

I firmly believe that this work would have never been possible without the profound support of many people. First of all, I would like to express my gratitude to my supervisor, Dr. Ivona Brandić who has guided me over the last three years, helped me to develop new ideas and given me the support needed to pursue my research unimpeded. I am also very grateful to my co-advisor, Prof. Helmut Veith, who gave me the opportunity to start my PhD study at Vienna University of Technology, and for his advice and support during the early stage of my research. A special appreciation goes to Prof. Erich Schikuta, who kindly accepts to review my dissertation as the second examiner.

In my journey, I have been fortunate to collaborate with several amazing people who show me how exciting, appealing and fun research can be. In particular, I acknowledge the contributions of Pooyan Jamshidi, whose constant support and advice, informative discussions, and his enthusiasm for research enlighten my PhD journey. A special appreciation goes to my collaborators from Umeå University, who made my visit in Sweden a very pleasant and fruitful research experience. Especially, to Prof. Erik Elmroth for his informative comments, and kind advice. To Ewnetu Bayuh Lakew, the co-author of my papers, who has been a continual source of helpful discussions, support, and encouragement throughout the last year of my study.

I have been really lucky to have the support and friendship of my great Croatian colleagues, Dražen Lučanin, Ivan Bresković, and Toni Mastelić. Thank you for bringing new ideas and knowledge to the group and making my PhD more fun. My big thanks goes to Toni Mastelić, whose incredible support, especially on my dissertation, paved the PhD path a lot for me. I would also like to thank my friend, David Williams, who has given me useful hints for my papers, and inspired me to learn more and believe in myself.

My deepest and sincere thanks goes to my family for their unconditional love, understanding and support. Mom, Dad, Saeed, and my lovely little sister, Razgol; you all have been the main source of encouragement in my life, and I can never thank you enough.

Most of all, I would like to thank the most precious person in my life, my husband Amir, for his amazing support during all the ten years that we have been together, and especially during these three years of my study. Amir, you have always believed in me, listened to me with patience and smiles. You gave me the biggest inspiration and confidence to become who I am now. You taught me how joyful life can be even, when you are experiencing the hardest moments of your life while being far from your family. There is no doubt that this accomplishment is yours, too.

Soodeh Farokhi
Vienna, December 2015

The research leading to this thesis has received funding from the following projects: “adaptive distributed systems“ doctoral college, Vienna University of Technology; “holistic energy efficient management of hybrid clouds“ (HALEY) project, granted under distinguished young scientist award by Vienna University of Technology; the Vienna science and technology fund (WWTF) through the PROSEED grant; the Austrian national research network S11403 and S11405 (RiSE) of the Austrian science fund (FWF); the short term scientific mission (STSM) grant by information and communication technologies COST action IC1304 “autonomous control for a reliable Internet of services“ (ACROSS).

To my closest friend, my husband, ***Amir****,*
whose unfailing love and support has made the pursuit of my dreams possible.

Kurzfassung

Die steigende Beliebtheit des Internets führte in Kombination mit dem rasanten Fortschritt bei Verarbeitungs- und Speichertechnologien zu einem Paradigmenwechsel der Ressourcenverwaltung. Heutzutage werden Ressourcen als Internetservice gemietet, entweder als Pay-as-you-Go oder On-Demand Dienst: Das sogenannte Cloud-Computing. Das sowohl von der Industrie als auch von akademischer Seite her wachsende Interesse am Cloud-Computing führt zu einer steigenden Anzahl an Cloud-Service Anbietern und Nutzern.

Cloud-Infrastrukturanbieter versuchen sich durch höhere Qualität von den anderen Anbietern abzugrenzen, bei gleichzeitiger Verringerung der Betriebskosten. Durch das stete Wachstum und die geographische Verteilung der Rechenzentren wird dies zunehmend anspruchsvoll. Das Hauptziel der Kunden, welche auf die Cloud statt auf eine IT-Infrastruktur setzen, liegt darin, eine hohe Dienstgüte (QoS) bei gleichzeitiger Kostenreduktion zu erreichen. Die Kunden werden bedingt durch die Vielzahl an Angeboten in Bezug auf Qualität und Kosten ermutigt, gleichzeitig Services mehrerer Cloud-Anbieter zu nutzen, die sogenannte Multi-Cloud. Die Verwendung von Multi-Cloud Services führt jedoch zu neuen Herausforderungen bei der Auswahl und Zusammenstellung von geeigneten Diensten. Im allgemeinen bieten Cloud-Anbieter keine Leistungsgarantie, obwohl Kunden dringenden Bedarf an garantierter Serviceleistung haben. Es fehlen praktikable Ansätze diese kosteneffektiv zu erreichen. Die Komplexität entsteht aufgrund der dynamischen Natur der Cloud, unkalkulierbarer Auslastung und der nichtlinearen Zuordenbarkeit von Leistungsbedarf auf Cloud Ressourcen.

Die Abwägung zwischen Dienstgüte und Kosten ist sowohl für Infrastrukturanbieter als auch Kunden ein herausforderndes Ziel. Diese Arbeit untersucht Modelle, Algorithmen und Mechanismen um den Zielkonflikt von beiden Perspektiven zu betrachten. Zuerst wird der Cloud-Anbieter Standpunkt vertreten, indem ein Platzierungsalgorithmus für virtuelle Maschinen bei geographisch verteiler Infrastruktur vorgestellt wird. Dabei wird ein Bayesisches Netz zur Entscheidungsfindung bei Unsicherheit verwendet. Anschließend adressieren wir die Kosten in Korrelation mit der Service-Dienstgüte vom Standpunkt des Kunden aus durch Nutzung des Multi-Cloud Paradigmas. Wir schlagen eine auf Erwartungstheorie basierende Serviceauswahl vor, um vergleichbare Angebote zu klassifizieren. Weiters schlagen wir autonome Ressourcenzuteilungstechnologien vor, um die Leistungszielvorgabe der Kunden zu garantieren. Zu diesem Zweck wird mittels Kontrolltheorie ein Ressourcenzuteilungs-Controller entwickelt und mehrere Controller werden mittels Fuzzyregelung zum Erreichen der geforderten Leistungsziele kosteneffektiv koordiniert. Schlußendlich wird das Ergebnis dieses Ansatzes mit dem existierenden Stand der Technik verglichen.

Abstract

The growth in popularity of the Internet, along with the rapid development of processing and storage technologies, has brought a paradigm shift in the way computing resources are provisioned. The technological trend today is to offer computing resources as services, leased and exposed via the Internet in a pay-as-you-go and on-demand fashion, called cloud computing. The interest in cloud computing is growing in both industry and academia, so the number of cloud providers offering their services, and the number of cloud customers interested in using such services is rapidly increasing.

Cloud infrastructure providers are trying to reduce their operating costs while offering their services with higher quality; something they strive to do to stand out among other providers. However, this is becoming challenging as providing such services needs operating large-scale and geographically distributed data centers. On the other hand, the main purpose of customers in using clouds is to achieve a high quality of service (QoS) while reducing their overall costs. Given the variety of offered services in terms of quality and cost, customers are encouraged to simultaneously use services from multiple cloud providers, known as multi-cloud. However, utilizing multi-cloud brings a new set of open challenges, such as selecting and composing the most appropriate services. Furthermore, despite the critical need of customers in having predictable service performance, in general cloud providers do not yet offer any performance guarantees. This gap is due to the complexity of practically addressing this issue in a cost-effective way. Such a complexity mainly comes from the dynamic nature of the cloud, unpredictable workloads, and non-linearity of mapping performance measurements into required cloud resources. Hence, controlling the trade-off between QoS and cost is a challenging goal for both cloud infrastructure providers and customers.

This thesis investigates models, algorithms, and mechanisms to tackle this trade-off from both perspectives. More specifically, in the scope of this thesis, we first take the cloud provider viewpoint by proposing an approach for virtual machine placement across geographically distributed infrastructures. In this approach, a Bayesian network model is used to address decision making under uncertainty. Then, we address the trade-off between QoS and cost from the cloud customer point of view by facilitating the utilization of the multi-cloud paradigm. We propose a service selection approach using prospect theory to rank the comparable service offerings. Furthermore, to guarantee the performance objectives of customers, we propose autonomic resource provisioning techniques. To this aim, control theory is used to design resource provisioning controllers, and fuzzy control is utilized to coordinate multiple controllers toward meeting the service performance objectives in a cost-effective manner. Finally, the evaluations of these contributions in comparison with the state-of-the-art approaches are presented.

Contents

List of Figures

List of Tables

List of Algorithms

Selected Publications

This thesis is based on work published in scientific conferences, workshops, and journals. For reasons of brevity, these core papers, which build the foundation of this thesis, are listed here once and will not be explicitly referenced again. Parts of these papers are contained in verbatim. Please refer to Appendix A for a full publication list.

Refereed publications in journals

1. **Soodeh Farokhi**, Pooyan Jamshidi, Ewnetu Bayuh Lakew, Ivona Brandic, and Erik Elmroth. *A Hybrid Cloud Controller for Vertical Memory Elasticity: A Control-theoretic Approach.* Future Generation Computer Systems (*FGCS*), The International Journal of Grid Computing and eScience, Elsevier (*under review*) [62].

Refereed publications in conference proceedings and workshops

2. **Soodeh Farokhi**, Foued Jrad, Ivona Brandic, and Achim Streit. *HS4MC: Hierarchical SLA-Based Service Selection for Multi-Cloud Environments.* 4th International Conference on Cloud Computing and Services Science (*CLOSER 2014*). pp. 722-734, SciTe Press. Barcelona, Spain. April 3-5, 2014 [64]. doi: 10.5220/0004979707220734

3. **Soodeh Farokhi**. *Towards an SLA-based Service Allocation in Multi-Cloud Environments.* 14th International Symposium on Cluster, Cloud and Grid Computing (*CCGrid 2014*). pp. 591-594. IEEE/ACM. Chicago, USA. May 26-29, 2014 [60]. doi: 10.1109/CCGrid.2014.62

4. **Soodeh Farokhi**, Pooyan Jamshidi, Ivona Brandic, and Erik Elmroth. *Self-adaptation Challenges for Cloud-based Applications: A Control Theoretic Perspective.* 10th International Workshop on Feedback Computing (*Feedback Computing 2015*). Seattle, USA. April 13, 2015 [61]. doi: 10.13140/RG.2.1.1810.1204

5. **Soodeh Farokhi**, Pooyan Jamshidi, Drazen Lucanin, and Ivona Brandic. *Performance Based Vertical Memory Elasticity.* 12th International Conference on Autonomic Computing (*ICAC 2015*). pp. 151-152, IEEE. Gronoble, France. July 7-10, 2015 [63]. doi: 10.1109/ICAC.2015.51

6. **Soodeh Farokhi***, Dmytro Grygorenko* and Ivona Brandic. *Cost-Aware VM Placement across Distributed Data Centers using Bayesian Networks.* 12th International Conference on Economics of Grids, Clouds, Systems and Services (*GECON 2015*). Springer LNCS. Cluj-Napoca, Romania. Sep 15-17 2015 (*contributed equally) [82]. doi: 10.13140/RG.2.1.2727.6247

7. **Soodeh Farokhi***, Ewnetu Bayuh Lakew*, Cristian Klein, Ivona Brandic, Erik Elmroth. *Coordinating CPU and Memory Elasticity Controllers to Meet Response Time Constraints.* International Conference on Cloud and Autonomic Computing (*ICCAC 2015*). pp. 69-80, IEEE. Cambridge, MA, USA. Sep 21-24, 2015 (*contributed equally) [65]. doi: 10.1109/ICCAC.2015.20

CHAPTER 1

Introduction

Cloud computing has emerged as a promising computing paradigm for providing "ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources (e.g., networks, servers, storage, applications and services) that can be rapidly provisioned and released with minimal management effort or provider interaction" [131] in a pay-as-you-go manner. The advantages of using cloud services have been increasingly encouraging customers to adopt them. The granularity of the offered service varies from low-level to high-level and are often categorized into three cloud delivery models:

1. **Infrastructure-as-a-Service** (IaaS). It allows cloud customers to run and deploy arbitrary software on a set of infrastructure, including fundamental computing resources such as storage, and virtual machine (VM). In this model, the customer has full control on both the application and the hosting infrastructure. A notable example of IaaS offering is Amazon Elastic Compute Cloud (EC2[1])
2. **Platform-as-a-Service** (PaaS). It offers cloud customers to use a hosting environment, including the programming languages, libraries, and tools, for their applications. While the customers have the capability of controlling their application, they have limited control of the operating system, hardware, and network devices in the hosting environment. A notable example of PaaS offering is Google App Engine[2].
3. **Software-as-a-Service** (SaaS). It offers software applications running on top of cloud infrastructures. The cloud customer can only use the application, i.e., neither the infrastructure nor the application can be controlled by the customer. Notable examples of this model are storage services such as Dropbox[3], or streaming services such as Netflix[4].

[1]Amazon EC2: `http://aws.amazon.com/ec2`
[2]Google App Engine: `http://developers.google.com/appengine`
[3]Dropbox: `http://www.dropbox.com`
[4]Netflix: `http://www.netflix.com`

Among the advantages of adapting cloud services, elasticity is the main selling point for cloud computing [96, 112, 90, 150], and it is a characteristic that differentiates it from previously proposed computing paradigms such as grid computing [71]. Elasticity is defined as the degree to which a cloud service is able to accommodate the varying demands at runtime by dynamically provisioning and releasing resources, such that the available resources match the current demands closely [90]. This can be realized by using a component named elasticity controller.

In cloud computing, provisioning the services is negotiated by means of service level agreements (SLAs). An SLA is defined as an agreement between a cloud provider and a cloud customer. It is expressed in terms of quality of service (QoS) parameters, i.e., measurable levels of non-functional attributes such as availability, scalability, and performance, as well as the service cost. If the cloud provider fails to adhere to this agreement (i.e., violates the SLA), the cloud customer can claim penalties that result in missed revenues, and decreased the provider's reputation. Therefore, the goal of any cloud provider is to meet the SLAs, while reducing the total cost of offering its services.

In the research carried out in the scope of this thesis, we mainly focus on cloud infrastructure services. This thesis develops different control mechanisms, including algorithms, models, and techniques to address the trade-off between QoS and cost for both cloud infrastructure providers and customers. In the following sections, we present the problem statement, research questions that motivate our work, and major scientific contributions that are addressed in the scope of this thesis.

1.1 Problem Statement

With the growth in popularity of cloud computing from both industry and academia, the number of cloud providers offering their services and the number of cloud customers interested in using such services is rapidly increasing. Cloud infrastructure providers are responding to this situation by offering their services with higher quality in order to stand out among other providers, while keeping their operating costs low. On the other hand, the main motivation of customers in utilizing cloud services rather than operating their own infrastructure is to achieve a high QoS while reducing the overall cost. Therefore, balancing the **trade-off between QoS and cost** is the interest of both the cloud provider and customer.

From the **cloud provider viewpoint**, addressing this trade-off is challenging as the complexity of cloud data centers are increasing, i.e., they are getting larger in scale and geographically distributed. For instance, Google has twelve cloud data centers across four continents [123]. However, control and management of such data centers in a cost efficient way, while avoiding QoS degradation is still a major challenge [116].

Tackling the trade-off between QoS and cost is also challenging from the **cloud customer perspective**. On one hand, to benefit from the variety of offered services with different quality and cost, customers are increasingly encouraged to simultaneously use services from multiple cloud providers [48], known as multi-cloud. However, multi-cloud itself brings a new set of challenges such as selecting and composing the most appropriate

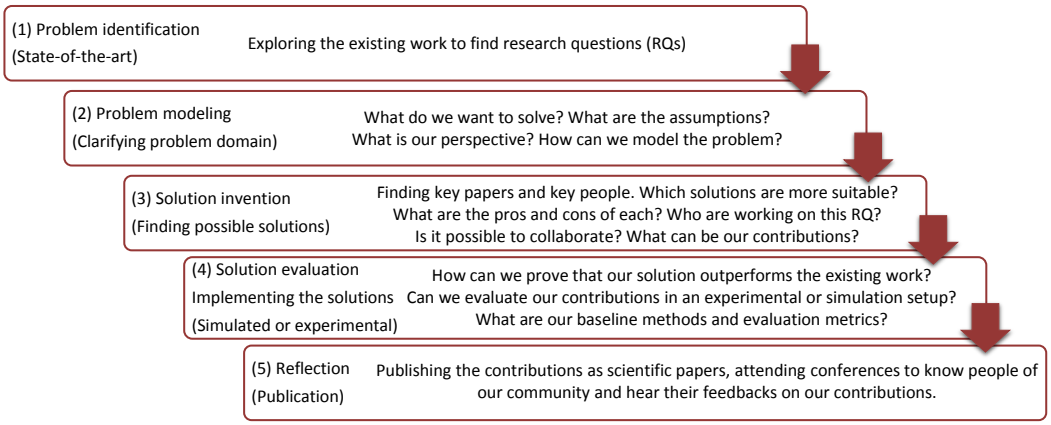


Figure 1.1: Overview of our research methodology, according to design-science [178]

services that best satisfy the customer in terms of QoS and cost [184]. On the other hand, despite the recent expectation of customers of having predictable performance, existing cloud providers do not offer any performance guarantees [111]. Due to the dynamic nature of cloud environments, unpredictable runtime workloads, and the non-linearity of mapping performance measurements to required resources [111, 171, 56, 24], having control mechanisms to provide performance guarantees in a cost effective way is still an unresolved challenge in the cloud.

1.2 Research Methodology

Two fundamental paradigms determine the research in the domain of information systems: behavioral-science and design-science [178]. The behavioral-science looks for developing and verifying theories that explain human or organizational behavior. While, the design-science is a methodology to seek the boundaries of human capabilities through building and evaluating innovative software artifacts. Since the synthetic nature of software engineering is in line with the subject of study at the design-science paradigm [95], we align the research carried out in this thesis with the design-science methodology.

The artifacts that are developed for this thesis are presented as a set of algorithms, models, and techniques for controlling the trade-off between QoS and cost for both the cloud infrastructure providers and customers. The application domain of the mentioned research artifacts is cloud computing. Taking the general guidelines for conducting research based on the design-science methodology, we perform our research according to the following steps:

1. **Problem identification**. In this step, we identify the research questions related to cloud computing, as the general domain of our research. Based on the problem statement introduced in Section 1.1, the problem that is tackled in the scope of this thesis is to control the trade-off between QoS and cost in cloud environments from two different perspectives: the cloud providers' and the cloud customers'. In

order to identify the research gap in the specified domain, we perform a systematic literature review to get to know the existing research work. Such a study enables us to identify the primary research questions, which are presented later in Section 1.3.

2. **Problem modeling**. In this step, we model the research questions extracted from the problem specified in the previous step. Modeling the problem by considering the assumptions and the corresponding perspectives is the output of this step. To this aim, we use various modeling techniques and languages to model the cloud environments, first from the perspective of an infrastructure provider, then from the viewpoint of an infrastructure customer, such as an application owner. We accordingly set the assumptions in these models in such a way to focus on the trade-off between QoS and cost.
3. **Solution invention**. In this step, we conduct a set of activities required to identify the potential solutions for addressing the extracted research questions based on the designed models of the previous step. Meanwhile, identifying the key researchers who are also working on the same research problems and complementary solutions can lead to establish possible collaboration in this step. The scientific contributions, which are presented later in Section 1.4, are the outputs of this step.
4. **Solution evaluation**. In this step, we evaluate the proposed contributions through controlled experiments on the developed techniques and mechanisms. Controlled experiments are frequently used to evaluate and validate research artifact correctness and how precisely the research goals are met through measurement of the various criteria [96]. It provides a better understanding of the problem and gives feedback to improve the mechanisms. Such experiments can also explain the contributions of the proposed techniques compared to the existing practices. The aim of this step is to show the ability of the proposed contributions on outperforming the state-of-the-art solutions. We use the existing solutions as baseline approaches to be compared with our proposed approaches. In this step is required to define the evaluation environment either a simulated or an experimental setup, along with the evaluation metrics, which reflect the validity of the contributions.
5. **Reflection**. Reflection consists of activities that involve illustration of the research impact on a specific research community. In the software engineering domain and in particular cloud computing, the research outcomes are communicated through publications in scientific conference proceedings, workshops, and journals. Moreover, releasing the source code of the implemented techniques is often used to facilitate the reproducibility of the research.

1.3 Research Questions

The challenges and problems identified in Section 1.1 serve as motivation for the research conducted throughout this thesis. In this section, we derive five primary research questions (RQs) that are addressed in this thesis.

Research Question I

How can a cloud provider control geographically distributed cloud infrastructures to reduce the operating costs while providing high QoS?

In recent years, cloud infrastructure providers have been trying to offer highly available, and scalable cloud services to surpass in the competitive market of various cloud service offerings by keeping their customers satisfied. Providing such high quality services requires having large-scale and geographically distributed cloud infrastructures. This has raised serious management issues for cloud providers, since they usually use inefficient cloud control and management solutions that only work for small-scale and centralized cloud infrastructure [130]. Providing cost-efficient control actions in geographically distributed cloud infrastructures while avoiding SLAs violation needs addressing several time- and location-dependent external and internal factors under some levels of uncertainty [116]. Some of these factors are regional power-outages, temperature, regional electricity prices, and unpredictable resource demands.

Research Question II

How can a cloud customer select services from multiple cloud providers to achieve the best combination of cost and QoS offerings?

Cloud customers are increasingly interested in simultaneously provisioning services from multiple cloud providers in order to have a wider range of cost and QoS offerings. However, selecting and composing the most suitable service offerings from multiple cloud providers is still an unresolved problem. This problem is challenging because the selected services should best satisfy the cloud customer in terms of the requested QoS while reducing the overall cloud leasing cost. Beside the challenges related to service selection such as efficient scoring of different service offerings, addressing other open issues caused by heterogeneous SLAs in such an environment is essential, too.

Research Question III

How to guarantee the performance objectives of a cloud application despite its dynamic runtime workload?

Web applications are usually exposed to dynamic, and unpredictable workload at runtime. Since cloud elasticity provides the ability to rapidly decide the right amount of resources needed, the application owners tend to target clouds as a fertile deployment environment. By using clouds, they are able to better satisfy the application QoS objectives, while reducing the resource cost. Horizontal and vertical are two types of cloud elasticity strategies. While horizontal elasticity is the ability to acquire or release the virtual machines, vertical elasticity is adjusting the capacity (e.g., memory, CPU cores) of individual virtual machines hosting the application to cope with runtime changes. Although vertical elasticity is recognized as a key enabler for efficient resource utilization of cloud infrastructure through fine-grained resource provisioning [138], a large body of research work as well as the commercial clouds have been only focused on horizontal elasticity. Nevertheless, only a few research efforts have addressed vertical elasticity, and

among them the approaches which are able to guarantee the application performance objectives are still scarce.

Research Question IV
How to make cloud applications vertically elastic to guarantee the performance objectives while using resources efficiently?

A commonly used vertical elasticity approach is a capacity-based, where the decision is made only based on the resource utilization. Only recently, a new trend has been realized by performance-based approaches, where the application performance is used as a decision making criterion. Although in a capacity-based approach a higher resource utilization can be achieved, unexpected runtime workloads can cause poor service performance, kill the end users' satisfactions, and eventually reduce the revenue of the cloud application owner [136]. On the other hand, a performance-based approach may cause resource over-provisioning as it does not have enough insight into the current resource utilization at runtime. Therefore, there is a need for vertical elasticity solutions that can guarantee the application performance while at the same time achieving a high resource utilization in spite of the varying application workload at runtime.

Research Question V
How to coordinate multiple elasticity controllers for a cloud application to efficiently meet the performance objectives?

The research efforts made on vertical elasticity mostly focus on a single resource (e.g., CPU cores, memory). The underlying assumption made in such approaches is that the application intensively uses one resource at runtime (e.g., it is either CPU-intensive or memory-intensive), while the application performance is rarely considered. However, during the application lifespan, it can show multiple resource intensive characteristics depending on the nature of the workload, which may affect its performance unless it is properly dealt with. However, existing techniques that support the elasticity of a single resource cannot readily be used as-is for scaling multiple resources at the same time. This is because uncoordinated control actions by different controllers may lead to sub-optimal or inconsistent resource allocations which may in turn result in SLA violations [52]. Therefore, a solution is needed that under some level of uncertainty dynamically performs coordinated adjustments of each resource allocation in order to meet the performance objectives of a cloud application.

1.4 Scientific Contributions

Guided by the research questions presented in Section 1.3, the core scientific contributions are highlighted in this section to provide an outline of the work carried out in the scope of this thesis. These contributions have been previously published in several scientific journals, conference proceedings, and workshops. In the following, for each contribution the original published reference is specified.

Scientific Contribution I

A cloud control mechanism to support VM placement across geographically distributed cloud infrastructures using Bayesian networks.

Taking the perspective of a cloud provider, who aims to manage geographically distributed infrastructure, we propose and develop a virtual machine placement approach to select the most suitable physical hosts for both the newly requested VMs and running VMs. Such an approach reduces the infrastructure operating costs while avoiding the customers' SLA violations. The novelty of our work lies in addressing the decision making under uncertainty in the cloud infrastructure management by modeling expert domain knowledge with Bayesian networks [137]. The constructed Bayesian network model is able to make control decisions regarding VM placement while taking into account time- and location-dependent factors under some levels of uncertainty. The proposed VM placement approach is evaluated in a realistic simulation setup and compared with two state-of-the-art baseline approaches. This contribution addresses RQ I, previously introduced in Section 1.3. It has been originally published in [82], and is discussed in detail in Chapter 3.

Scientific Contribution II

An SLA-based service selection approach for a multiple cloud environment

We present and develop a novel service selection approach which is placed between the cloud customer and the cloud providers in a multiple cloud environment. It enables the cloud customer to find and compose the best set of cloud service offerings that satisfy the SLA in terms of the requested QoS and the overall leasing cost. The novelty of our approach lies in addressing the problem of ranking comparable services offered by multiple cloud providers with respect to the customer's satisfaction, using prospect theory [102]. The proposed approach first constructs a set of SLAs to cope with the heterogeneity of different providers' SLAs. Then, it selects and composes a set of services that most closely fulfills the customer's SLA. We evaluate our approach in a realistic simulated environment by comparing the results with a state-of-the-art utility-based selection algorithm. This contribution addresses RQ II presented in Section 1.3. It has been originally published in [64, 60] and is fully elaborated in Chapter 4.

Scientific Contribution III

A performance-based vertical memory elasticity controller for cloud applications.

In order to enable the cloud customer (i.e., an application owner) to meet the application performance objectives via a fine-grained autonomic resource provisioning, we propose a vertical memory elasticity controller. To design this controller, we use control theory to guarantee the application performance by adjusting the allocated memory as a control knob. The main benefits of realizing such a controller by using control theory are: being fast and robust; having adjustable parameters; and being grounded on a solid mathematical background [125]. The novelty of our work lies in applying a control

design process that guarantees the robustness and stability of the controller, while taking into account the application response time as a decision making criterion. To verify the efficiency of the implemented controller for meeting the application response time despite varying workloads, we run an experimental study on a cloud benchmark application deployed in a virtualized environment under real-world workloads. This contribution addresses RQ III introduced in Section 1.3. It has been originally published in [63, 61] and is discussed in Chapter 5.

Scientific Contribution IV

A hybrid vertical memory elasticity controller for cloud applications, taking into account both resource utilization and application performance.

A commonly used vertical elasticity approach is a capacity-based that decides based on the resource utilization. In a new trend, performance-based approaches (e.g., contribution III) are coming into play in which the application performance is used as a decision making criterion. In the scope of this contribution, these two approaches are discussed and a novel hybrid approach is proposed. The hybrid approach makes memory elasticity decisions based on both the application performance and the resource utilization to leverage the benefits of both mentioned approaches. To this aim, we use control theory to synthesize a feedback controller that maintains the application response time while achieving a high resource utilization by adjusting the allocated memory. The proposed controller is implemented and evaluated in an experimental setup, using a cloud benchmark application under both synthetic and real-world workloads, and compared with two baseline controllers. This contribution addresses RQ IV discussed in Section 1.3. It has been originally presented in [62], and is elaborated in Chapter 5.

Scientific Contribution V

Using fuzzy control to coordinate CPU and memory elasticity controllers to meet the application performance objectives.

We design and implement a fuzzy coordination approach for multiple resource coordination to guarantee the application performance. The proposed approach consists of three sub-controllers: fuzzy controller, CPU controller, and memory controller. The fuzzy controller acts as a coordinator and dynamically infers the degree of the contributions of both CPU and memory to the application performance change. By using such a coordinator, the control actions of the CPU and memory controllers complement each other in order to fulfill the application performance objectives, without over-provisioning any of the resource types. We perform a thorough experimental evaluation of the proposed approach compared to a baseline approach using three different cloud benchmark applications under various workload patterns. This contribution addresses RQ V, has been originally published in [65], and is presented in details in Chapter 6.

1.5 Structure of the Thesis

According to the research questions and the scientific contributions presented in this chapter, the remaining of the thesis is organized as follows. The five scientific contributions outlined in Section 1.4 are grouped into four main chapters (Chapters 3 to 6).

- Chapter 2 discusses the cloud computing concepts as well as the primary well-established theories, techniques, and models from other domains used in the scope of this thesis.
- Chapter 3 discusses a VM placement approach that enables the cloud providers to manage and control multiple geographically distributed cloud infrastructures. The goal is to reduce the operating costs while keeping the cloud customers satisfied in terms of QoS. We first present the motivation and then provide the details of the proposed models, and the algorithms.
- Chapter 4 introduces a service allocation framework applied in a multiple cloud environment. The main focus of the chapter is on the service selection by proposing an approach that enables the cloud customer to choose the best set of services, in terms of QoS and cost, from multiple cloud providers. Finally, the service selection model and algorithm are presented.
- Chapter 5 first motivates the effect of vertical memory elasticity on the application performance. Then, it follows a control design process used in control theory and elaborates the design process of the two proposed memory elasticity controllers.
- Chapter 6 presents an abstract coordination model for elasticity controllers. Then as the focus on the chapter, it proposed a fuzzy coordination approach that controls different resource types to meet the application performance objectives by coordinating multiple elasticity controllers. The process of designing the fuzzy controller is elaborated in the remaining of the chapter.
- Chapter 7 describes the evaluation of each contribution proposed in the scope of this thesis (Section 1.4). The first two contributions are evaluated in a simulation-based environment, while the other three contributions are evaluated in experimental setups. The evaluation of each contribution is discussed in a separated section. Each section covers the details of the evaluation setup and the baseline approaches, as well as an extensive discussion on the achieved results.
- Chapter 8 presents the related work, compares it to the work carried out in this thesis, and outlines the enhancements that this thesis has brought. The research work presented in this chapter is divided into four sections: (i) cloud infrastructure management; (ii) service selection in multiple clouds; (iii) vertical resource elasticity; and (iv) control-theoretical approaches for cloud elasticity.
- Chapter 9 concludes the thesis while discussing the limitations of the presented contributions, and providing an outlook into possible future directions.

CHAPTER 2

Background

This chapter provides background information about well-established concepts and technologies which form the basis of the work carried out in the scope of this thesis. We first illustrate and clearly define the cloud computing concepts which are used in our research, and then introduce the models, theories, and concepts that are utilized in our solutions from other scientific domains.

2.1 Cloud Computing Concepts

Cloud computing is an emerging paradigm that is based on different areas and technologies, such as Internet, service-oriented architecture (SOA), grid computing, and the virtualization technology. Cloud computing offers services that follow pay-as-you-use and on-demand computing models to customers. One of the core technologies that enable cloud computing as a popular paradigm is virtualization [72, 111].

The virtualization technology facilitates cloud resource management and provides efficient resource utilization by providing the ability of sharing resources in the form of virtual machines [180]. A virtual machine is typically a basic building block running a separate operating system (OS), which can be easily started, stopped, hibernated, or migrated [106]. The virtualization technology also enables adding or removing the resources such as CPU, memory, storage, and communication bandwidth from one VM to another, on demand at runtime [111]. The software that provides the virtualization is called a virtual machine monitor or hypervisor. A hypervisor virtualizes all the resources of physical machines, thereby supporting the execution of multiple VMs [164] in a single physical host. In the scope of this thesis, we mainly work with Xen [28], and KVM [86] hypervisors.

2.1.1 Cloud Deployment Models

The deployment models in clouds are the different ways of using cloud infrastructure and services. The most common categorization is as follows [158, 128]:

1. **Public cloud**. In a public cloud, services are available publicly to all customers through the Internet. The public cloud providers have well-defined pricing models and accounting mechanisms for their offered services.
2. **Private cloud**. This deployment model is used by organizations for deploying their private applications in their in-house data centers. Access to a private cloud is being granted only to the members of the organization.
3. **Hybrid cloud**. A hybrid cloud deployment model represents a combination of at least two distinct clouds that connects two or more clouds in terms of their deployment models (e.g., public and private) [131]. Often a hybrid cloud model is used in the case of workload bursting, i.e., the usage of external cloud services when the private cloud is not sufficient [149].

In the case of a hybrid cloud model, if the underling clouds, which are simultaneously used by a customer, are restricted to only public clouds, it is called multiple clouds [48]. In other words, in a multiple cloud model, a cloud customer, e.g., an application owner, deploys the application simultaneously on services from multiple cloud providers. In general, two types of delivery models exist for multiple clouds [148]: *federated-cloud* and *multi-cloud*. These models differ in the degree of collaborations between the involved cloud providers and the way the cloud customer interacts with them:

- **Federated-cloud model**. In this model, the cloud providers are in agreement with each other to provide a federation in order to enhance the services offered to their customers. In this model the usage of multiple cloud services is transparent from the cloud customer viewpoint.
- **Multi-cloud model**. A multi-cloud model represents the usage of multiple, independent clouds by a cloud customer. This model does not imply interconnection and sharing among the clouds [149], i.e., there is no need for an agreement among them. Furthermore, in this model, the cloud customer is aware of using services from multiple clouds, and usually a third party, e.g., a *multi-cloud middleware*, is responsible for communicating among the providers involved.

The required agreements between cloud providers in federated-cloud model is a real barrier for the popularity of this model in the commercial world. Therefore, this model mainly has been implemented so far in the academic world where establishing an agreement between the cloud providers is easier [148]. In contrast, the multi-cloud model seems to be more attractive in the commercial world and this interest has been raised in the last five years [149]. It is mainly because this model is not intrusive from the perspectives of the cloud providers. Such a middleware exists now and it addresses the interoperability issue of this model by providing unique entry points for the various involved clouds [149].

SLA parameter	desired value
Contract duration	3 months
Budget for virtual machine	0.07 $ per hour
Budget for storage	0.1 $ per month
Budget for traffic	0.1 $ per month
Availability	99.9 % in 3 months
Response time	600 ms

Table 2.1: A sample SLA agreement in the cloud computing domain

Based on the mentioned benefits of the multi-cloud model over the federated-cloud model, and the growth in its popularity, in this thesis, we focus on a multi-cloud model by proposing a multi-cloud service selection approach.

2.1.2 Service Level Agreement

The service level agreement is not a new term since it has been used widely in the telecommunication and networking domains in order to specify the quality of service objectives [42, 186]. However, this term is now adopted in the computer science field for the same purpose of specifying quality of service for the services offered via Internet [57].

Cloud computing as a clear example of Internet-based services utilizes SLA in order to cover quality and cost aspects of this technology [36, 57].

In the context of cloud computing, a service level agreement is a contract between a cloud provider and a cloud customer that covers a clear description of the agreed service, quality of service parameters, service cost, and compensation actions in case of violating the agreement. In other words, the main idea behind using SLA is to provide a clear definition of the formal agreements about the non-functional aspects of the service such as QoS parameters, e.g., performance and availability, as well as the service billing [18].

Two types of SLA can be defined [22]: (i) off-the-shelf (non-negotiable) agreements; (ii) customized, negotiated agreements. While public clouds usually offer a non-negotiable SLA, such off-the-shelf agreements may not be acceptable for cloud customers who have critical data or applications to be deployed in the cloud. In the research carry out in the scope of this thesis, we consider SLAs as off-the-shelf agreements, so the cloud provider must satisfy them otherwise in case of violation, the customers can claim for the penalty.

Table 2.1 presents an example of the SLA agreement specifying some QoS parameters in the cloud computing context, such an SLA is used in the scope of this thesis.

2.1.3 Cloud Infrastructure Management

Utilizing large-scale data centers are a prerequisite for providing high quality cloud services [37]. Data centers for cloud computing are often called cloud data centers, but we use the term “data center“ or “cloud infrastructure“ for this aim in the scope of this thesis. A data center is a facility used to house computer systems and all the associated

components like telecommunications, storage, power supply, and cooling systems. To be able to accommodate all the cloud customer demands, the energy consumption of data centers is increasing rapidly. For example, large-scale data centers use as much electricity as a small town [66], and in overall accounting for 1.5% of global electricity usage [187]. Cloud infrastructure management is defined as the process of managing physical and virtual computing resources in a way to ensure that all the offered infrastructure services are working efficiently while avoiding the violation of customers' SLAs.

The key to efficient cloud infrastructure management is a solid configuration of virtual machine and their physical hosts [139]. To this end, an efficient VM placement can improve the utilization of available resources. VM placement is defined as mapping the requested VMs to the physical hosts so as to select the most suitable host (in terms of required resources) for each VM [167], it is often called VM allocation. Moreover, VM consolidation is defined as a mean for time-sharing the virtual resources between multiple users to minimize the energy consumption of the infrastructures by maximizing the number of inactive physical machines [164].

One of the common ways to realize consolidation in virtualized environments, is the live virtual machine migration which leads to efficient power management and load balancing across data centers. Live VM migration is the replacement of running VMs seamlessly across distinct physical hosts without any impact on VM availability from the end user's viewpoint [43, 179]. An efficient cloud infrastructure management can be realized through an effective VM placement (i.e., VM allocation) for each incoming VM demand followed by a dynamic VM consolidation, which triggers necessary VM migrations at runtime. In this thesis, a VM placement approach for efficient management of geographically distributed cloud data centers is presented.

2.1.4 Cloud Resource Elasticity

The term elasticity is widely used in the field of physics and economics, and it has been transferred to the context of cloud computing, as one of its core characteristics [150]. In the cloud computing domain, elasticity is defined as the ability of a system to dynamically adjust its main attributes in response to runtime changes such as varying workloads. Resource elasticity is introduced where the allocated resources are considered as the elastic attributes of the system [54]. In other words, resource elasticity is the ability of a cloud system to automatically provision and release computing resources on demand to accommodate dynamic workloads over time in order to meet the quality of service requirements. Two cloud resource elasticity strategies are defined, as shown in Figure 2.1:

- **Horizontal elasticity** is the ability to acquire or release virtual machines which host the application according to workload changes.
- **Vertical elasticity** is adjusting the capacity (e.g., allocated memory or CPU) of individual VMs hosting the application to quickly cope with runtime changes.

Horizontal elasticity is coarse-grained, as the units which are added or removed are fixed size VMs, and is relatively slow (in order of minutes) to be applied. It also needs

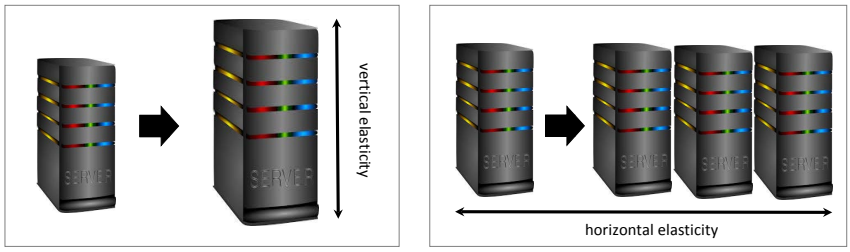


Figure 2.1: Horizontal elasticity vs. vertical elasticity

some application-level features, such as load balancing and state synchronization, to be supported. In contrast, vertical elasticity is fine-grained, as the units are in any arbitrary size such as a few MB of memory or a portion of a CPU core. Vertical elasticity is relatively fast (less than a second), and in spite of the need of hypervisor-level support, it requires only some basic application-level features such as multi-threading. In the scope of this thesis, vertical resource elasticity, is the primary focus of interest.

Note that in the cloud computing domain, elasticity covers both increasing and decreasing the capacity of the deployed environment, while scalability only addresses the increasing of the capacity. Moreover, the elasticity concept implies the live configuration at runtime where there is no need for restarting the application. In the scope of this thesis, when the resource elasticity term in used, it implicitly indicates live resource auto-scaling either *up or down* (i.e., vertical elasticity), or *in or out* (i.e., horizontal elasticity).

Mechanisms for vertical memory elasticity

Since the focus of this thesis is more on memory elasticity, in this section we elaborate hypervisor-level mechanisms that realize it. We explain two different mechanisms that are used in hypervisors to enforce memory actuation, and finally we clarify which of these mechanisms are used in our proposed solutions. In general, the hypervisor is responsible to provide users with application programming interfaces (APIs) for vertical elasticity. In the case of memory operation, it also requires some support from the virtual machine's kernel, hence two mechanisms are commonly mentioned:

- **Hot memory add or remove**. Adding or removing resources without having to reboot the system is called hot add or remove. Assuming a kernel supports *hot memory add or remove*, this concept can easily be extended to virtual environments: whenever the hypervisor wants to take memory from a virtual machine, it would request it through a VM-hypervisor interface, and the VM's kernel would be elastic with respect to memory. This mechanism is not widely used since it cannot be supported by the guest operating systems without restarting the VM.
- **Memory ballooning**. In this mechanism, instead of adding or removing memory, the VM's kernel can ban the usage of a portion of memory in spite of the fact that initially it was allocated to the VM. This is achieved by running a custom

device driver, the so-called *ballooning driver*, in the VM's kernel, which creates a bridge between the hypervisor and the VM. Using this mechanism, the VM's kernel is booted with a certain amount of memory. Initially, the balloon would be *deflated*, i.e., the ballooning driver would request no memory from the VM's kernel. Hence, the VM could use all the initial memory. If the hypervisor wants to reduce the memory allocation of the VM, then it would tell the balloon to inflate to that amount. When the balloon expands, the physical memory available in the VM is reduced that compels the guest operating system to reduce the memory footprint of other processes when insufficient free memory is detected; for instance, via passing some of the processes' memory pages to the swap space, or killing some of them in extreme situations. Then, the memory allocated by the balloon process in the guest OS can be reclaimed by the host OS, and can then be used by other co-located VMs, enabling a higher consolidation ratio on the physical host [169].

Finally, if the hypervisor decides to increase the memory allocated to the VM, it would map that amount to the VM address space, in the region allocated by the balloon driver. Now the balloon driver has access to that memory, and can safely release it to the VM's kernel. Despite its complexity, this mechanism reacts almost instantaneously, and the guest OS reflects the memory change a few moments after the operation is executed through the hypervisors' APIs [134].

Based on the above explanation, in contrast to *hot memory add or remove*, *memory ballooning* has some restrictions in order to support memory elasticity: (i) a maximum amount of memory needs to be specified; (ii) some ballooning drivers can only deflate as much as they had been previously inflated. However, in the case of *memory ballooning*, it is supported by all recent Linux kernels and no additional features are required. This makes *memory ballooning* a practical mechanism for realizing vertical memory elasticity, as it is supported by both Xen and KVM hypervisors, while *hot memory add or remove* is currently not supported by any guest OSs without restarting the VM [169]. Hence, in our work, we utilize *memory ballooning* mechanism through the reconfiguration APIs provided by the Xen and KVM hypervisor.

2.2 Scientific Models, Theories, and Methods used in this Thesis

In this section, we briefly explain the well-established models, theories, and concepts used in the proposed contributions of this thesis. They are either concepts from the computer science domain, such as Bayesian networks, model driven architecture, and autonomic computing; or from the interdisciplinary domains, such as control theory (mostly used in mechanical engineering), and prospect theory (a well-known theory in economics).

2.2.1 Bayesian Networks

In what it follows, we first introduce Bayesian networks (BNs) covering the definition, the structure, and their benefits. Then, in order to better exemplify their applications in

the cloud computing domain, we explain the application of a sample Bayesian network in solving a small-scale cloud computing problem.

Definition. Bayesian networks have emerged as a practical form of knowledge representation. A BN is a graphical model to represent a variety of interests such as event occurrences and the probability between them via a direct acyclic graph (DAG) [137]. In a DAG, nodes are random variables and edges show conditional dependencies among the nodes; it means nodes that are not connected represent variables that are conditionally independent of each other. Each node is associated with a probability function that takes a particular set of values of the node's parent variables as input and gives the probability of the variable represented by the node. BNs can be modeled both via graphical representations, i.e., qualitative models, and probability values which indicate the relationship strength of each edge, i.e., quantitative models. The most common task which can be solved using Bayesian networks is probabilistic inference [135]. As an example, consider a simple network with three nodes: Web application response time, user workload, and physical host failure. Suppose we observe the fact that the application response time is increasing. There are two possible causes for this: either the workload is increasing, or the physical host faced a failure. By designing a Bayesian network for this simple example, and feeding it with the historical data, we can further answer such questions: which node has a higher probability to be the reason of the performance degradation; or computing the probability that the application response time will be high given that the application workload has been increased. General questions that can be answered by using a BN are: what are the most probable hypotheses for the set of training data? Or what is the most probable category of an observed data?

Structure. The structure of a DAG is defined by two sets: a set of nodes, and a set of direct edges. The nodes which represent random variables are drawn as circles labeled with the variable names, similar to nodes shown in Figure 2.2. As depicted, the edges are drawn as arrows between nodes and represent the direct dependency among them. In particular, an edge from node X_i to node X_j represents a statistical dependency between the corresponding variables. Thus, the arrow indicates that a value taken by variable X_j depends on the value taken by variable X_i, while node X_i is a parent of X_j.

In a BN the conditional probability is represented by a conditional probability table (CPT) for each node, listing the local conditional probability that a child node takes for each combination of values of its parents, where each cell contains the calculated conditional probabilities, similar to the CPT shown in Figure 2.2 for node *Cost limit exceeded*. If there are no parents for X_i, its local probability distribution is considered unconditional, similar to the CPT shown in Figure 2.2 for node *Change in energy tariffs*. If the variable represented by a node is observed, then the node is said to be an evidence node, otherwise the node is said to be hidden or latent [152].

Assume $U = \{V_1, ..., V_n\}$ which represents all random variables in a system which is modeled by a BN. The goal of reasoning under uncertainty is to calculate the conditional probability of a variable V_i in one of its states given the status of a set of other variables $\{V_1, ..., V_k\}$, where $(\{V_1, ..., V_k\} \subset U) \bigwedge (V_i \notin \{V_1, ..., V_k\})$. This conditional probability is

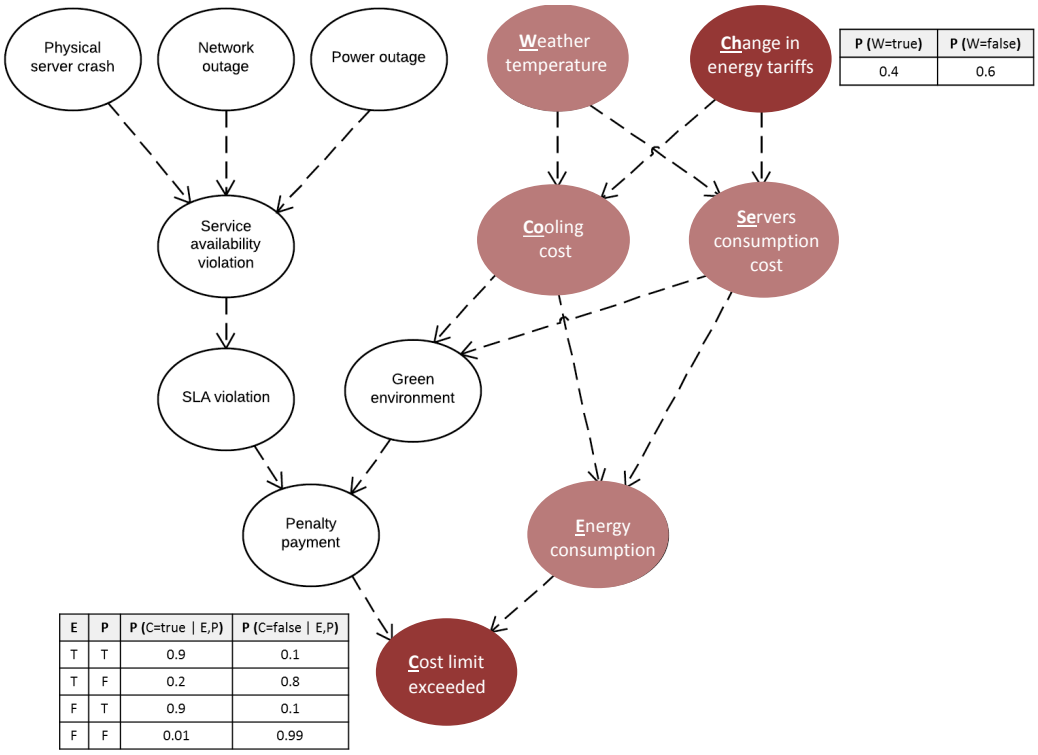


Figure 2.2: A sample Bayesian network model for a cloud computing scenario

formally specified as Equation (2.1).

$$P(V_i \mid \{V_1, ..., V_k\}) = \frac{P(V_i, V_1, ..., V_k)}{P(V_1, ..., V_k)} \tag{2.1}$$

In probability theory, both $P(V_i, V_1, ..., V_k)$ and $P(V_1, ..., V_k)$ can be calculated if the full set of joint probability distributions $P(U)$ is known. However, for large and complex systems, determining $P(U)$ is a computationally expensive process. Nevertheless, in a BN each node is conditionally independence of its ancestors given the values of its parents, as depicted in Equation (2.2).

$$P(node|\, ancestors) = P(node|\, parents) \tag{2.2}$$

Therefore, by considering Equations (2.1) and (2.2) and having a BN modeled based on the variable set of $U = \{V_1, ..., V_n\}$, the joint probability distribution $P(U)$ is defined as Equation (2.3), which is named as *chain rule*.

$$P(U) = P(V_1, ..., V_n) = \prod P(V_i|\, parents\ \ of\ \ (V_i)) \tag{2.3}$$

Application of Bayesian networks. Bayesian networks are powerful tools for deep understanding of very complex, high-dimensional problem domains which deal with

uncertainty, i.e., where the correlation between variables cannot be clearly observed [175]. Since a BN can simulate the mechanism of exploring the causal relationship among various factors, it can facilitate the prediction and cognitive activities via causal reasoning [151]. There are various areas in which Bayesian networks can be used, such as machine learning, text mining, natural language processing, speech recognition, signal processing, Bioinformatics, medical diagnosis, weather forecasting, cellular networks, root cause analysis, risk management, system reliability analysis [33, 152]. The main benefit of a BN model is simulating the mechanism of exploring causal relations between key factors using Bayes' theorem [100]. Bayes' theorem is a simple mathematical formula used for calculating conditional probabilities. In other words, this theorem explains the probability of an event based on the conditions related to the event.

In comparison with other modeling techniques, BN has the following benefits [109, 81]: (i) using probabilistic rather than deterministic expressions to describe the relationships; (ii) the ability to deal with systems where uncertainty is inherent; (iii) facilitating learning of causal relations among variables; (iv) the ability to be adjusted for new knowledge; (v) the ability to learn a model based on observations. Therefore, in the scope of this thesis, Bayesian networks are used for the solutions proposed in cloud infrastructure management.

The steps for using a Bayesian network

The usage of a Bayesian network can be done through the following steps [193]:

1. Transforming the problem statement into a Bayesian network. This step includes identifying the root variable nodes (the nodes with no parents). Then, determining the inputs and outputs of all other individual variable nodes. This can shape the causal relationship that the variable nodes can have with each other; A summary of the activities in this step is as follows [193, 88]: (i) identifying the goals of modeling (e.g., prediction, explanation, exploration); (ii) identifying the observations that may be relevant to the identified problem; (iii) determining which subset of those observations is worthwhile to be modeled; (iv) organizing the observations into variables having mutually exclusive and collectively exhaustive states.
2. Configuring the parameters of the Bayesian network. In this step the *prior probability* of each node based on the past values of the variable, should be obtained. Moreover, for each node a conditional probability table should be extracted by using the historical data gathered from the domain experts. The values of a CPT can be either deterministic (i.e., 0 1) or probability (e.g., 0.95).
3. Reasoning with Bayesian networks. Two types of reasoning are possible with a Bayesian network [152]: (i) causal (top-down) inference. Such usage of Bayesian networks is often named *generative* model, since they specify how causes generate effects [135]; (ii) diagnostic (bottom-up) inference, where reasoning is about cause based on the evidence.

   To find the cause, the node with the highest value, the posterior probability of cause node is calculated by considering the evidence set of corresponding nodes, and

finally the *maximum a posterior* approach [47] is applied. In practice, a Bayesian network is a useful model for root cause analysis and decision support [153, 182].

In the remaining of this section, in order to exemplify the usage of Bayesian networks in the cloud computing domain, we apply the above mentioned steps to design a Bayesian network for root cause analysis of QoS degradation in the scope of cloud infrastructure management, depicted in Figure 2.2.

Using Bayesian networks in a sample cloud computing scenario

Every cloud provider has certain goals such as providing services with high QoS, and minimizing the cloud operation cost. At runtime when QoS is decreasing, the provider needs to quickly find the cause and solve it. However, the relationships between the provider's goals and the factors that influence these goals are non-deterministic. In such situations, the probability of possible influencing factors can be helpful in detecting the cause of the problem. As a sample goal for a cloud provider, assume the provider aims to keep the operating costs under a certain level for a certain period.

There are many factors that directly or indirectly influence the cloud operating costs. The relationships between increasing costs and some of these factors are non-deterministic and may only be defined by their probabilities. In the following, we explain how to utilize Bayesian networks to detect the cause of the violating for a cloud infrastructure provider. Note that, as previously mentioned, there are two types of reasoning in Bayesian networks top-down and bottom-up reasoning. In this problem, the bottom-up reasoning is leveraged which is for diagnostic problems, i.e., reasoning about causes based on the evidences.

We can model the influencing factors and the predefined cloud provider goals as the nodes of a DAG, as shown in Figure 2.2. The aim of this modeling is *analyzing* the cause of the cost violation. We simply extract all possible reasons that influence the operating costs from the perspective of a cloud provider. Then we draw edges from cause variables to their immediate effects as their children. The causal relationships between the goals and the associated influencing factors are modeled as a child-parent relationship. In order to make a probabilistic inference to detect the cause of the operating costs violation, we leverage Equation (2.3) to compute the posterior probability of each cause and determine the most probable root of the problem. In the following, we calculate the posterior probability of each node by considering the evidence set from the corresponding observation node, such as *Cost limit exceeded* node in Figure 2.2) and then based on maximum a posterior strategy, the node with the highest value is the most probable responsible cause of the cost violation.

Based on the designed Bayesian network, Figure 2.2, and the introduced Equations (2.1) to (2.3), we can find the answers of questions such as "what is the probability of *change in energy tariffs*, given the *cost limit exceeded?*". Notice, as shown in Figure 2.2, these two nodes in the graph are colored in dark red and influencing nodes are colored in light red, plus the initial letters of each node are used in Equations (2.4) and (2.5). It is worth mentioning that for each node the only influencing probability is the value of the

probability of its parents.

$$\begin{cases} P(Ch|C = true) \Rightarrow \\ P(Ch|C = true, E) \Rightarrow \\ P(Ch|C = true, E, Co, Se) \Rightarrow \\ P(Ch|C = true, E, Co, Se, W) \Rightarrow \end{cases} \tag{2.4}$$

and then by using Equation (2.1) we have:

$$\Rightarrow \frac{P(Ch, C = true, E, Co, Se, W)}{P(C = true, E, Co, Se, W)} \tag{2.5}$$

Now by using Equation (2.3) and the values of CPT of each node, we can calculate the probability.

2.2.2 Multiple-Criteria Decision Analysis

A multiple-criteria decision analysis (MCDA) is concerned with structuring and solving decision making problems where multiple criteria are involved. There is no longer a unique optimal solution to an MCDA problem that can be obtained without incorporating preference information. Therefore, it is necessary to take the preferences of the decision makers to differentiate between solutions and score them in order to choose the best solution, i.e., the most preferred alternative of a decision maker. There are two main classifications of MCDA problems depending on whether the solutions are explicitly or implicitly defined [177]:

- **Multiple-criteria design problems**. In this category, the alternatives are not explicitly known. A solution can be found in this case by solving a mathematical model. In such problems, the number of alternatives is typically either infinite, e.g., in case of continuous variables, or very large, e.g., in case of discrete variables.
- **Multiple-criteria evaluation problems**. This category of problems consists of a finite number of alternatives that are explicitly known in the beginning of the solution process. In this category, the problem can be defined as finding the best alternative or a set of good alternatives for the decision maker. This type of MCDA problem is the focus of interest in the scope of this thesis.

To solve MCDA problems in either categories, preference information of the decision makers is required. While Bayesian network can effectively be used to make decisions under uncertainty, they cannot deal with multiple criteria decision making problems. Therefore, along with using Bayesian networks, MCDA technique is applied to help multi-criteria decision making under uncertainty in the scope of this thesis.

2.2.3 Model Driven Architecture

Model driven architecture (MDA) [141] is a software design approach based on models, launched by the object oriented group (OMG)[1] in 2001. It provides a set of guidelines

[1]Object oriented group: `http://www.omg.org`

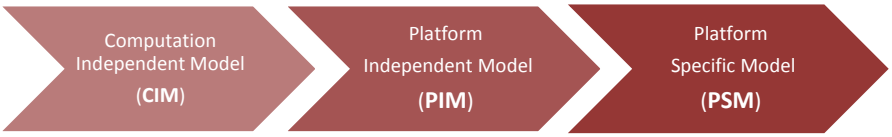


Figure 2.3: The basic process of model driven architecture [87]

to structure the specifications that are expressed as models. MDA introduces certain types of models as well as the relationships among them. The main concept of MDA is separating the system operation from the details provided by using a target platform. It defines three different viewpoints on a system which are mapped to the following three models [87]:

- **Computation independent model** (CIM). A CIM, or often called a domain model, represents the system from a computation independent viewpoint, where the focus is on the requirements of the system in a specific domain without any details about the system structure and processing.
- **Platform independent model** (PIM). A PIM reflects the system from the platform independent viewpoint, where the operation of a system is concerned without the details of a particular platform. In other words, this model covers the system specification that does not change from a platform to another, i.e., the implementation details are hidden.
- **Platform specific model** (PSM). A PSM shows a view of the system from the platform specific viewpoint, where the details of the used platform are added. This model covers the details about the way the target platform is used by the system.

As shown in Figure 2.3, the basic process of using MDA for building a software system, starts with defining the CIM for the target domain by a business analyst. Then, this model is enriched and transferred into a PIM by an architecture who adds the architectural details of the system without showing the details of the target platform. Finally, the platform specialist completes the PIM by adding all the implementation details needed for the system to operate. In general, knowledge is added by different system professionals to each model so that it is transferred to another model at each step [87]. The introducing MDA concept and its corresponding models are used in the solutions proposed in the scope of this thesis.

2.2.4 Prospect Theory

Prospect theory [102] is a behavioral economic theory developed by Daniel Kahneman and Amos Tversky in 1979. Daniel Kahneman won a Nobel Prize (2002) in economics for his work on this theory. Prospect theory is a descriptive model for decision making under uncertainty based on the potential value of losses and gains rather than the final outcome. It is an alternative and psychologically more accurate decision making model for utility theory and it is more realistic in calculating the user satisfaction [102, 174].

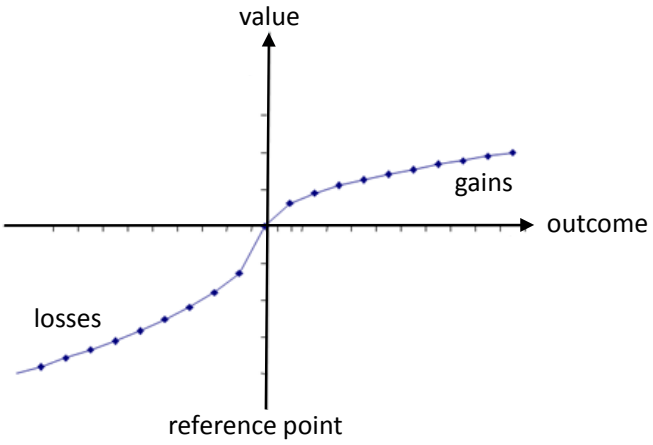


Figure 2.4: A sample value function based on prospect theory [102]

It uses the concept of *value* instead of *utility*, where *utility* is typically defined only in terms of net wealth, while *value* is defined in terms of gains and losses. Based on prospect theory, probabilities are replaced by decision weights [142]. Figure 2.4 shows a sample value function under prospect theory, which is s-shape and asymmetrical and defined based on the deviations from a reference point. As depicted, this value function is concave for gains (implying risk aversion) and is convex for losses. It is also steeper for losses than for gains (loss aversion) [142]. In the scope of this thesis, the user satisfaction for a specific cloud service is modeled using prospect theory.

2.2.5 Autonomic Computing and Control Theory

The term autonomic computing first was proposed by IBM [91] in 2001 for describing computing systems that are self-management [105]. In such systems, humans do not control the system and only define the general goals and rules as the inputs [57]. A self-management system constantly adapts itself to accommodate runtime changes like varying workloads, or software failures [105].

To achieve self-managing system by utilizing autonomic computing, a closed control loop is suggested as a reference model [92], where an autonomic manager, also called autonomic controller, controls the states and behaviors of the system. In this reference model, an autonomic system is implemented following a MAPE loop [91] including four main steps: monitoring, analyzing, planning, and execution. The autonomic controller monitors the system to detect the changes (M), analyzes their impacts (A) and if needed plans based on them (P) to execute suitable control actions on the system in response to the changes (E).

The introduced autonomic computing reference model (i.e., the MAPE loop) is a *software engineering perspective* toward realizing self-adaptive systems. From this viewpoint, a software system is adaptive when it includes features to adapt its structure or behavior at runtime, without interrupting its service. If an adaptive system is coupled

with an autonomic controller, which continuously satisfies the system requirements in spite of runtime changes, creates the system is called self-adaptive [70].

Control theory as an interdisciplinary branch of engineering and mathematics deals with the behavior of systems, and how their behaviors are modified by feedback loops. Control systems have been widely used in many engineering domains [68]. From the *control engineering perspective*, an adaptive system consists of a closed control loop [26, 89] with a controller and a plant that is being controlled. The controller is designed to follow a control signal (reference) by monitoring the output and comparing it with the reference. The controller periodically configures the controlled plant by adjusting a control knob taking a quantitative feedback. This feedback is the actually the difference between the actual output and the reference named as control error. The main goal of the controller is to bring the actual output closer to the reference.

Control theory is emerging as an approach for the design of self-management software systems [70, 68, 104], where an adaptive software system is defined as the plant. Applying the theory of controlling industrial plants (i.e., control theory) on the software engineering domain to design self-adaptive software systems can enhance the software engineering process with a variety of mathematically grounded adaptation formulas [70]. In this thesis the control theoretical techniques are applied in the cloud computing domain as a sub-domain of the software engineering to make cloud application self-adaptive.

Elasticity controller. In the context of cloud resource elasticity, previously introduced, we can consider a so called elasticity controller that continuously monitors the QoS attributes of a cloud application, and in the case of necessity applies either horizontal or vertical elasticity strategies at runtime to meet its QoS objectives. In this thesis, a self-adaptive cloud application is realized by coupling it with an elasticity controller.

2.3 Self-Adaptation Process for Cloud Applications

Current cloud elasticity features provided for cloud applications rely on the knowledge of the applications' owners as cloud customers in configuring the elasticity parameters, but achieving smooth elasticity is intrinsically hard to be performed by customers. In order to overcome this dependency, using approaches from autonomic computing is shown to be appropriate. Control theory, as previously introduced, proposes a systematic way to design feedback control loops to handle unpredictable changes at runtime for software applications. Although there are still substantial open issues to effectively utilize feedback control in self-adaptation of software systems, software engineering and control engineering communities have made recent progress to consolidate their differences by identifying challenging points that can be addressed cooperatively. This section is in the same vein, but in a narrower domain given that cloud computing is a sub-domain of software engineering. More specifically, although feedback control is a powerful approach to construct any adaptive software systems, in this section our focus, as a member of cloud community, is on self-adaptive cloud applications. We highlight the aspects that are important in the self-adaptation process of cloud applications from the perspective of control engineers.

2.3.1 The Trend of Self-Adaptive Software Systems

There is some research on self-adaptive solutions for software systems which take the software engineering point of view [50, 41]. Moreover, there is a new trend of applying the control theory in software systems, taking the point of view of control engineering [67, 126]. Recently, a Dagstuhl seminar[2] [69, 70] gathered two communities of software engineering and control theory to develop their cooperation for devising new modeling strategies to empower software engineers with theoretical and practical skills of control engineers and bring control to the core of adaptation. As a result of the last seminar, in [70] they present a general control design process for software systems which enables automatic (i) analysis and (ii) synthesis of a controller that is guaranteed to have the desired properties and behavior. However, the research regarding the application of control theory to enable self-adaptation in software engineering, despite its recent progress, is still in a very early stage [67].

Considering cloud computing as a sub-domain of software engineering, there are also some research attempts which focus more on the cloud scenarios [106, 55, 108, 146]. In this trend, *cloud control*, as a new research area, is proposed [106], to apply control theoretic approaches in a range of cloud management problems, such as managing resource-optimized cloud data centers. Their idea leads to establishing a series of scientific meeting called *cloud control workshops*[3] aim to foster research in the area of cloud computing and control theory. Current existing research on combining cloud computing and control theory have predominantly the perspective of cloud providers, hence their focus is more on controlling the cloud data centers. Whereas we look at the main aspects from the cloud application's point of view. In particular, as discussed, there are only a few research attempts [146, 106] that address the challenges of self-adaptive cloud applications, so there are still open issues which have not been thoroughly investigated.

By combining cloud computing, modern software systems, and control theory, the ultimate objective is to turn cloud applications into self-adaptive systems which are performance-aware, robust, flexible, and resource- and cost-efficient. The aim of the remaining of this section is along with the lines of *cloud control* research area proposed in [106], but with a more pronounced application perspective. We bring up a range of important aspects which need attention on the way of realizing self-adaptation for cloud applications. In the following, we first present an overview of self-adaptation cloud applications, then we highlight various main aspects of this process.

2.3.2 Self-Adaptive Cloud Applications

The key aspect of an elastic software is its capability to autonomic ally adapt at runtime (i.e., self-adapt) in response to changes in the operating conditions, such as fluctuations in workload, by automatically stretching and shrinking the resources. Cloud elastic software systems are the most common realization of elastic software systems. This category of software systems exploits the ability of cloud environments to acquire and

[2]Dagstuhl seminars `https://www.dagstuhl.de/programm/dagstuhl-seminare`
[3]Cloud control workshop series: `http://cloudresearch.org/workshops`

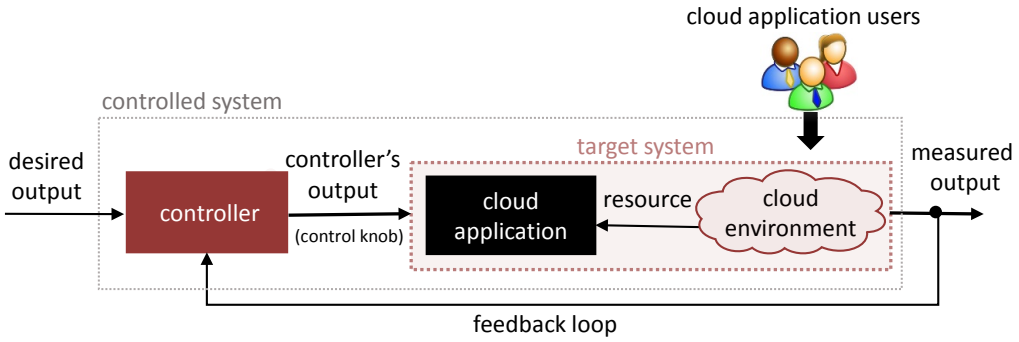


Figure 2.5: A realization of self-adaptive cloud applications via feedback control loop

release resources while serving end users. For instance, when the usage of the system increases, the allocated resources may saturate and in order to avoid degradation of QoS, the elastic system allocates more resources to rectify the situation. Once the incoming workload diminishes and the allocated resources become under-utilized, the elastic system consolidates the load for a portion of resources and releases unused resources to reduce the costs. In this interpretation, elasticity is a feature or a means to avoid under- or over-provisioning and allows elastic software to service end users with acceptable QoS while minimizing the operational costs. In the cloud citation, as previously mentioned, horizontal and vertical are defined as two elasticity strategies.

In the context of control theory, a standard feedback control loop roughly looks as shown in Figure 2.5. The system which is being controlled is labeled target system and the combination of the controller and the target system is labeled controlled system. There is a desired output that needs to be achieved by tuning the controller's output, which can be one or more configurable parameters of the target system. The controller periodically adjusts the value of the controller's output (often named control knob) at runtime in such a way to ensure continued satisfaction of the desired output despite the runtime change.

Self-adaptive cloud software applications can be realized via a feedback control loop architecture. Figure 2.5 depicts a reference architecture, where a controller supervises a software application. The target system consists of an application deployed in a cloud environment (i.e., cloud application). The desired output is one or more QoS attributes, such as application response time, which are monitored periodically as the measured output. The workloads for a cloud application are changing unpredictably at runtime. Since the controller cannot control the workload, it should apply corrective actions and change the cloud environment in a way to meet the desired QoS.

By taking the resource elasticity as a control knob, the controller implements a logic that adjusts the resources consumed by the application to accommodate workloads. The controller monitors the operational condition of the cloud application at runtime. Based on the output, the controller instantiates new virtual machines or terminates existing ones–horizontal elasticity, or adjusts the capacity of individual active VMs hosting the application–vertical elasticity, to cope with runtime changes on demand. Finally, the

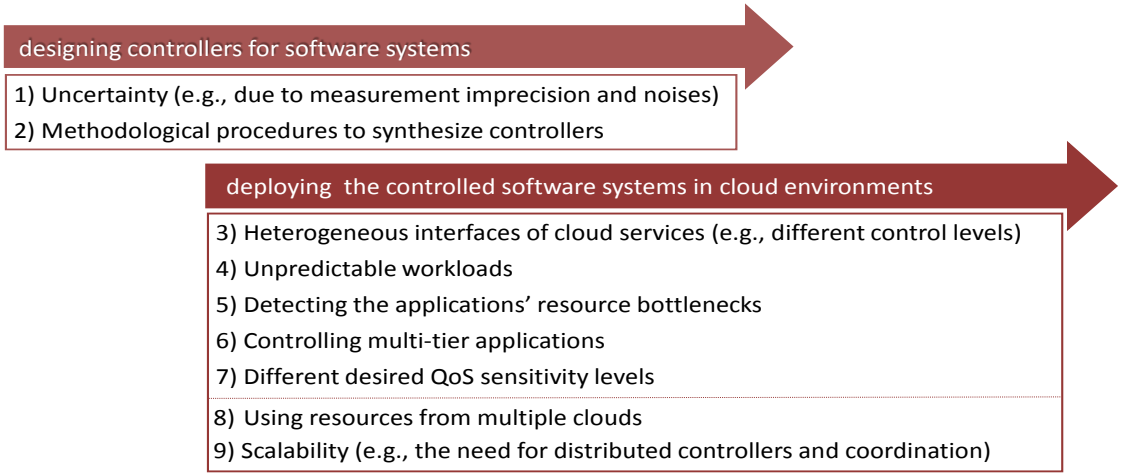


Figure 2.6: Summary of the main aspects of self-adapting cloud applications

cloud provider calculates the total usage and bills the application owner for the cost of leasing cloud resources. In order to design and maintain an effective feedback control loop for cloud applications, there are several parts of the process which need attention and they are discussed in the next section.

2.3.3 Main Aspects of Self-Adapting Cloud Applications

In this section we scratch the surface by exploring the most important aspects in the process of making a cloud application self-adaptive and briefly propose some hints to pave the way for each aspect. As shown in Figure 2.6, we start with the relevant aspects in the process of designing a controller for a software system in general, then extend them by bring up particular aspects in the process of deploying the application in the cloud. Finally more long-term aspects are presented that need attention due to the cloud computing trend.

Uncertainty

Designing elasticity mechanisms poses complexity challenges because of uncertainty [75, 122, 96, 95] that is likely to be present in every facet of elasticity reasoning. For instance, users often find it difficult to accurately describe elasticity policies, or knowledge used for elasticity reasoning may not be accurate. Moreover, in order to make decisions about corrective control actions, monitoring tools provide input data for elasticity decision making. These measurements are not usually free of noise and contain random and persistent disturbances that can affect the clarity of a given property, especially in cloud environments. They may also contain irrelevant or meaningless data. This affects elastic systems in a way that they are not able to replicate a given measurement consistently throughout a control period. If these potential sources of uncertainty are not explicitly

taken into account in elasticity reasoning, they affect runtime scaling decisions which are often unreliable.

Theoretically, elasticity should accommodate even unexpected changes in capacity, adding resources when needed and reducing them during periods of low demand, but the decisions to adjust capacity must be made automatically and accurately to be cost effective. If elasticity decisions are made without considering uncertainty, then available resources may not be sufficient or cost-effective at a certain point in time. Several approaches are used in practice to cope with uncertainty, e.g., in software engineering [77] or self-adaptive software [59]. However, as discussed in [96, 122], uncertainty in the context of dynamic resource provisioning for cloud application [75] is still unclear.

Developing methodological procedures to synthesize controllers

Cloud computing is not a deployment environment to which existing software solutions can be transferred easily. Instead, it offers novel characteristics not existing in traditional deployment environments such as seemingly endless resource pool [80]. Therefore, the advantages of using cloud as a deployment environment for a software systems are leveraging such characteristics. For instance, cloud elasticity can be used to provide consistent performance while minimizing resource cost for application owners. As previously discussed, horizontal and vertical elasticity as two possible cloud elasticity strategies in clouds, have their own pros and cons to be adopted as control knobs. Hence, these strategies should be used in accordance to the application requirements at runtime as possible control actions. From the cloud provider's perspective, the details of the applications that they host are basically black-box and not visible. This makes it difficult to accurately devise an optimal set of corrective actions (i.e., adopting a proper elasticity controller at runtime or defining thresholds for elasticity mechanisms). Thus, the burden of such tasks falls on the application owner as a cloud customer [76], which does not have deep knowledge about the application workloads, cloud environment characteristics, and performance modeling.

To address this aspect, the control community can provide certain generic methodological solutions to facilitate the design of controllers for software systems and consequently cloud applications. A solution in which the application owner is only required to define a desired level of QoS attributes and put the decision making responsibility on an autonomic controller at runtime. As a recent and promising research work, Filieri et al. [68, 70] propose a generic and yet practical methodology to synthesis controllers for software systems. The main benefit of this methodology is to reduce the need for a strong mathematical background as a software engineer to devise ad-hoc control solutions. Based on this, having chosen a target system, one only needs to indicate a controller's output, which can change the behavior of the target system as well as specifying a desired output to be achieved by the controller. This methodology as a control design process is used in the scope of this thesis.

Several aspects should be carefully considered while designing and maintaining elasticity controllers are as follows: (i) determining when a resource is insufficient; (ii) quantifying requirements according to application environment; (iii) identifying when

and how much of resource can be added or removed without degrading the application performance; (iv) finding a safe adjustment granularity at runtime as the reaction of the application deployed in the cloud for the applied controller's outputs is not deterministic.

Heterogeneous interfaces of cloud services

A cloud application can be deployed either in an IaaS or PaaS. As previously illustrated, these two delivery models provide different control levels of the environment which hosts the application such as the interface for monitoring and the interface for a reactive control knob. For instance, while the amount of resources (e.g., memory or CPU) can be adjusted at runtime using an infrastructure service, such levels of control on resources are not yet possible for a PaaS. As a result, from a control perspective, applying certain control actions or monitoring some QoS attributes might not be possible in some cloud environments, so the interface between cloud applications and cloud services is an aspect that needs attention while designing controllers. Therefore, both interfaces must be designed cooperatively by taking into account the control level of the deployed environment which are efficiently supportable by the cloud provider. Note that in the scope of this thesis our focus in on cloud infrastructure services.

Unpredictable workloads

Typically, a variety of different application types can face different workloads, or even for a certain cloud application, different users usually have different usage patterns [19]. Self-adaptation of such applications can be realized by using controllers that dynamically tune the amount of allocated resources. The change of the resources should be according to the changing workload at runtime. Such changes are sometimes very sudden and unpredictable with sporadic runtime peaks. Since controlling the workload is unrealistic, classification and using workload analyzing tools can improve workload predictions. This way, it is possible to further synthesize controllers that can deal with a specific category of workload at runtime, more effectively. This knowledge is also beneficial at runtime for dynamically adopting a set of controllers to cope with various situations.

Ali-Eldin et al. [19] address this research challenge by proposing a workload classification tool to analyze the application workloads, and assign them to the most suitable elasticity controllers. In a more generic view, in order to have an effective adaptive solution for cloud applications, selection of a controller among a set of synthesized controllers based on runtime situations (e.g., workload) is inevitable. For instance, during the runtime, different vertical elasticity controllers or horizontal elasticity controller can be adopted for a cloud application. Therefore, investigating on solutions in which dynamic switching among various controllers are doable at runtime is an important aspect in designing self-adaptive cloud applications.

Detecting applications' resource bottlenecks

The host cloud environment should be able to provide resources which are critical for the application at runtime. However, in spite of the importance of identifying the nature of

the application and its resource bottleneck before deploying it in cloud environments, application owners do not pay attention to this issue while choosing a cloud environment. Without enough knowledge of what is the application bottleneck, designing corrective actions (control knob) is impossible. Different resources can be the main reasons of QoS degradation for an application at runtime. For instance, an application can be CPU-intensive, memory-intensive, IO-intensive, or a combination of them.

An elasticity controller should be able to adjust the allocated amount of such resources. To this aim, (i) bottleneck detection should be applied on an application before synthesizing the controller; (ii) application should be deployed in a cloud environment which can provide elasticity and control permission on the detected resources. Although, utilizing methods such as the "trigger-less black-box bottleneck detector" presented in [183] or familiarity with the potential cloud application categories [132] are possible solutions, software engineering community can still provide clearer guidelines or more effective tools to facilitate this process for cloud community.

Controlling multi-tier applications

The pervasive and popular architectural patterns for a cloud application is the 3-tier pattern [80]. It comprises presentation tier (representing user interface (UI)), business logic (BL) tier (featuring the main application computation), and data storage (DS) tier (storing and managing the persistent data). Realizing self-adaptation of a multi-tier application deployed in cloud environments acquires research attention. In a multi-tier application, every tier can be the main reason of QoS degradation in a specific period of time; therefore, a possible solution can be adopting separated controllers for each tier and then use coordination methods such as message passing techniques among these tiers to make them isolate and avoid cascading effects. Each controller can pass the monitored data of its own tier as part of the input for controllers at other tiers.

Different desired QoS sensitivity levels

In a cloud application, different users may have different priority classes, such as Gold, Silver and Bronze [97]. Each class can define various sensitivity levels for the desired QoS attributes such as performance. Therefore, a solid self-adaptive solution for a cloud application should be able to make satisfaction of different classes of application users. In [31], Bayuh Lakew et al. propose a performance-based service differentiation where in case of overload, a service differentiation schema dynamically decides which services to degrade and to what extent. In case where enough capacity is available, each service is automatically allocated by the right amount of capacity that meets its target performance.

Using resources from multiple clouds

The traditional approach of using a single cloud as the only deployment environment for an application has several limitations in terms of QoS, vendor lock-in, unoptimized renting cost for world-wide users [148]. Therefore, as previously mentioned, using resources from multiple clouds has envisioned as a future trend for cloud community. Since in a

multi-cloud model dependent tiers of a single application can be deployed across multiple clouds in a distributed manner, making such an application self-adaptive is even more complex. Hence, enlightening solutions for interoperability and distributed controllers is getting necessary. As a recent work, Copil et al. [46] propose control mechanisms to address the elasticity of a multiple cloud deployment model.

Scalability

On one hand, software applications tend to be more large-scale and distributed; therefore, cloud environments are the most suitable environment to host such distributed applications. On the other hand, centralized control of a large-scale distributed system is seldom feasible. A solution which proposes a hierarchical control and leverages distributed controllers seems practical. However, this causes a problem of co-existence and possible inconsistencies and interference between controllers. Hence, coordination is recognized as an important aspect, not completely solved by existing research [51, 84, 162], and requiring special attention [83], which is in the scope of this thesis, too.

CHAPTER 3

VM Placement across Distributed Cloud Infrastructures

In recent years, cloud providers have been seeking for solutions to enable them to provide highly available and scalable cloud services to stand out in the competitive market of various cloud services. The difficulty is that to provide such high quality services, they need to manage large-scale and geographically distributed cloud infrastructure. While the energy consumption and consequently the operating costs of such infrastructures (i.e., cloud data centers) has been turned into a global problem [123]. Hence, leveraging cost-aware solutions to manage resources is necessary for cloud providers to decrease the total energy consumption, while keeping their customers satisfied with providing their expected QoS. An effective cloud management solution for geographically distributed cloud infrastructure should make decisions while taking into account various time- and location-dependent factors. These factors can be internal factors such as dynamic and unpredictable cloud resource demands, or external such as regional power-outages, temperature, regional electricity prices, and the ability to use different cooling modes. All of these factors can influence the decision under some levels of uncertainty.

In this chapter, we address research question I by explaining the details of contribution I, specified in Chapter 1. We propose a new approach to assist the cloud infrastructure provider in order to reduce the cloud operating costs by applying a novel cost- and QoS-aware virtual machine placement approach that is applied across geographically distributed data centers. We model the VM placement problem along with its related factors as a Bayesian network and then apply the multi-criteria decision analysis method on it. The proposed solution includes two algorithms for the virtual machine allocation and consolidation.

3.1 Motivation

The main goal of a cloud infrastructure provider is to minimize the operating costs of running data centers while meeting SLAs of the customers. Cloud providers tend to distribute their data centers all over the world in order to cover specific customer requirements and improve the performance of their services. However, for supporting the virtual machine placement across geographically distributed data centers, cloud providers need to consider several aspects in order to achieve a cost-aware solution such as:

- Each region has its own electricity market that directly affects energy cost. Global electricity price comparison [4] shows quite big price differences that can dynamically change in various countries. Moreover, due to the different temperatures in each region, temperature-aware management of distributed infrastructure can greatly reduce the energy cost, especially the cooling cost. More precisely, data centers located in cold regions have smaller partial power usage effectiveness (pPUE) rate [188], i.e., consume less energy to cool their infrastructures.

- Power outages can cause big issues for a cloud provider in terms of SLA violation. Statistics of electrical outages [14] reports the countries with frequent power outages in spite of a low energy price. Hence, it might be impossible to guarantee some QoS requirements such as availability in such regions.

- Decision making regarding the live VM migration, as one of the common ways for realizing efficient power management and load balancing across data centers, needs to consider the influence of the following factors on the migration period: (i) VM random-access memory (RAM) size, amount of data that should be transferred via network; (ii) bandwidth of the migration link, the higher speed links the faster data is transferred and the lower time is consumed to complete a VM migration to a destination physical machine (PM); (iii) dirty page rate (DPR), the rate at which memory pages are modified. The higher the rate, the larger amount of information needs to be sent, so the longer total migration time of the VM.

Considering the introduced aspects, the trade-off between reducing the operating costs (including power and cooling) of cloud infrastructures on the one hand, and keeping the customers satisfied in terms of QoS on the other hand, brings many challenges for cloud providers. Inappropriate management decisions such as frequent switching on and off virtual machines, or large number of unnecessary VM migrations can lead to SLA violations and consequently penalty cost to the cloud provider that can inversely affect the cost efficiency. The aim of this chapter is to propose a solution to assist the cloud provider to reduce the infrastructures operating costs without degradation of providing QoS.

3.2 Modeling VM Placement Problem

In this section, we formally model the aspects of cloud that are used in the proposed VM placement approach.

VM States. At each point of time t each VM can operate within two possible sets, either already allocated to a PM, $allocated(t)$, or has to be allocated, $waiting\ (t)$. A set consists of all VMs is called $all\ (t)$, where $all\ (t) = waiting\ (t) \bigcup allocated(t)$. The set $migrated\ (t)$ defines a set of VMs that are being migrated to other PMs at time t, where $migrated\ (t) \subseteq allocated(t)$, i.e., all VMs of this set are currently under migration. At each execution step, a VM placement method should find a target PM for: (i) all the VMs in the set $waiting\ (t)$; (ii) the VMs from the set $allocated\ (t)$ that their current allocation is not optimal enough based on a calculated utility value.

Resources. In our modeling, a data center consists of M distinct PMs. Each PM m is defined by a certain set of resources R. Each resource r has a known limited capacity C_{mr}, where $m \in \{1..M\}$ and $r \in \{1..R\}$. We define the binary variable $x_{ij}(t)$ that indicates if a VM v_i is allocated to a PM j at time t. Equation (3.1) states that each VM from the set $allocated\ (t)$ is allocated exactly to one PM.

$$\sum_{j=1}^{M} x_{ij} = 1, \quad \forall\ v_i \in allocated\ (t) \tag{3.1}$$

Each VM v_i has its specifications that define an upper bound of each resource $max(vr_{ir}(t))$ required by it at any point of time. During each execution step, a VM requires a certain amount of resources vr_{ir} that is considered during the decision making process of the VM placement. Since these resources are not being necessary provisioned for the VM, we introduce the amount of resources $vp_{ir}(t)$ that are provided for the VM. This value can be less (in case of the VM downtime) or equal to the resources required by the VM $vr_{ir}(t)$. Equation (3.2) guarantees that the amount of the provisioned resources for all VMs allocated to a PM does not exceed the overall capacity of the PM.

$$\sum_{i\ \in\ allocated\ (t)} x_{ij}(t) \cdot vp_{ir} \leq C_{jr}, \quad \forall\ j = 1..M,\ r = 1..R \tag{3.2}$$

Moreover, Equation (3.3) states how the utilization U_{jr} of a PM j and certain resource r with allocated VMs can be computed. Note that the primary focus of this chapter in on CPU as the cloud resource.

$$U_{jr} = \sum_{i \in allocated\ (t)} x_{ij}(t) \cdot vp_{ir}, \quad \forall\ j = 1..M,\ r = 1..R \tag{3.3}$$

Live VM migration. In Equation (3.4), we define a binary variable $y_{ij}(t)$ that indicates a VM v_i is under migration to a PM j at time t. This equation states that each VM from the set $migrated\ (t)$ can be migrated exactly to one PM.

$$\sum_{j=1}^{M} y_{ij} = 1, \quad \forall\ v_i \in migrated\ (t) \tag{3.4}$$

In our model, we assume that the migration of a VM does not affect the resources of a target PM until the migration is completed. Equation (3.5) states how DPR depends on the RAM size of a migrated VM:

$$dpr_i(t) = f(vr_{iram}), \quad \forall\ v_i \in migrated\ (t) \tag{3.5}$$

where $dpr_i(t)$ is the DPR of the VM and f is a custom defined functional dependency. For the simplicity, we assume f is a certain linear function. The amount of migrated RAM of VM i to another PM, $migratedRAM_i(t)$, is computed by Equation (3.6), where $bw(t)$ is a bandwidth speed rate between the source and the target PMs and $\Delta\ (t)$ is a period when the VM has been under migration.

$$migratedRAM_i(t) = \frac{bw(t) \cdot \Delta\ (t)}{dpr_i(t)} \tag{3.6}$$

Energy consumption and cost. We utilize a commonly used technique for power saving, namely dynamic voltage and frequency scaling (DVFS) [107]. DVFS allows to adjust the frequency of a microprocessor and thereby to reduce power consumption. In our model, energy consumption of a certain PM j is defined by CPU utilization and is stated in Equation (3.7), where f is the power specification of the PM:

$$W_j = f(U_{jCPU}) \cdot \Delta\ (t) \tag{3.7}$$

Energy consumption of a data center is the sum energy consumption of all included physical machines plus energy consumption for the cooling of the data center. As originally modeled in [188], overall energy consumption of a data center is defined as Equation (3.8):

$$W_{DC} = \sum_{i=1}^{n} W_i \cdot pPUE_{DC}(T) \tag{3.8}$$

where $pPUE_{DC}(T)$ is the pPUE rate of data center at temperature T. The energy cost of a data center for a given period of time depends on energy price at that period and the amount of consumption. In our model, energy cost is defined as Equation (3.9), where P_{DC} is the regional energy price of the data center location.

$$C_{DC} = W_{DC} \cdot P_{DC} \tag{3.9}$$

3.3 Designing the Decision Model

Building a decision model is started with the definition of objectives and an appropriate set of actions that are allowed to achieve the provider's goal, which is reducing the operating costs while satisfying the customers in terms of QoS. In our model, the set of possible decision actions are *Allocate VM*, *Migrate VM*, *Switch-on PM*, and *Switch-off PM*. Afterwards, we identify a set of criteria which are important to be considered from the cloud provider's point of view during the VM placement. Each criterion is a function of a certain quantitative measurement of a cloud infrastructure. Table 3.1 contains a list of criteria used in our model, and they are introduced in the following:

criteria	abbreviation	related equations
VM unavailability	g_1	Eq. 3.10
PM power consumption (incl. cooling)	g_2	Eqs. 3.7, 3.11
PM CPU utilization	g_3	Eqs. 3.1, 3.2, 3.3
VM migration duration	g_4	Eqs. 3.4, 3.5, 3.6
Energy price	g_5	Eq. 3.12

Table 3.1: The list of criteria used for the VM placement modeling

1. **VM unavailability** ($\mathbf{g}_1$). There are several assumptions based on which we define the migrated list *migrated* (t). We assume that the penalty cost is relatively high for all the incoming requests, hence the cloud provider aims to avoid placement of the virtual machines to the physical hosts, where the SLAs can be violated with a high possibility. Hence, we define *VM unavailability* (g_1) according to Equation (3.10):

$$g_1 = \frac{\sum(\textit{downtime duration } v)_i}{\textit{billing period}}, \quad \text{where } v \in \textit{allocated } (t) \tag{3.10}$$

   where the numerator is the duration of VM downtime i during the billing period. A high value of this criterion increases the possibility of SLA violation.

2. **PM power consumption** ($\mathbf{g}_2$). It directly influences the energy cost of the cloud provider. Equation (3.11) shows the calculation of g_2 for a certain PM j:

$$g_2 = (W_{max} - W_j \cdot pPUE_{DC}(T))/W_{max} \tag{3.11}$$

   where W_{max} is a constant that defines the maximal utilized power of a PM by considering the energy consumption for cooling. While W_j is the PM power consumption (Equation (3.7)), $pPUE_{DC}(T)$, as introduced in Equation (3.8), is the pPUE rate of the data center at temperature T. Equation (3.11) utilizes the pPUE rate of a data center where a certain PM is hosted. Indeed, we define g_2 as a function where values closer to 1 are preferred over the values closer to 0.

   To clarify Equation (3.11), assume PM_1 with $W_1 = 150Wh$ is hosted in DC_1 with $pPUE = 1.2$, thus we can imply that the real power consumption of PM_1 is $180Wh$. While $W_2 = 160Wh$ and it is located in DC_2 with $pPUE = 1.1$, so the real power consumption of PM_2 is $176Wh$. Therefore, based on the Equation (3.11), if $W_{max} = 250Wh$, $g_2(PM_1) = 0.28 < g_2(PM_2) = 0.296$, and as the result in our model migrating the VM to PM_2 at DC_2 is preferred to PM_1 at DC_1 despite the lower power consumption of PM_1.

3. **PM CPU utilization** ($\mathbf{g}_3$). This criterion is an indicator of efficient energy consumption. Although a cloud provider tends to utilize as less resources as possible, it should consider the risk of higher CPU demands than the PM capacity which may lead to QoS degradation.

4. **VM migration duration** ($\mathbf{g}_4$). Run time load balancing in cloud infrastructure is performed via live VM migration. Since a lower *VM migration duration* decreases the period of VM re-allocation, it allows more efficient use of cloud resources, so in our model, we try to decrease this criterion.

5. **Energy price** ($\mathbf{g}_5$). It explicitly impacts the energy cost of a cloud provider. Equation (3.12) defines the calculation of this criterion:

$$g_5 = (P_{max} - P_{target\ PM})/P_{max} \tag{3.12}$$

where P_{max} defines the maximal energy price over all geographically distributed data centers managed by a certain cloud provider, and $P_{target\ PM}$ is the energy price of a data center, where hosts the target PM that the VM is going to be migrated.

In summary, some of these criteria can be directly measured or observed (e.g., g_2, g_3, g_5). However, some of them (e.g., g_1, g_4) depend on the hidden factors, and such dependencies induce a level of uncertainty during the decision making process from the cloud provider's point of view. Therefore, utilizing Bayesian networks is a proper way as they can reason under uncertainty.

3.4 VM Placement Phases

The proposed VM placement approach works in the following three phases:

1. Designing the Bayesian network to model the expert knowledge of the cloud infrastructure management.
2. Quantifying the underlying measures of each criteria used in the Bayesian network model.
3. Applying multi-criteria decision analysis method [177] to create a utility function as the decision making indicator.

As previously explained in the background chapter (see Section 2.2.1), **Bayesian networks** are graphical models that represent variables of interest and probabilistic dependencies among them. The main benefit of such models is their ability to explore the causal relations between the key factors using Bayes theorem.

Although a Bayesian network model can be efficiently used to aid decision making by observing the value of uncertainty corresponding to each node, since the VM placement problem is a multi-criteria decision problem, utilizing a Bayesian network model solely is insufficient. To this aim, the **MCDA** method, previously explained in the background chapter (see Section 2.2.2), is an effective mean to combine measured results and rank all alternatives. MCDA allows sophisticated and flexible utilization of Bayesian networks in decision making analysis [177].

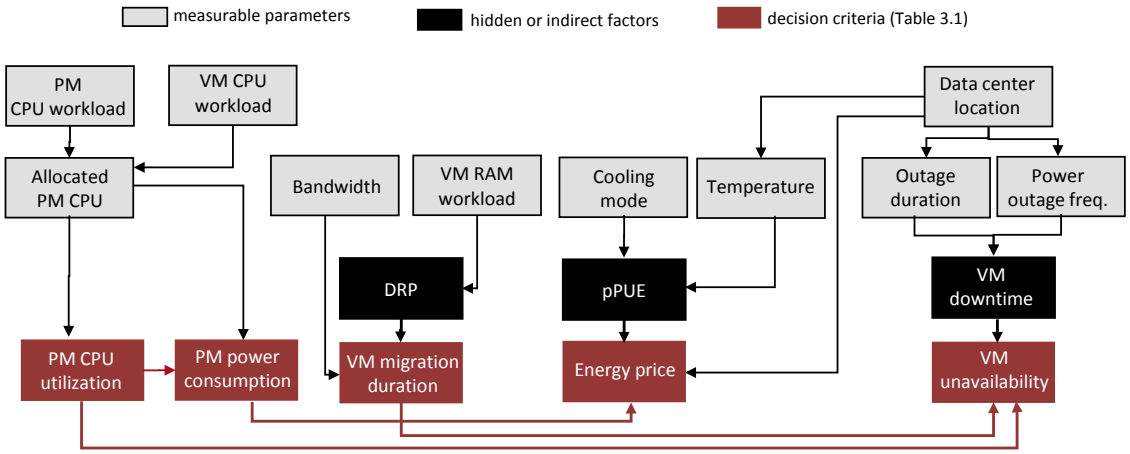


Figure 3.1: A simplified snapshot of the designed Bayesian network

Phase 1: designing the Bayesian network. As the first phase of the proposed approach, the Bayesian network depicted in Figure 3.1 is constructed. For the sake of readability, this figure represents a simplified version of the designed Bayesian network. Recalling the steps of transferring a problem into a Bayesian network model, previously presented in the background chapter (see Section 2.2.1), here we follow them to design the Bayesian network for the VM placement problem.

The initial step is to recognize the goal of modeling which is prediction is our work. This model is used to enable the decision making for managing geographically distributed cloud data centers. The initial model is extracted based on a data collection approach from a group of technical domain experts who have enough knowledge of cloud infrastructure management. The next step is to configure the initial model by using the factors that can influence the decision, e.g., using the time- and location dependent input parameters that influence the management decisions such as temperature data or regional power outage statistics (see Section 3.6.2 for more details).

The structure of the designed Bayesian network is shown in Figure 3.1 in which there are three types of nodes. The gray nodes define the parameters that can be directly measured at runtime. The red nodes denote the criteria presented in Table 3.1, which have influences on the VM placement decision making. The values of the red nodes are further used as the inputs of the utility function. The black nodes are the hidden factors that indirectly affect the decisions. The probabilistic dependencies between nodes are determined based on the experts' knowledge and expressed by conditional probability table, previously introduced in the background chapter. In our problem, CPT of each node is obtained by extracting statistical data of the cloud data centers, the specifications of the PMs inside each data center, the used time- and location dependent parameters, and the expert domain knowledge.

Furthermore, in order to proactively apply optimum actions at runtime on the cloud infrastructure, we use several prediction policy that can estimate the future workload of resource demands. These **workload prediction policies** are used as the input of the

g_1,g_2,g_4,g_5	0:10	10:20	20:30	30:40	40:50	50:60	60:70	70:80	80:90	90:100
value	1	0.9	0.8	0.7	0.6	0.5	0.4	0.3	0.2	0.1
g_3 [%]	0:10	10:20	20:30	30:40	40:50	50:60	60:70	70:80	80:90	90:100
value	0.125	0.25	0.375	0.5	0.625	0.750	0.875	1	0.66	0.33

Table 3.2: Mapping the continuous criteria values to the discrete values

BN. Each policy uses a different technique to predict the future workload, as defined in the following:

- *Last workload (BN-LW)*: the next workload is estimated as equal to the last value.
- *Trend workload (BN-TW)*: the next values follow a certain linear trend.
- *Linear regression on workload (BN-LRW)*: the next values are predicted by applying the linear regression of the historical data.

Phase 2: quantifying the underlying measures of the criteria used. While the Bayesian network model is constructed in the first phase, the second phase is to convert the chosen criteria (continuous values) to discrete values. Table 3.2 represents the conversion of each criterion value $\in [0, 100]$ to its corresponding discrete value $\in [0, 1]$, where 0 denotes the worst value and 1 indicates the best value. For all the criteria except g_3, 10% increments of their values is considered as 0.1 increment of the corresponding discrete value. While, in the case of g_3, the discrete value is found empirically, so that if the value exceeds 100% (i.e., over-usage of CPU), the selected action which leads to this condition is immediately rejected.

Phase 3: applying MCDA method. As the third phase, once discrete values for all criteria are computed based on the previous phase, MCDA method is applied to combine the values for each possible action, namely allocation or migration to the PMs, and ranks the results. Each criterion g_i is defined with a weight w_i that represents its relative importance in the context of the given decision problem. Equation (3.13) defines the utility function $U(a)$ of an action a:

$$U(a) = \sum w_i \cdot g_i(a) \tag{3.13}$$

Where $g_i(a)$ is taken based on the discrete utility value of each criterion presented in Table 3.2. The utility function $U(a)$ is used to evaluate the benefits of the possible actions. A VM is either allocated or migrated to a PM with the highest utility value.

3.5 Algorithms for VM Placement

We propose a VM placement algorithm based on the defined utility function (Equation (3.13)) to leverage the benefits of the geographical distribution of the data centers managed by the cloud provider. The proposed algorithm makes decisions according to the time- and location-dependent factors of such data centers such as regional energy prices, or the temperature difference between these data centers.

Algorithm 3.1: VM allocation algorithm

```
   input  : pmList, vmList: vm ∈ waiting (t)
   output : vmAllocationMap
 1 vmList.sortDecreasingSLAPenalty();
 2 foreach vm ∈ vmList do
 3 |  maxUtility ← 0 ;
 4 |  allocateToPm ← NULL;
 5 |  foreach pm ∈ pmList do
 6 |  |  utility ← computeUtility(pm,vm);          ▷ Equation (3.13)
 7 |  |  if utility > maxUtility then
 8 |  |  |  allocateToPm ← pm;
 9 |  |  |  maxUtility ← utility;
10 |  |  end
11 |  end
12 |  if allocateToPm ≠ NULL then
13 |  |  vmAllocationMap.put(vm, allocateToPm);
14 |  |  if allocateToPm.isSwitchedOff() then
15 |  |  |  allocateToPm.switchOn();
16 |  |  end
17 |  end
18 end
19 return vmAllocationMap;
```

VM placement algorithm use the Bayesian network model along with applying the MCDA method. In our proposed solution, the virtual machine placement can be divided into the allocation of the incoming VM demands to PMs (i.e., the *Allocate VM* action), and the consolidation of the current running VMs (i.e., the *Migrate VM* action). The VM allocation is triggered when a new VM request arrives, while the VM consolidation is applied periodically on running infrastructure at each control interval. Both VM allocation and consolidation use a similar best fit decreasing heuristic that utilizes a certain utility function for assessments of the most optimal decision.

3.5.1 VM Allocation Algorithm

As presented in Algorithm 3.1, first VMs $\in$ *waiting* (t) are sorted in a decreasing order by their SLA priorities. In our approach, we support three SLA priority levels, namely *Gold*, *Silver*, and *Bronze*. They define the priority of resource allocation for a VM. A VM with a *Gold* SLA has the highest priority for the resource allocation and consequently has the highest penalty cost in case of SLA violation. Afterwards, based on the final utility value for each PM according to Equation (3.13), a PM with the highest utility value is chosen as the target PM for allocating the VM. If the chosen PM is off, the action *Switch-on PM* is applied (Lines 14-16).

Algorithm 3.2: VM consolidation algorithm

```
   input  : pmList, vmList: vm ∈ allocated (t) \ migrated (t),
            vmAllocationMap
   output : vmMigrationMap
 1 vmList.sortDecreasingSLAPenalty();
 2 foreach vm ∈ vmList do
 3 |   maxUtility ← 0 ;
 4 |   migrateToPm ← NULL;
 5 |   foreach pm in pmList do
 6 |   |   utility ← computeUtility(pm,vm);          ▷ Equation (3.13)
 7 |   |   if utility > maxUtility then
 8 |   |   |   migrateToPm ← pm;
 9 |   |   |   maxUtility ← utility;
10 |   |   end
11 |   end
12 |   currentPm ← vmAllocationMap.get(vm);
13 |   if migrateToPm ≠ currentPm then
14 |   |   vmMigrationMap.put(vm, migrateToPm);
15 |   end
16 end
17 foreach pm ∈ pmList do
18 |   if pm.hasNoVMs() then
19 |   |   pm.switchOff();
20 |   end
21 end
22 return vmMigrationMap;
```

3.5.2 VM Consolidation Algorithm

The consolidation of the running VMs is performed in two steps. First, the VMs that need to be consolidated is identified, based on the calculated utility value of each physical machine using the presented workload prediction policies. Then, the chosen VMs are migrated according to Algorithm 3.2 that acts similar to the allocation algorithm (Algorithm 3.1). Migration of a VM to a certain PM is triggered, if the utility value of that PM is higher than the utility value of the current host PM. Algorithm 3.2 triggers *Switch-off PM* action, if there is no allocated VMs on a PM after the VMs migration (Lines 17-21).

3.6 Implementation Details

In this section, we first introduce a simulation tool that is designed and implemented for the evaluation of the proposed VM placement approach, then we explain the data extracted from the real-world traces used as the input parameters for simulating a realistic environment.

3.6.1 CloudNet Simulation Framework

CloudNet [81] is a framework that allows cloud providers to simulate their infrastructure in a repeatable and controllable way, in order to find the performance bottlenecks, and evaluate different management scenarios under real-world data traces. The most important feature of *CloudNet* that distinguishes it from the other similar frameworks is the ability of simulating geographically distributed cloud infrastructures while taking into account time- and location-dependent parameters such as infrastructure energy and cooling costs, power outage statistics, regional temperature and electricity prices. It also supports the three presented SLA priority levels with different penalty cost. *CloudNet* is a flexible simulation framework written in `Java`, which is designed based on the usage of Bayesian networks for the decision making analysis. It follows a loosely coupled design paradigm, where its components communicate through a message oriented middleware (MOM).

For the evaluation of the proposed VM placement approach, we simulate the management of geographically distributed data centers with frequent power outages by using *CloudNet*. To facilitate the reproduction of our research, we release the source code of *CloudNet*[1].

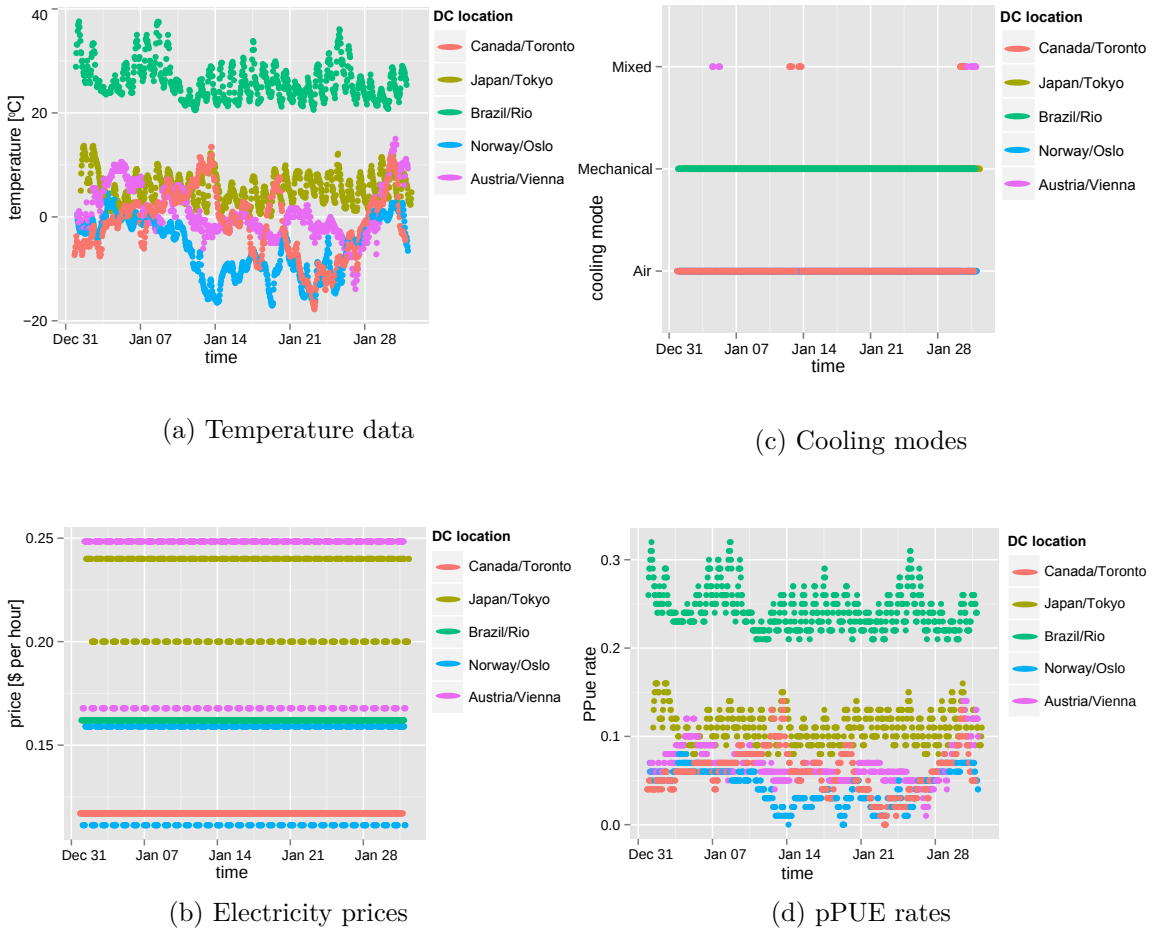
3.6.2 Time- and Location-dependent Input Parameters

For the evaluation, we setup *CloudNet* with five distributed data centers located in different time zones. Each location has a regional electricity price and temperature value. The temperature changes can cause the necessity of using different cooling modes and directly affect the pPUE rate. By utilizing a competitive energy price among the chosen data centers and the temperature differences due to the various time zones, the proposed algorithms are able to make a migration decision depending on the temperature and energy price thereby decreasing the operating costs for the cloud provider. We use the following real-world data traces as the input parameters of *CloudNet*:

- **Temperature data**. We retrieve the real temperature data traces for the chosen period (Jan-Feb 2013) and selected locations from the public web service, `Forecast.IO` [6] with the granularity of one hour (see Figure 3.2a).

- **Cooling modes**. We simulate Emerson's DSE^{TM} cooling system, described in [188]. This system has three different cooling modes: *Air*, *Mechanical*, and *Mixed*. One

[1] *CloudNet* source code: `https://github.com/dmitrygrig/CloudNet`



(a) Temperature data

(c) Cooling modes

(b) Electricity prices

(d) pPUE rates

Figure 3.2: Real-world data traces for the chosen data centers used as input parameters

mode switches to another one when the outside temperature is changed. In our evaluation, we switch *Air* mode to *Mixed* after temperature exceeds 12°C and *Mixed* to *Mechanical* after exceeds 18°C. Figure 3.2b depicts the switching between various modes of cooling system.

- **Electricity prices**. Electricity prices for each location is extracted using statistics in [4]. Some locations such as Austria have different pricing models for day and night as shown in Table 3.3. As presented, some locations such as Austria have different pricing models for day and night. Figure 3.2c shows changes in electricity prices for different locations.

- **Power outage statistics.** We obtain the data traces of the power outages corresponding to the chosen locations and period from [14]. As shown in Table 3.3, the electric measure *system average interruption duration index* (SAIDI) is utilized. Note that we use the scaled real-world values of one year of the simulation period

data center specification	Brazil	Canada	Norway	Austria	Japan
Day/Night switch hours (hour)	8-23	8-23	8-23	6-22	8-23
Day energy price ($/kWh)	0.162	0.117	0.159	0.2484	0.24
Night energy price ($/kWh)	0.162	0.117	0.1113	0.1678	0.20
SAIDI (min/month)	1101.6	220	218	39	6

Table 3.3: CloudNet input parameters for the chosen data centers

of one month in order to better show the ability of our approach in handling more unreliable data centers in terms of power outage.

- **PM power specification**. We use data traces of *SPECpower benchmark* [13] to define power specification of each physical machine in the simulated data centers.

Figure 3.2d shows the values of the pPUE rate for the chosen data center locations. In general, as shown in Figures 3.2a and 3.2d, a lower temperature drastically decreases pPUE and hence is more energy efficient due to the lower cooling cost.

The described simulated framework and the introduced input parameters are used for evaluating the VM placement approach, including the VM allocation and VM consolidation algorithms, in comparison with two state-of-the-art baseline algorithms. The evaluation details are explained in Section 7.1 of the evaluation chapter.

CHAPTER 4

Service Selection in Multi-Cloud

Cloud computing popularity is growing rapidly and consequently the number of companies offering their services in the form of cloud services (i.e., IaaS, PaaS, or SaaS) is increasing. The benefits of the cloud infrastructure offerings are encouraging application owners to lease resources from the cloud providers instead of operating their own data centers. This can help them to reduce the maintenance overheads and operating costs while being able to better satisfy their end users in terms of QoS as long as they deploy their application in suitable cloud services [184].

The diversity of cloud providers both in the number of players and the variety of offered services forces cloud customers to deal with a complex service selection problem [45]. Meanwhile, this problem is getting even more challenging with the emergence of a new delivery model of cloud services in which cloud customers can deploy their applications on multiple clouds instead of sticking to a single cloud and thereby achieving a better cost and QoS. As previously discussed in the background chapter, in our work we focus on multi-cloud delivery model where the cloud customer is aware of using multiple clouds, and usually a third party, i.e., a middleware, deals with the heterogeneity of cloud providers.

In this chapter, we address research question II by presenting a multi-cloud service allocation framework, and in particular, a multi-cloud service selection approach (contribution II). The proposed approach is used by an application owner, as a potential cloud infrastructure customer, to cost- and QoS-aware leverage the multi-cloud model. The proposed multi-cloud service selection chooses the most suitable combination of services that best satisfy the cost and QoS requirements of the customer. In this chapter, first a realistic use case is introduced as the motivation, then the SLA-based service allocation framework is proposed. Afterwards, we elaborate the proposed multi-cloud service selection approach, including an SLA definition and a service selection algorithm that use prospect theory. To the best of our knowledge, this is the first application of prospect theory in the scope of cloud computing service selection.

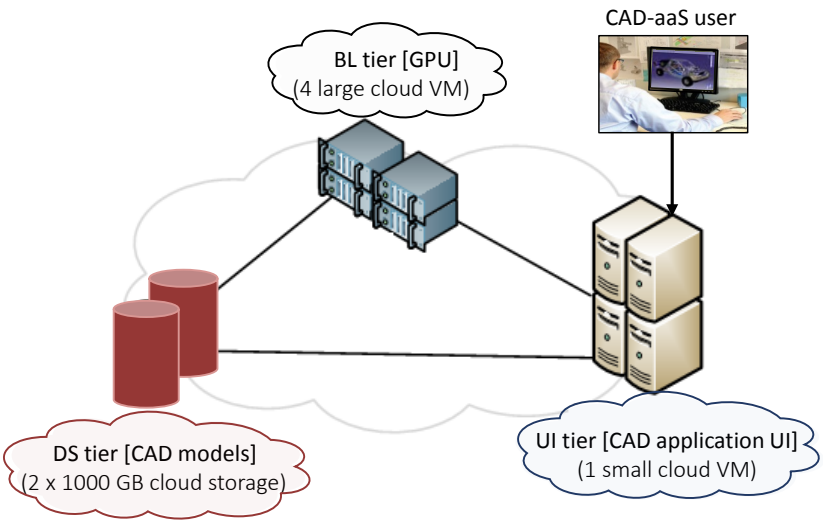


Figure 4.1: CAD-aaS as a motivation use case

4.1 Motivation

By using the multi-cloud, dependent components (or tiers) of a single software application can be deployed on different cloud infrastructures which have different values for QoS and cost. From the perspective of the application owner, such a deployment can be considered as an abstract composite service with a set of functional and non-functional requirements for each component included. The question remains how to, on the one hand, score and select services for each single component and on the other hand, optimize this selection in such a way that the requirements of the composite service are also satisfied. Furthermore, the concerned QoS parameters can be conflicting or have various importance levels. As a motivation use case, we consider a computer-aided design (CAD) application owner, who aims to deploy the CAD application in multi-cloud and offers it as a public cloud software service, called CAD-as-a-Service (CAD-aaS); accordingly the CAD application owner is called CAD-aaS provider. The rationale behind choosing such a use case is that deploying CAD applications in the cloud has been recently investigated widely. One example is the CloudFlow project [2] that aims to make the cloud infrastructures a practical solution for manufacturing by automatic provisioning of SaaS applications over cloud infrastructure services.

In order to attract different groups of end users, CAD-aaS is delivered in different software editions which have a certain set of cost and QoS requirements, expressed as various SLAs. In our use case, there are three software editions: enterprise; professional; and standard. As depicted in Figure 4.1, the CAD application has three tiers: (i) the user interface tier, which needs one small cloud virtual machine; (ii) the business logic tier that provides the computations and needs for large cloud VMs equipped with graphics processing units (GPUs); (iii) the data storage tier that stores the CAD models and requires two 1000 GB cloud storage. To leverage the benefits of using multiple clouds,

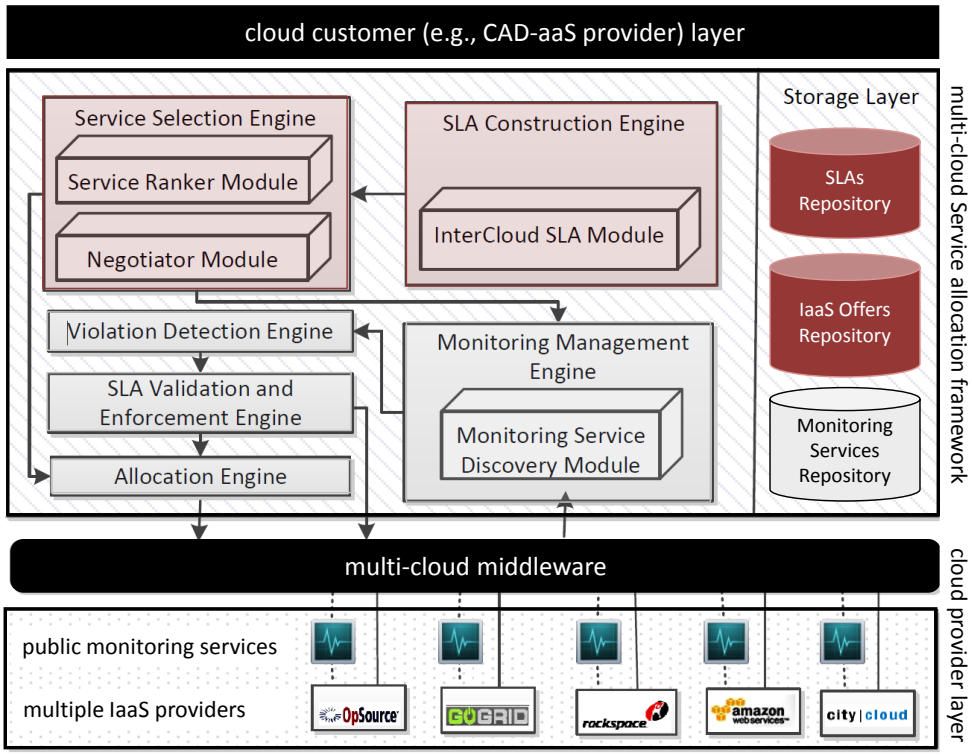


Figure 4.2: The architecture of multi-cloud service allocation framework

the CAD-aaS provider aims to deploy each tier of the CAD application on a separated cloud which best satisfy the QoS requirements of each tier, while keeping the overall leasing cost within a specific range. Different tiers of the CAD application communicate with each over the Internet based on the application topology. In the following sections, we exemplify our proposed solutions on this use case.

4.2 Multi-Cloud Service Allocation Framework

The architecture of the proposed multi-cloud service allocation framework is depicted in Figure 4.2. This framework is located between the cloud customer layer and the cloud infrastructure provider layer and manages the allocation of services in a multi-cloud environment with respect to the SLA requested by the customer. As previously discussed in the background chapter, there is a middleware in the multi-cloud model that carries out all the transactions between the cloud customers and the cloud providers. It is responsible for handling the interoperability issues in a multi-cloud, for example by communicating via different APIs of the involved cloud providers.

In this framework (Figure 4.2), *SLA Construction Engine* and *Service Selection Engine* (colored in the light red) handle the design-time activities such as SLA formation and service selection using their interactions with *SLA Repository* and *IaaS Offerings*

Repository components. While other components (colored in gray) cover the runtime activities such as SLA validation tracking, violation detection, and SLA enforcement. These activities are done by monitoring the allocated services at runtime and applying certain strategies in order to detect SLA violations and react accordingly. The service allocation process is realized in this framework through two main phases: (i) design-time phase which includes *SLA Construction*, and *Service Selection* steps; (ii) runtime phase consisting of *SLA Monitoring* and *SLA Violation Detection* steps. In the remaining of this section, these two phases are explained with more details.

4.2.1 Design-Time Phase

As the input of this phase, the CAD-aaS provider submits its requirements in terms of cost and QoS to the framework. According to the requirements, a set of high level SLAs is constructed which are cloud provider neutral. Afterwards, a service selection algorithm is used to find the best set of services that satisfy the requirements of each involved service. Additionally, it considers the whole request as a composite service and tries to choose an optimum combination of services by using service offerings from multiple cloud providers. The algorithm takes into account the runtime latency and data traffic as two main factors among the selected services of the composite service.

As the user satisfaction has been a subject of great interest to cloud providers, prospect theory is used in our selection algorithm to model the customer satisfaction as a function of service cost and quality aspects, as well as their importance level according to the customer. This theory as a descriptive model of decision making under uncertainty, is an alternative to utility theory while being more accurate in calculating the customer satisfaction. As previously discussed in the background chapter, the core rationale behind this theory is that the satisfaction is subjective, it means two different customers might have completely different satisfaction levels when offered the same service with a exactly similar quality level. This difference comes from the differences in priorities and prospect theory takes them into account for a better result.

4.2.2 Runtime Phase

An SLA cannot guarantee that the service is delivered as it has been described, similar to the case that a car guarantee cannot claim that your car will never break down. “In particular, an SLA cannot make a good service out of a bad one. However, it can mitigate the risk of choosing a bad service“ [20]. While we have the same purpose for using SLA in our research, the runtime phase is responsible for handling the activities including: SLA monitoring; SLA validation tracking; SLA violation detection; and SLA enforcement. While more investigation of runtime activities is out of the scope of this thesis, in the following sections, the focus is on the activities of the design-time phase by proposing an SLA-based multi-cloud service selection approach including a formal definition of the SLA model along with a multi-cloud service selection algorithm.

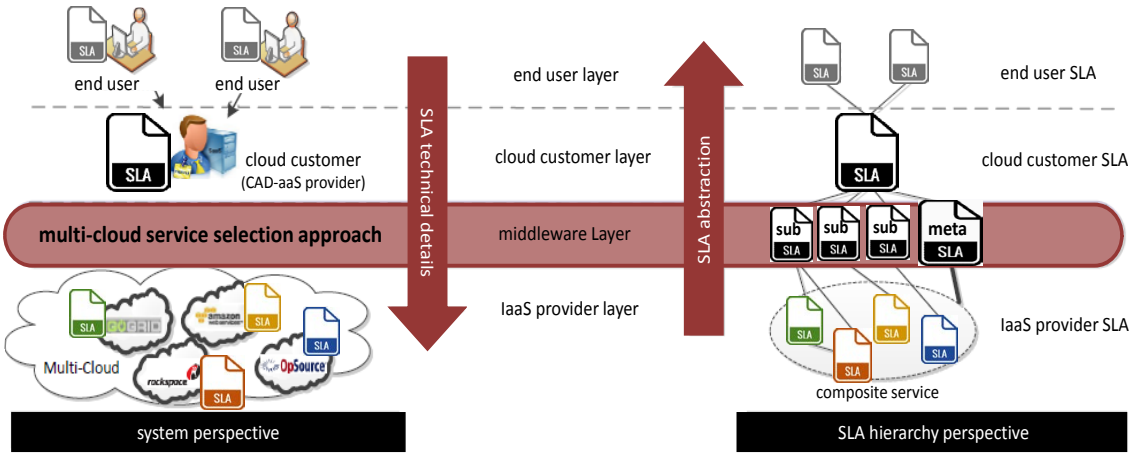


Figure 4.3: SLA hierarchy and service diversity in the multi-cloud model

4.3 Overview of the SLA-based Multi-Cloud Service Selection

As previously introduced in the background chapter, an SLA serves as an agreement for the expected functional and non-functional requirements of a cloud service between the involving parties [110], who are an infrastructure provider and cloud customer in our work. To provide a service selection approach to facilitate choosing the best set of services from a multi-cloud environment for a cloud customer, it is necessary to specify and manage the SLAs in two layers, as shown in SLA hierarchy perspective of Figure 4.3: (i) an SLA between the end user and the cloud customer (e.g., CAD-aaS provider) that directly reflects the QoS aspects of the offered services to the end users; (ii) an SLA between the cloud customer and the infrastructure provider, this SLA implicitly affects the end user satisfaction.

The proposed SLA-based multi-cloud service selection approach, as shown in the system perspective of Figure 4.3, is located between the cloud customer and cloud infrastructure provider layers and it handles the SLA heterogeneity and service selection. In our approach, the concept of sub-SLA and meta-SLA as InterCloud-SLAs are proposed to cover the requirements of the cloud customer for the requested composite service and all individual services.

The proposed multi-cloud service selection approach focuses on the SLA between the cloud customer and the IaaS provider which includes both functional and non-functional parameters. SLA functional parameters express the number and the type of the required cloud service. While each SLA non-functional parameter can be defined as a hard or a soft constraint. A hard constraint must be satisfied (the infrastructure cost *must* be less than a specific amount), while satisfaction of a soft constraint is not mandatory, but is preferred (response time *is preferred to be* less than 2sec). By SLA satisfaction, we *do provide* all the functional and hard non-functional parameters and also *try to provide* the

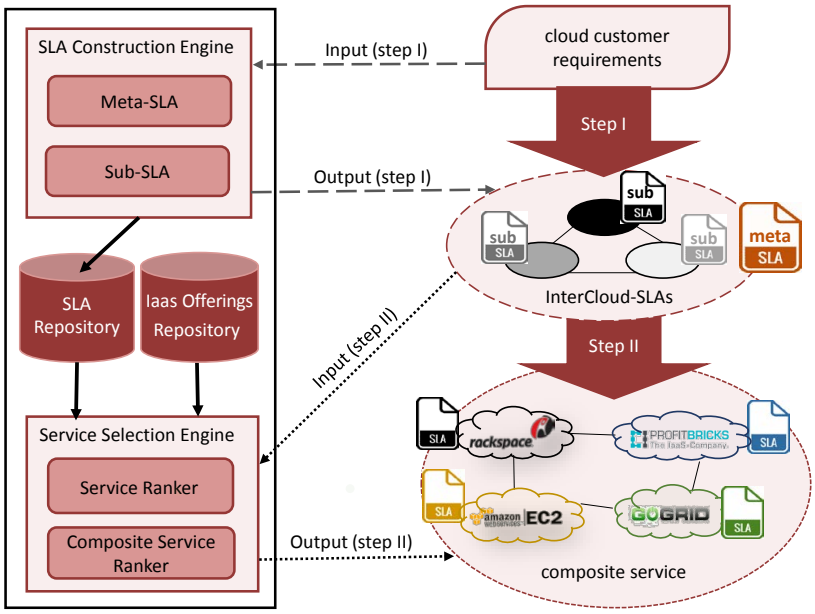


Figure 4.4: The architecture and steps of the multi-cloud service selection approach

soft constraints. The proposed approach works in two steps: *(i) SLA Construction*; *(ii) Service Selection*. Figure 4.4 depicts the architecture and the steps of the multi-cloud service selection approach along with inputs and outputs of each step. These two steps cover all activities of the design-time phase of the previously presented framework.

Step I: SLA construction

As the input of Step I, the cloud customer submits its infrastructure requirements of the *SLA Construction Engine* as a single extensible markup language (XML) file. These requirements contain two parts: (i) the first part includes the requirements for each cloud service (cloud VM or storage) as the host of each application tier; (ii) the second part contains the requirements of the whole deployed application, considered as a composite service. The information related to these two parts is extracted from the given XML file and transformed into a set of SLAs by using the principles of model driven architecture. The generated SLAs are provider independent, and named InterCloud-SLAs in our work. Note that as a recent investigation of SLA interoperability issues in a multi-cloud, an IEEE working group, InterCloud Working Group (ICWG)[1], has been established to develop a set of standards for InterCloud interoperability. The name InterCloud-SLAs has been inspired by the name of a sub-group of this working group.

The main purpose of constructing such InterCloud-SLAs is addressing the SLA interoperability issue in a multi-cloud. Based on the model driven architecture, previously introduced in the background chapter, an InterCloud-SLA can be considered as a platform

[1]IEEE InterCloud Working Group (ICWG): `http://grouper.ieee.org/groups/2302`

independent model. Since each cloud infrastructure provider has offered an SLA that is dependent on the details of its provided infrastructure, this SLA can be modeled as a platform specified model. One of the activities of this phase is to construct the PIMs and then transform them to the corresponding PSMs of the selected cloud providers. In other words, *SLA Construction Engine* is responsible to automatically transform a PIM to a PSM, i.e., to map the customer requirements to the available service offerings. While investigating more about proposing an automatic solution for such a mapping is out of the scope of this thesis, effective techniques can be found in [35, 155]. A formal definition of the proposed SLA used in our approach is introduced later in Section 4.4.

Step II: Service selection

As shown in Figure 4.4, the InterCloud-SLAs and the IaaS providers' offerings are the two inputs of this step. *Service Ranker* component is responsible to create a ranking list of services for each sub-SLA, which are then used by *Composite Service Ranker* component to score the combinations of services for the meta-SLA. The output of this phase is a composite service, which has the best satisfaction score among all other candidates. The details regarding the service scoring is presented as a service selection algorithm in Section 4.5.

4.4 SLA Formal Definition

In this section, we formally define the specification of the SLAs used in Step I. A set of m infrastructure services can be defined as Equation (4.1) where F_i is the functional parameter of service S_i and NF_i is its non-functional parameter.

$$\textit{Service Offerings} = \{S_1, \cdots, S_m\}, \quad \textit{where} \;\; S_i = \{F_i, \; NF_i\} \quad 1 \leq i \leq m \tag{4.1}$$

The InterCloud-SLA contains a set of n sub-SLAs and one meta-SLA and is defined as Equation (4.2).

$$\textit{InterCloud-SLA} = \{\{subSLA_1, \cdots, subSLA_n\}, \; metaSLA\} \tag{4.2}$$

Where each $subSLA_j$ and $metaSLA$ includes a set of functional parameters F and non-functional parameters NF that are defined as Equations (4.3) and (4.4).

$$subSLA_j = \{F_j, \; NF_j\} \quad 1 \leq j \leq n \tag{4.3}$$

$$metaSLA = \{F, \; NF\} \tag{4.4}$$

Non-functional parameters of $subSLA$ or $metaSLA$, NF_k are represented as Equation (4.5).

$$NF_k = \{Min_k, \; Max_k, \; W_k, \; T_k\} \quad 1 \leq k \leq l \tag{4.5}$$

Where Min_k and Max_k determine the accepted boundaries for the values of parameter k. $W_k \in (0, 1]$ represents the cloud customer priority on parameter k in the service selection.

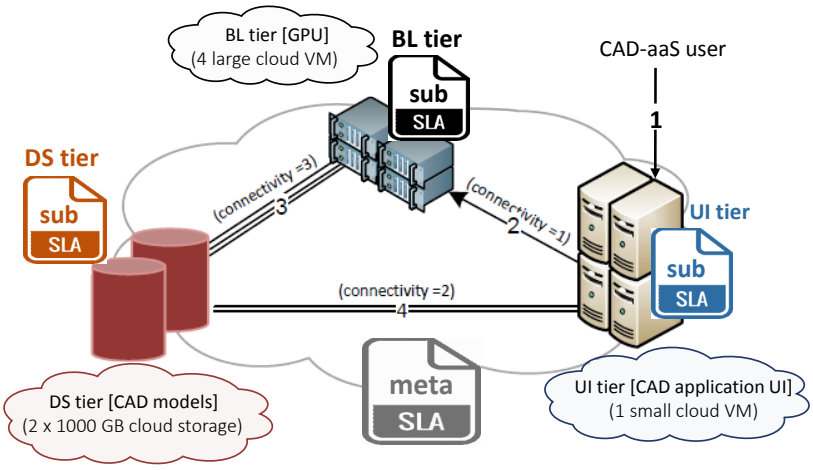


Figure 4.5: Meta-SLA and sub-SLAs of the CAD-aaS use case

A larger value for W_k represents the importance of the parameter k for the customer, and accordingly the more impact of this parameter on the service ranking. T_k specifies the type of constraint for parameter k, which can be a hard or a soft constraint. In our proposed service selection approach, the hard non-functional parameters are treated similar to the functional parameters. If the customer specifies no value for each of the introduced factors, the default values are assigned, $T_k = soft$ and $W_k = 0.5$ that show the medium importance (customer priority) of parameter k. Moreover, for Min_k and Max_k, the default values are defined as the smallest and largest values for parameter k within a corresponding set of *Service Offerings*. For example, if $k = availability$, then $Min_k = 90\%$ and $Max_k = 100\%$ can be assigned as the default values.

The last part in our modeling is a graph which describes the degree of connectivity among sub-SLAs, in which the connectivity can have a number between 1 and 3, a larger value represents a more connectivity. In this graph, nodes present the sub-SLAs and the edge shows to which degree these two corresponding components are going to transfer data at runtime, so it influences the traffic cost and latency of the composite service deployment. Figure 4.5 shows a complete version of the motivation use case presented in Figure 4.1. As depicted, each requested service of each application tier has a sub-SLA. Non-functional parameters related to the whole CAD composite service form the meta-SLA. The edge numbers represent the execution sequence of the CAD-aaS. The degree of data communications between the components is depicted as the connectivity value of each edge, which shows the amount of transferred data between the involved nodes. For example, in the CAD-aaS graph, the business logic tier node transfers the highest amount of data with the data storage tier node (i.e., connectivity=3), among all other edges.

4.5 Algorithm for SLA-based Multi-Cloud Service Selection

The cornerstone of the service selection step is a selection algorithm that works based on prospect theory to compute the cloud customer satisfaction score for a certain service. This theory is proper for describing user decisions among various choices under uncertainty, and considers human behavior in the computation of the user satisfaction. The proposed algorithm supports service selection for a cloud composite service in a multi-cloud, and covers all functional and non-functional parameters of the proposed InterCloud-SLAs. The service selection algorithm is depicted in Algorithm 4.1 and is described in the following six steps.

Algorithm 4.1: SLA-based multi-cloud service selection algorithm

input : ServiceOfferingList, subSLAList, metaSLA
output: CompositeService

```
1  foreach subSLA ∈ subSLAList do
2  |  ServiceOfferingList.filter(F,Hard-NF);
3  |  foreach NF_i of S_i ∈ ServiceOfferingList do
4  |  |  foreach NF_j ∈ subSLA_j do
5  |  |  |  N_ik=normalize(NF_ik);                          ▷ Eq. (4.6)
6  |  |  |  computeSatisfactionScore(N_ik);                 ▷ Eq. (4.7)
7  |  |  end
8  |  |  FinalSatisfactionScore.compute(S_i,subSLA_j)       ▷ Eq. (4.8)
9  |  end
10 end
11 CompositionServiceList.add(ServiceOfferingList);
12 foreach Composition ∈ CompositionServiceList do
13 |  foreach NF ∈ metaSLA do
14 |  |  aggregateFunction(NF);                             ▷ Table 4.1
15 |  end
16 end
17 CompositionServiceList.filter(F,Hard-NF);
18 foreach Composition ∈ CompositionServiceList do
19 |  foreach NF ∈ metaSLA do
20 |  |  normalize(NF.AggregateValue);                      ▷ Eq. (4.6)
21 |  |  satisfactionScore.compute(Composition);            ▷ Eq. (4.7)
22 |  end
23 |  FinalSatosfactionScore.compute(Composition);         ▷ Eq. (4.8)
24 |  FinalCompositionScore.compute(subSLAs,metaSLAs);     ▷ Eq. 4.9
25 |  CompositionServiceList.sort();
26 end
27 return the first ranked CompositeService;
```

1. A set of services that satisfies the functional and hard non-functional parameters is chosen for each sub-SLA (line 1-2). We assume that the filtered list is not empty at this step, and the negotiation with the cloud customer in case of finding no service for the given requirements in out of the scope of this thesis.

$$Norm(NF_{ik}) = N_{ik} = \begin{cases} \frac{NF_{ik} - Min_{jk}}{Max_{jk} - Min_{jk}} & if\ a\ larger\ NF_{ik}\ is\ desirable \\ \\ \frac{Max_{jk} - NF_{ik}}{Max_{jk} - Min_{jk}} & if\ a\ smaller\ NF_{ik}\ is\ desirable \end{cases} \tag{4.6}$$

2. Due to the variety of supported non-functional parameters in metrics and scales for a given service set, they need to be normalized before being used in the service ranking. In Equation (4.6), we normalize QoS parameters in such a way that a higher value always means better. For instance, for the parameters like availability which a higher value is desirable, we use the first case of Equation (4.6), and for the parameters like cost, where minimization is the goal, the second case is used for the normalization. This equation calculates the normalized result, N_{ik}, for the values between their accepted boundaries [Min-Max]. For values better than the accepted boundaries, the result is 1 and for the ones that their values are not inside the boundaries, the normalized values are 0 (line 3-5).

3. Let N_{ik} is the normalized value of NF_{ik} of service S_i, the satisfaction scoring function (SSF) can be defined as Equation (4.7). This Equation computes the customer satisfaction score of the normalized value N_{ik} for each non-functional parameter of S_i based on W_j (the priority value of the customer) defined in $subSLA_j$ (line 3-7).

   The rationale behind SSF is based on prospect theory. This theory implies that changes in a specific quality aspects of a service are sensed more by customers who have assigned higher weights (priority) to those quality parameters. Indeed, the satisfaction of a cloud customer for a service is based on the gains and losses relative to the reference point (normalized value of 0.5) instead of taking the absolute normalized QoS parameters of that service. Moreover, satisfaction is influenced by the customer priority weight assigned to a specific quality parameter. According to this theory, the satisfaction function should be concave for gains, and convex for losses.

   To clarify the behavior of SSF, we present the diagrams of Figure 4.6. Each curve shows the behavior of SSF based on different priority weights for different non-functional parameters supported either at sub-SLA or meta-SLA, for the three defined software editions (standard, professional, and enterprise). For example, in Figure 4.6 (b) the accepted boundaries are 99.5% as the minimum and 100% as the maximum, so 0.4 (the normalized value of 99.7%) has different satisfaction scores for each software edition based on the specified priority weights assigned by customers for availability as an SLA parameter. Other diagrams of Figure 4.6 also show the influence of customer priority weights on SSF for differing non-functional

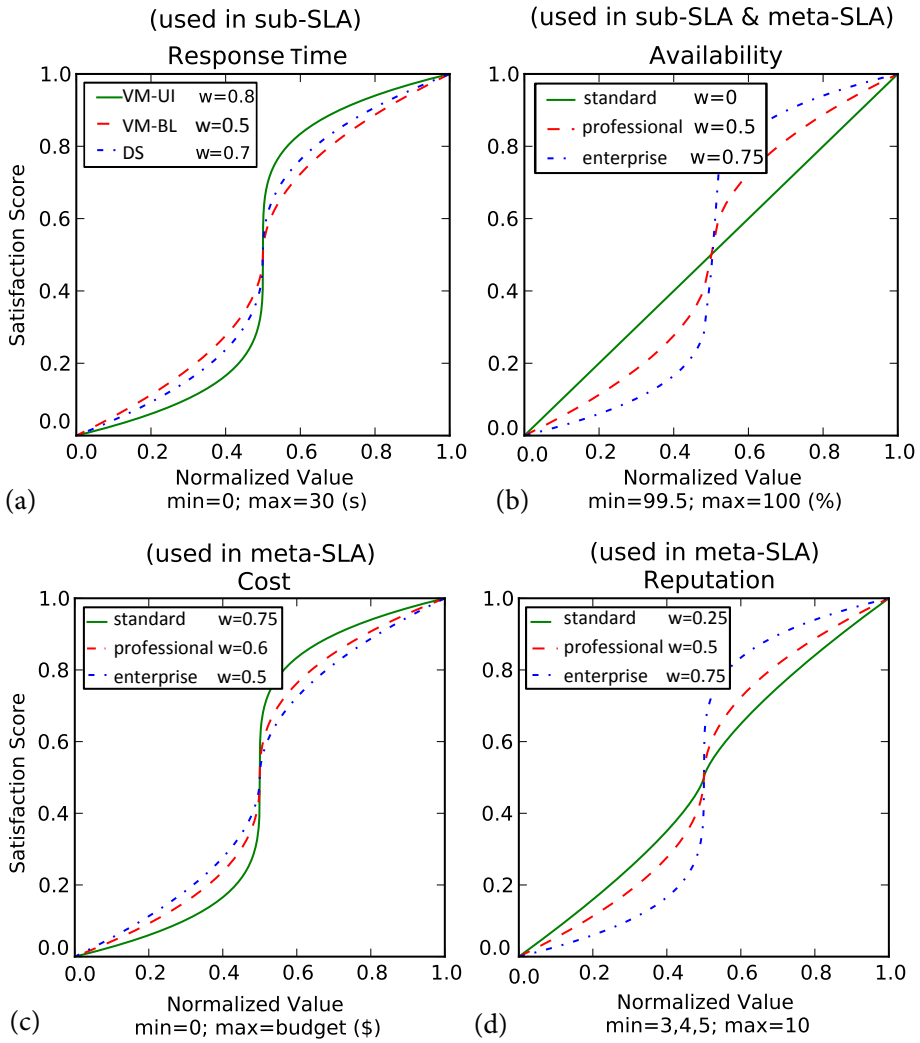


Figure 4.6: Satisfaction scores for sub-SLAs and meta-SLA, using prospect theory

parameters. The priority weights can also be different based on the goal of the customer for a certain service. For example, Figure 4.6 (a) shows the several priority weights that a customer has assigned to response time of the three application tier. As depicted response time for the virtual machine which is dedicated to the user interface tier is more important for the customer (having a higher priority weight) than the virtual machine for the business logic tier.

$$SSF(N_{ik}) = \begin{cases} 0.5.(2N_{ik}-1)^{1-W_{jk}} + 0.5 & N_{ik} > 0.5 \\ -0.5.(-2N_{ik}+1)^{1-W_{jk}} + 0.5 & N_{ik} \leq 0.5 \end{cases} \tag{4.7}$$

meta-SLA parameter	aggregate function formula
Budget	$Cost_{agg} = \sum$ {VM+storage+traffic}
Throughput	$Th_{agg} = min_{i=1}^{n} Th(S_i)$
Reputation	$Rep_{agg} = \frac{\sum_{i=1}^{n} Rep(S_i)}{n}$
Latency	$Lat_{agg} = \frac{\sum_{i,j=1}^{n} W(S_{ij}).Lat(S_{ij})}{\sum_{i,j=1}^{n} W(S_{ij})}$
Availability	$Ava_{agg} = \prod_{i=1}^{n} Ava(S_i)$

Table 4.1: Aggregate functions of meta-SLA parameters [192]

4. Until now, we have calculated the satisfaction score for each NF_i of service Si, so we have a satisfaction score set $\{SC_{i1}, \cdots, SC_{ik}, \cdots, SC_{il}\}$ calculated based on the $\{NF_{j1}, \cdots, NF_{jk} \cdots, NF_{jl}\}$. At this point, it is needed to compute the final satisfaction score of each service by the combination function (CF) presented in Equation (4.8) (line 8-10).

$$CF(S_i, subSLA_j) = \frac{\sum_{k=1}^{l} SC_{ik}.W_{jk}}{\sum_{k=1}^{l} W_{jk}} \tag{4.8}$$

5. The four previous steps (line 1-10) are executed on the sub-SLA level, while from this step on, the meta-SLA are processed. First, for all possible combinations of the chosen services corresponding to each $subSLA$, the aggregate non-functional values are calculated by using the corresponding aggregate function of each parameter, presented as Table 4.1 [192] (line 11-16). These functions calculate the aggregate values of composite service non-functional parameters based on its constituent services. In Table 4.1, $Lat_{(S_{ij})}$ at the aggregate function for the latency, represents the delay between the deployed services, which is dependent on the location where the host cloud infrastructure is located. The details of these functions and their corresponding evaluation values are presented in the evaluation chapter. Then, the algorithm again follows step 1 to 4 by considering the aggregate values, meta-SLA and all possible service combinations, which is named *Composition* (line 17-22).

6. For the final selection, the algorithm considers the influence of both sub-SLAs and meta-SLA. To this aim, the average *Final Satisfaction Scores* of services included in *Composition* is computed (line 23). Then, by Equation (4.9) in which $W_{metaSLA}$ is the customer priority weight of the meta-SLA and W_{subSLA} is the customer priority weight for the sub-SLA, the *Final Composition Score* is calculated for the final selection (line 24).

$$Score_{final} = W_{metaSLA}.Score_{agg} + W_{subSLA}.Score_{ave} \tag{4.9}$$

The first ranked composite service would be the output of the service selection algorithm (line 27). The top ranked composite service is the one with the highest satisfaction score. It means it includes the services that have the closest values to

the customer requested QoS. Hence, in the proposed algorithm the services whose have the best QoS values may not be chosen, instead the services which best satisfy the customer requirement are selected. This is beneficial in improving the overall service utilization, and reducing the leasing cost by not paying for the services which have far better QoS values than the customer requested QoS level.

The proposed service selection approach is evaluated by comparing its results with a state-of-the-art utility-based selection algorithm [101]. For this purpose, both algorithms are implemented in `Java` language and a set of simulation scenarios is conducted on these algorithms. The simulated-based evaluation results are discussed in Section 7.2 of the evaluation chapter. The used simulation environment is enriched by realistic data sets taken from the commercial public cloud infrastructure providers.

CHAPTER 5

Vertical Memory Elasticity Control

Since the owners of modern interactive cloud applications are becoming increasingly interested in having high and predictable, if not guaranteed, performance, the need for having robust elasticity solutions that would meet their SLAs is rising. Lack of such solutions result in a poor service performance, in case of unexpected workload and kills the satisfaction of end users. Several studies have shown that increased response time reduces the revenue [136]. For instance, Amazon found every 100ms of latency costs them 1% in sales [119]. Google found only half a second delay in search page generation time dropped traffic by 20% [79].

A possible solution to deal with this issue is cloud elasticity. As previously explained in the background chapter, horizontal and vertical are two types of elasticity. While horizontal elasticity allows virtual machines to be acquired and released on-demand, vertical elasticity allows adjusting the resources of individual virtual machines to cope with runtime changes. The problem of supporting elasticity for cloud applications can be categorized as an autonomic computing problem where systems make use of autonomic controllers [96]. These autonomic controllers can be realized in several ways, including statistical machine learning [34], and control theory [144]. As previously discussed in the background chapter, control theory is a promising fit to deal with unpredictable changes in systems by introducing feedback control loops [194]. Although the problem of automated resource provisioning is among the primary application areas in which control theory has been applied [89], cloud computing introduces many more challenges making control-based mechanisms a more prominent solution for such problem spaces. Moreover, in spite of the new trend of applying control theory in software systems, research on the application of control theory for enabling elasticity mechanism to support cloud elasticity is still in a preliminary stage [70]. In our research, we use control theory to synthesize a controller for vertical memory elasticity of cloud applications.

In this chapter, we address research questions III and IV by presenting their corresponding contributions III and IV, introduced in Chapter 1. We first experimentally motivate the effect of vertical memory elasticity on the application performance at Section 5.1. Then, we introduce a model that covers approaches that can realize cloud elasticity (Section 5.2). We present a performance-based controller as contribution III, using a control design process (Section 5.3) to guarantee the application performance objectives by adjusting the allocated memory of the VM hosting the application (Section 5.4). Afterwards, in order to enhance the resource utilization of the resource controller, while meeting the application performance, we propose a hybrid memory controller (Section 5.5) as contribution IV, which uses both the application performance and the resource utilization at the same time as elasticity decision making criteria. The hybrid memory controller allocates the right amount of memory for the cloud application to meet the application performance objectives, while achieving a high resource utilization. Finally, we provide the formal assessment of the two proposed controllers from the control perspective (Section 5.6).

5.1 Motivation

Horizontal elasticity has been widely adopted by commercial clouds due to its simplicity as it does not require any extra support from hypervisors. However, due to the static nature and fixed virtual machine size of the horizontal elasticity, applications cannot be provisioned with arbitrary configurations of resources based on their runtime demands. This leads to inefficient resource utilization as well as SLA violations since the demand cannot always exactly fit the size of the virtual machine. To efficiently utilize resources and avoid SLA violations, horizontal elasticity should be complemented with fine-grained resource allocations where virtual machine sizes can be dynamically adjusted to an arbitrary value according to runtime demands. Based on a European commission report on the future of cloud computing [160], vertical elasticity is one of the areas that is not fully addressed by current commercial efforts, although its importance is acknowledged. Vertical elasticity is also considered recently as a key enabling technology to realize resource-as-a-service (RaaS) clouds, and as one of the main driving features of the second-generation infrastructure as a service (IaaS 2.0) [138], where users pay only for the resources they actually use, and cloud providers can use their resources more efficiently and hereby serves more users [15, 112].

Nevertheless, from the research point of view, in the last decade, most elasticity research has focused on horizontal, while only few research efforts have addressed vertical elasticity [134] due to lack of support from hypervisors. However, vertical elasticity of resources has recently started to be supported by hypervisors such as Xen and KVM. Unlike horizontal elasticity that is widely supported by almost all commercial clouds, only a few cloud providers such as dotCloud[1] and ProfitBricks[2] have started commercial support for vertical elasticity. However, with the current rate of technological developments and

[1]dotCloud: `https://www.dotcloud.com`
[2]ProfitBricks: `www.profitbricks.com`

user expectations, the support of vertical elasticity techniques is becoming necessary by any public cloud service providers in the future [134].

In theory, one can make any cloud resource, like CPU or memory, vertically elastic for a cloud application if there is a way to measure the effects of the allocated resources on the application performance continually over time, and by at least one knob for changing the resource size. However, the practical exploitation of vertical elasticity is challenging due to the following reasons: (i) intrinsically dynamic and unpredictable nature of the workloads generated by the dynamic number of users at runtime; (ii) the difficulty of determining which resource (e.g., CPU or memory) is responsible for the application performance degradation [93]; (iii) non-linear relationship between the performance metrics (e.g., throughput or response time) and the amount of required resources; (iv) difficulty of determining the right time for adjusting the amounts of resources.

Multiple tiers of a cloud application may be involved in processing user requests, thus, requiring different resources such as CPU or memory. In our research, as we discuss in the following, we focus on vertical *memory* elasticity of the business logic tier which hosting Apache Web server. Apache Web server performance tuning point [1] emphasizes that, "the single biggest issue affecting Web server performance is memory. The more memory your system has, the more processes and threads Apache can allocate and use; which directly translates into the amount of concurrent requests or clients, Apache can serve". Note that allocating more memory to the virtual machine hosting data storage tier also leads to cache more data into memory, therefore, increasing the probability of a cache hit and consequent enhancement of the application response time (RT). However, due to the lack of research on memory elasticity of the business logic tier such as Apache server, and also the need for the further application support in case of vertical memory elasticity of the data storage tier (e.g., for MySQL [157]), we choose to primarily concentrate on the business logic tier. The proposed solution targets the applications that can benefit from live memory elasticity at runtime, i.e. application with dynamic memory requirements, and also applications in which the performance bottleneck is memory, not CPU.

In order to experimentally show that the dynamic thread creation of Apache is working as expected in accordance to the allocated memory, and highlight the effect of the allocated memory on the application performance, we conducted an experimental motivation. We use `ab`, Apache HTTP Server benchmark tool[3] for RUBBoS benchmark application [11]. RUBBoS is a bulletin board application, which enables users to browse and submit stories or comments. Some stories may attract a huge number of visitors in a short period of time. Since the nature of this application may include sudden bursts or occasionally daily or monthly peaks, the application needs to be able to quickly grow or shrink its resources. Therefore, cloud infrastructure is a suitable environment where the application can benefit from the cloud elasticity features. The aim of applying an elasticity solution for such an application is to dynamically adjust the amount of memory to keep the application performance, e.g., RT in our research, under a desired value regardless of the variation in workloads.

[3]Apache benchmark tool (`ab`): `https://httpd.apache.org/docs/2.2/programs/ab.html`

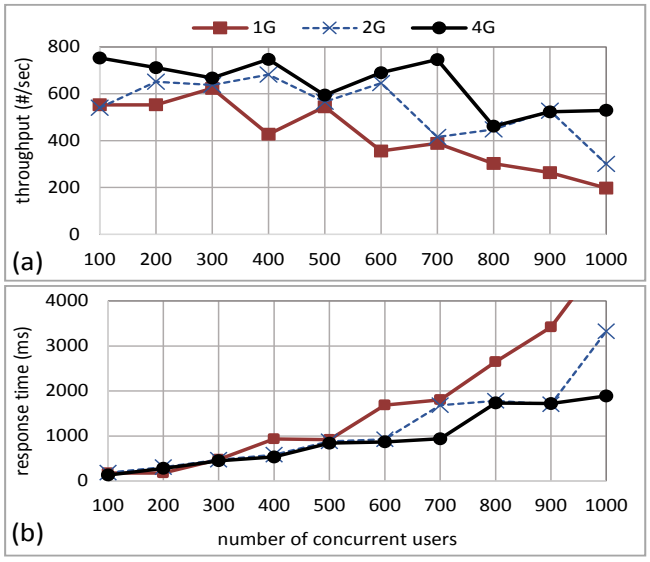


Figure 5.1: The effect of vertical memory elasticity on the application performance

To conduct the experiment, we deploy the BL and DS tiers of RUBBoS on different VMs, while provisioning sufficient memory and CPU cores to the VM hosting the DS tier. We also over-provisioned the VM hosting the BL tier in terms of CPU (8 CPU cores). We configure three different values for the allocated memory (i.e. 1GB, 2GB, and 4GB) in order to show the effect of allocated memory on the application performance under different workloads. By using a workload generator tool, *httpmon*[4], we define variable workload patterns which stress the BL tier for memory during the application lifespan at runtime by slowly increasing the number of users (i.e., requests/ Sec) from 100 to 1000 (with 100 users increment each time).

During the experiment, the number of concurrent Apache processes, effected by the number of concurrent users and allocated memory size, is monitored by using the Apache server status[5]. As shown in Figure 5.1, when more memory is allocated to the VM, a better application performance is achieved as Apache can create more threads to process the incoming requests in parallel. As the number of concurrent users is increased, the difference in performance (in both throughput and response time) among the three memory configurations becomes more apparent.

5.2 A Solution Model for Vertical Memory Elasticity

As shown in Figure 5.2, resource (e.g., memory) vertical elasticity solutions can be modeled as performance-based, capacity-based, and hybrid approaches:

[4] *httpmon*: `https://github.com/cloud-control/httpmon`
[5] Apache server status: `w3m http://VM-IP/server-status`

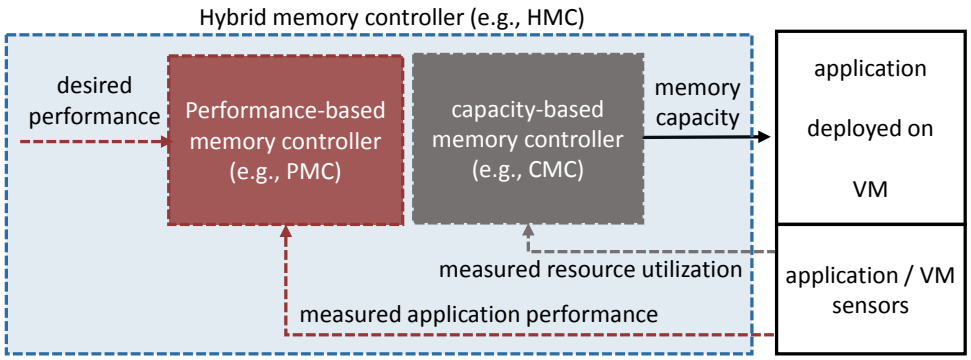


Figure 5.2: The solution model for vertical memory elasticity

- **Capacity-based**. As the most popular elasticity solution from the cloud provider point of view, such an approach uses resource utilization as the decision making criterion to change the amount of resource to be allocated. In other words, the utilization data is used to estimate the required resource at runtime. Efficient use of resources is one of the main advantages of such an approach. While decision making solely based on resource utilization can lead to violating performance guarantees as such decisions are oblivious to the application performances at runtime [111]. In other words, a sustained high resource utilization can tell us that the system is suffering, but it cannot determine by how much. While the whole point of elasticity is to figure out by how much we need to scale up to meet the current demand or scale down to avoid the resource wastage, and consequently taking the appropriate action [8]. In Figure 5.2, *capacity-based memory controller* (CMC) (the gray module) is representative of a capacity-based solution for realizing vertical memory elasticity. Such a capacity-based solution is used as a baseline approach to evaluate the proposed controllers in our research.

- **Performance-based**. As a new trend for realizing resource elasticity, this category of solutions decide to what extent either increase or decrease, the allocated resource based on the application performance properties, such as response time or throughput at runtime. In Figure 5.2, *performance-based memory controller* (PMC) (the red module) represents a performance-based solution for realizing vertical memory elasticity. This novel performance-based memory controller is proposed in Section 5.4.

- **Hybrid**. In Section 5.5, we bring up the idea of a new approach named hybrid in which we leverage from the advantages of both performance-based and capacity-based solutions. In Figure 5.2 *hybrid memory controller* (HMC) (the blue module) as a representative of hybrid elasticity solutions is depicted. While a capacity-based approach is inadequate to ensure the application performance, a performance-based approach may not able to provide a sufficient level of resource efficiency that a capacity-based approach can provide. Therefore, taking both the application

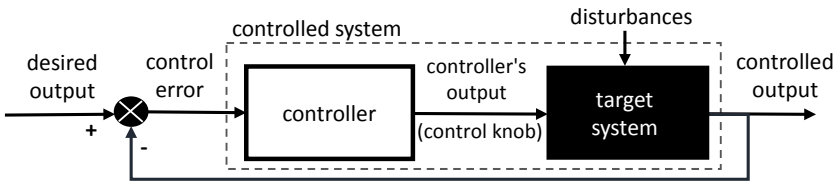


Figure 5.3: The standard feedback control loop

performance and the resource utilization as two decision making criteria results in better performance and resource utilization. The proposed hybrid memory controller is elaborated in Section 5.5.

5.3 Overview of the Control-Theoretical Design Process

As previously discussed in the background chapter, from the *software engineering perspective* a self-adaptive system is designed based on the MAPE loop reference model, while from the *control engineering perspective* an adaptive system is realized via a feedback control loop. In this chapter, we take a control theoretical perspective to design a controller for realizing vertical memory elasticity of cloud applications. The main motivation behind the choice of control theory in our work is to use this well-established theory for modeling and designing feedback loops to make the cloud applications self-adaptive and achieve a proper balance between fast reaction and better stability.

In the context of control theory, a standard feedback control loop is what is shown in Figure 5.3. The system which is being controlled is labeled *target system*, and the combination of the controller and the target system is labeled *controlled system*. It has a *desired output* that needs to be achieved by tuning a *control knob*. The control knob affects the *controlled output* at each control interval. In other words, the controller periodically adjusts the control knob in such a way that the controlled output can stay close to the desired output. The controller aims to maintain the difference between the desired and the controlled output (referred to as the *control error*) close to zero, in spite of the *disturbances* in the target system. The disturbance affect the controlled output, while not being controllable [194].

In the remaining two sections of this chapter, we develop two memory controllers for cloud applications exploiting the control design process proposed by Antonio Filieri and Martina Maggio, et al. [70]. This process includes six steps that one should follow to develop a self- adaptive system (e.g., a cloud application in our work) with control-theoretical guarantees. These steps are depicted in Figure 5.4 and are explained in the following section, for the design of a vertical memory controller.

1. **Identify the goals**. The first step is to identify the measurable goals that the controller needs to achieve. In our work, we consider cloud infrastructure as the host of interactive applications, which face variable workloads at runtime. Each application has an SLA that stipulates a target performance expressed as *mean response time*.

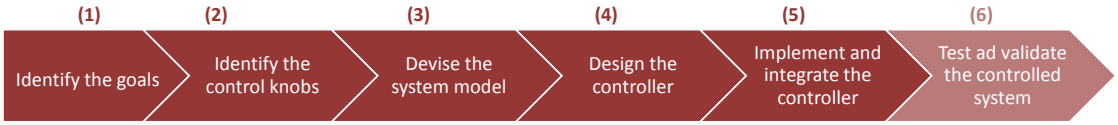


Figure 5.4: Steps of the control design process [70]

The controller goal is to continuously adjust the host cloud infrastructure without any human intervention, so that to drive the applications' performance toward their targets. Based on Figure 5.3, the application response time is the desired output, while the target system is a cloud application. Since the performance is the main goal of the controller, recalling the solution model of Section 5.2, such a controller would be either a performance-based controller or a hybrid controller.

2. **Identify the control knobs**. The second step is to determine the control knobs that can change the behavior of the target system. To find a suitable control knob, we run the motivation experiment previously presented in Section 5.1. Where we show that the application performance (i.e., the controller's goal) is dependent on the size of the allocated memory of the VM-hosting the application. Hence, the control knob in our work is the VM memory. Considering the identified control goal (step 1) and the control knob, the controller should continuously adjust the allocated memory of applications to meet the applications' performance objectives. Specifically, the desired controller should be capable of allocating just the right amount of memory for each application at the right time, avoiding resource under- or over-provisioning.

3. **Devise the system model**. The next step of the control design process is to devise a system model for the controlled system. In general, this model should capture the relationship between the control knob identified in step 2 and the controller's goal identified in step 1. Therefore, the system model should represent the relationship between the allocated memory (control knob) and the desired response time (controller's goal). In our work, we use linear regression analysis [161] as a statistical process for estimating these relationships. Building the system model is explained separately for the two proposed controllers at Sections 5.4.2 and 5.5.2.

4. **Design the controller**. To design a controller various, techniques can be used that are different in the level of required information to set up the control, and the guarantees they offer [70]. In this chapter, we loosely follow the control synthesis technique proposed in [68]. The main reason of using such a technique is to reduce the need for a strong mathematical background to devise ad-hoc control solutions. This step is explained separately in Sections 5.4.3 and 5.5.3 for the two memory controllers of this chapter.

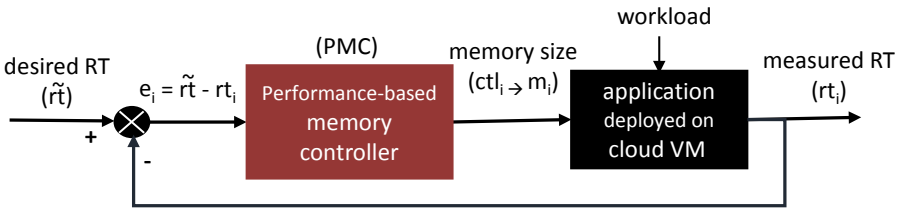


Figure 5.5: The feedback control loop of the performance-based memory controller

5. **Implement and integrate the controller**. After designing a controller, it is needed to be implemented and integrated with the target system. As mentioned, our target system is a cloud application, hence in this step the memory controller is integrated with the cloud application, and the actuator that applies for adjusting the control knobs is implemented. In our work, the controllers are implemented on top of the hypervisor and the actuator is the hypervisor API for adjusting the allocated VM memory. This step is explained separately in Sections 5.4.4 and 5.5.4 for the two controllers.

6. **Test and validate the controlled system**. The last step includes testing and validating the controller. This can be divided into two main parts. First, one needs to test the controller itself and check if it works correctly, i.e., by formal assessment of the controller's properties. This part is presented in Section 5.6 for the proposed controllers. Once the controller itself is verified, it can be tested as an integration with the target system by means of running experiments. This part is discussed in details in the evaluation chapter (Sections 7.3 and 7.4).

While steps 1 and 2 have been already defined for the controllers designed in this chapter, in the following sections, we explain the details of steps 3–5 for the proposed *performance-based memory controller* and *hybrid memory controller* (Section 5.5).

5.4 Performance-based Memory Controller (PMC)

In this section, we first give an overview of the proposed *performance-based memory controller*, and then explain the process of devising the system model, designing and implementing the controller.

5.4.1 PMC Overview

To realize the controller, we design a feedback control loop depicted in Figure 5.5 based on the standard feedback loop (Figure 5.3). The controller's output at each control interval i is named ctl_i and is mapped to the memory size m_i as the control knob. The desired RT ($\tilde{rt}$) and the measured RT (rt_i) are equivalent to the desired output and controlled output of standard feedback loop, respectively. The control error (e_i) is the difference between these two values at each interval, as shown in Equation (5.1). The

changes in the number of requests and their patterns are considered as disturbances. Since the controller has no control over the workload, it has to adjust the resources in order to meet the desired RT.

$$e_i = \tilde{rt} - rt_i \tag{5.1}$$

5.4.2 Devising PMC System Model

The aim of this step is to find the relationship between the memory size, as the control knob, and the response time, as the controlled output. The model building is started when the controller passes a minimum number of control intervals to be able to gather enough information to calculate the system model parameter α that is used in the control designing step. To find the suitable value for α the controller traverse a range of $0 < ctl < 1$ at each interval, starting from the beginning of the range (e.g., 0.01) to the end (e.g., 0.99). The corresponding clt and rt values of each interval are stored to be used later in extracting the α at the end of this step. At each control interval, ctl_i is mapped to the VM memory size $m_i \in [m_{min}, m_{max}]$ using Equation (5.2).

$$m_i = ctl_i \cdot (m_{max} - m_{min}) + m_{min} \tag{5.2}$$

Note that based on Equation (5.2), the continuous output $ctl_i \in (0, 1)$ is mapped to a discrete value of memory size m_i expressed by the number of memory units m_{unit}. The memory unit m_{unit} is defined as a discrete block of memory, e.g., 64 MB. The influence of increasing or decreasing memory can be set to be more significant with a larger memory unit m_{unit} and less significant with a smaller m_{unit} depending on the target system. The calculated memory size m_i at each interval is used to adjust the size of VM memory, and then the influence of the memory change is reflected in the application rt_i of the current interval. At the end of this step, i.e., when $ctl_i \geq 1$, the system model parameter for PMC is captured by applying the linear regression analysis on the stored values of ctl and rt. As emphasized in [68], this model is not necessary to capture the exact relationship, but a rough estimation is enough to tune the controller for the control design step.

State machine presented in Figure 5.6 shows the states on which the controller can traverse. The initial state (i_0) is representing the system model devising step. As shown in Figure 5.6, as long as $ctl_i < 1$ the controller is still trying to capture the system model.

5.4.3 Designing PMC

At the beginning of this step, the system model parameter (α) has been extracted. By having α, the controller is able to track the desired RT by rejecting the influence of workload fluctuation on the measured RT at each control interval and to keep the control error as low as possible. The control formula, as originally devised in [68], is presented in Equation (5.3).

$$ctl_i \;=\; ctl_{i-1} \;-\frac{1 - pole}{\alpha} \;\cdot\; e_i \tag{5.3}$$

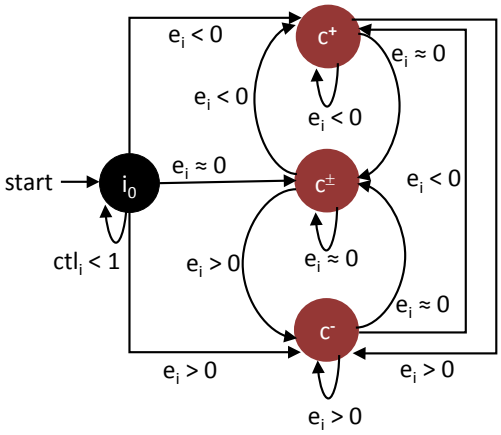


Figure 5.6: The state machine of the performance-based memory controller

Where the controller's output ctl_i is calculated based on its previous value ctl_{i-1} and a coefficient of the control error (Equation (5.1)). The coefficient is based on the value of system model parameter α, and an input parameter *pole*.

The choice of *pole* value influences the stability of the system and determines how fast the system approaches to its equilibrium. Based on the formal assessment provided in [68], the stability of the synthesized controller is ensured as long as $pole \in [0, 1)$. The value of *pole* trades responsiveness, i.e., how fast the controller reacts, and robustness in the face of noise. In other words, it determines how aggressive the controller can handle the control error and reject disturbances at runtime. The higher the value of *pole*, the less aggressive *ctl* value is changed at each control interval, so it leads to having longer settling time, namely the time which is needed for the controlled system to reach its desired output.

In order to develop a more stable and robust controller, we use two error smoothing techniques in PMC. First, we apply the weighted moving average (WMA) filtering technique [23] used in time series analysis on calculating the control error. This method weights the history of the errors by a series of weighted decreasing factors. A weighted factor close to one gives a large weight to the first samples and rapidly makes old samples negligible. The main purpose of using WMA is to calculate the average of the last few samples and not the average of all samples. Based on the chosen weighted factor, the average can be more or less sensitive to changes (the lower the value, the more sensitive WMA is for peaks). Second, we utilize the idea of a technique used in machine learning named support vector regression (SVR) [30]. Instead of conducting the control error to zero, SVR tries to keep errors within a margin defined by a threshold representing an ignorable error. The value of this threshold depends on the sensitivity of the desired output for the user. If the user is very sensitive, a lower value should be assigned as the SVR threshold.

Recalling the state machine of Figure 5.6, the controller can go to one of the three

possible final states (denoted by c^+, c^-, and $c^\pm$) based on the value of the control error e at each control interval. c^+ (*increasing* state) presents a situation in which the measured RT is larger than the desired RT (i.e., $rt_i > \tilde{rt}$), so consequently, based on Equation (5.1), $e < 0$. This situation indicates that the application deployed in the VM needs more memory, so the controller should increase the amount of the allocated memory. That is why this state is denoted by a plus superscript. c^- (*decreasing* state) is exactly the opposite situation of c^+, where the controller should decrease the memory of the VM to avoid resource wastage. $c^\pm$ represents a *no-change* state in which the control error is insignificant, so the desired RT and measured RT are close enough and there is no need for a change. These three states are the final states of the state machine, so the controller can stop at any of them.

At each control interval, the controller's output $ctl \in (0, 1)$ is mapped to a memory size $m_i \in [m_{min}, m_{max}]$ using the previously introduced Equation (5.2). Then, the calculated memory size m_i at each interval is used to adjust the size of VM memory, and then the influence of the memory change is reflected on the measured RT and also control error of the current interval.

It is worth mentioning that the value of the control interval should be set considering several factors such as: (i) the delay between applying the controller's output on the target system and observing its influence on the controlled output; (ii) the effect of the controller's output on the stability of the target system; (iii) the nature of the disturbances on the controlled-system; (iv) the sensitivity of the user on the desired output. Generally speaking, the control interval value should be short enough to make the controlled system reactive and long enough to observe the effects of the new memory allocation on the application's performance [112]. Based on our conducted experiments, presented in Chapter 7, using a short control interval results in oscillations of the measured RT, since the controller reacts too quickly. On the other hand, using a long interval reduces the oscillations, but reduces the responsiveness of the controller. Therefore, we empirically examine the controller with a different control interval in a boundary of values and select the best interval accordingly. Note that there is no general guideline for deriving a control interval and the best interval varies for different workloads and applications.

5.4.4 Implementing and Integrating PMC

Performance-based memory controller is implemented in `Matlab R2014a`. In order to apply the controller's output on the target system, we need an actuator. In our implemented feedback control loop, the KVM hypervisor memory API[6] uses as the actuator to set the allocated memory of the VM hosting the application. In KVM, live configuration of the allocated memory for a VM is possible within a negligible delay. On the other hand, the measured RT should be measured at each interval. In our implementation, we use *httpmon* as a load generator and also a monitoring tool. The integration of the controller, *httpmon*, and the KVM API is realized in `Java` and is executed on a Linux server with `Java SDK 1.7`.

[6]KVM memory API: `virsh setmem`

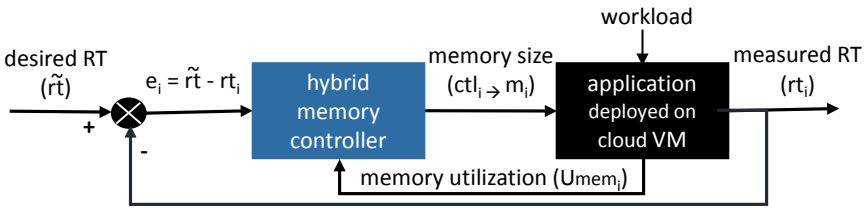


Figure 5.7: The feedback control loop of the hybrid memory controller

As the last step of the introduced control design process (Section 5.3), to test and validate the proposed *performance-based memory controller* on handling unexpected workloads, we run an experimental study using RUBBoS as the benchmark application deployed in a virtualized environment using KVM hypervisor under two sets of real-world workload traces (from Wikipedia [10] and FIFA WorldCup websites [5]). The experimental results are discussed in Section 7.3 of the evaluation chapter.

5.5 Hybrid Memory Controller (HMC)

In this section, a novel hybrid approach based on the presented solution model presented in Section 5.2 is proposed to realize the vertical memory elasticity. This approach takes into account both the application performance and the resource utilization to leverage the benefits of both performance-based and capacity-based approaches. We use control theory to synthesize a feedback controller that meets the application performance objectives by dynamically adjusting the allocated memory. Different from *performance-based memory controller* presented in Section 5.4, the novelty of the proposed *hybrid memory controller* is utilizing both the memory utilization and application response time as decision making criteria, and using an adaptive system model parameter that can be updated at runtime.

In what follows, first an overview of HMC is explained, then the process of devising the HMC system model, and its design is elaborated. In other words, steps 3–5 of the presented control design process (Section 5.3) are explained, respectively.

5.5.1 HMC Overview

From the control theoretical perspective, the designed feedback control loop for *hybrid memory controller* looks as shown in Figure 5.7. It is similar to the designed feedback loop of *performance-based memory controller* (see Figure 5.5). However, the hybrid controller takes memory utilization (U_{mem_i}) as the second input parameter.

Figure 5.8 shows the architecture of the proposed *hybrid memory controller*. It loosely follows a MAPE loop (monitoring, analysis, planning, execution phases). The monitoring phase gathers information such as the observed response time, average memory utilization from the application and the host virtual machine at each control interval. During the analysis phase, the memory required by the application to meet its performance objective

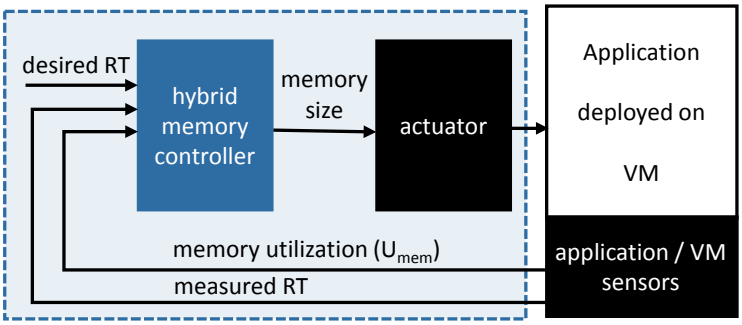


Figure 5.8: The architecture of the hybrid memory controller

is computed using the proposed controller. The goal of the controller is to allocate the right amount of memory in order to meet the application performance objective. The proposed memory controller determines the amount of memory that should be allocated using the application RT and VM memory utilization as decision making criteria. The system model parameters are captured based on the stored monitoring data. Finally, during the planning and execution phases, hypervisor is configured to enforce the computed memory in the VM hosting the application. In the following, the main functionality of the components depicted in Figure 5.8 are briefly described.

- **Hybrid memory controller**. It is an adaptive controller that dynamically tunes the amount of memory required for the application using the values of measured RT given by *application sensor* and the desired RT as well as the value of the memory utilization (U_{mem}) given by *VM sensor*. It is called adaptive since it dynamically keeps its system model updated at runtime.
- **Sensor**. This component gathers the application- and VM-level real-time performance information consisting *mean response time* and the average memory utilization of the application and the hosting VM at each control interval.
- **Actuator**. At each control interval, the controller invokes this module which is the Xen API for memory allocation, to either increase or decrease, the allocated memory of the VM hosting the application at runtime.

While *sensor* and *actuator* are introduced later in Section 5.5.4, the design details of *hybrid memory controller* are discussed in the following sections.

5.5.2 Devising HMC System Model

Similar to PMC, for HMC the system model parameter α represents a first order model of the reaction to the controller's output. However, the system model of HMC can be updated at each control interval. It is calculated by applying the linear regression technique based on the effect of *ctl* on *rt*. The model building is started when the controller passes a minimum numbers of control intervals to be able to gather enough information to calculate and update the value of α, e.g., 25 control intervals used in

our experiments, before that a default value is used as the α (10 in our experiments). Moreover, to improve the stability of the controller, we again apply the weighted moving average filtering technique on the gathered values before calculating the value of α at each control interval. The benefit of updating α (used at HMC) at runtime compared to a static α (used at PMC) is the ability of the controller to better withstand the workload changes at runtime. In other words, this way the controller is able to have a more realistic model of the system throughout the application lifespan at runtime.

5.5.3 Designing HMC

To synthesize *hybrid memory controller*, we improve the control formula previously used in *performance-based memory controller* (see Section 5.4.2). As shown in Equation (5.4), the controller's output ctl_i is calculated based on its previous value ctl_{i-1} and a coefficient of the control error. Different from PMC, the error coefficient here is based on the three parameters α, *pole*, and a new parameter β.

$$ctl_i \;=\; ctl_{i-1} \;- \frac{1 - pole}{\alpha_i} \;\cdot \beta_i \cdot \; e_i \tag{5.4}$$

The parameter β is a function of memory utilization $U_{mem_i} \in [0, 1]$ of the VM hosting the application and it is calculated dynamically at each control interval. The rationale behind having such a parameter is to have more insight into the current memory required by the application before increasing or decreasing the memory size. β is defined as shown in Equation (5.5).

$$\beta_i = \begin{cases} 1 - U_{mem_i} & e_i > 0 \\ U_{mem_i} & e_i \leq 0 \end{cases} \tag{5.5}$$

The first case, where $e_i > 0$ ($e_i = \tilde{rt} - rt_i$) means the measured RT rt_i is lower than the desired RT, $\tilde{rt}$, which can indicate an over-provisioning situation where the current allocated memory is more than enough for the application to process the current workload. Therefore, the controller should carefully decrease the memory to some extent to avoid over-provisioning while still maintaining $\tilde{rt}$.

However, based on our observation while performing the experiments, there is a period in which in spite of having a reasonable RT, memory utilization is getting very high. Such a period is when the allocated memory is near to its saturation point, but still the remaining memory is good enough for the application in response faster than the desired RT. In this situation, if the controller only reacts based on the control error, it would decrease the memory and this can cause a sudden increment in the measured RT. To avoid this situation, β is defined as $1 - U_{mem_i}$ when $e_i > 0$. Therefore, β is low when the memory utilization is relatively high, and this lessens the memory decreasing action.

On the other hand, $e_i \leq 0$ represents an overload condition when the application workload is high and more memory is needed to be able to meet the desired RT, so the controller should increase the amount of the allocated memory. Nevertheless, if the memory utilization U_{mem_i} is low it can indicate that memory is not the main reason of having a high RT, so the controller should be conservative on adding a large amount

of memory in this situation. Therefore, β is defined as U_{mem_i} and consequently this influences the change at the allocated memory.

Similar to PMC, the choice of *pole* determines the stability of the controlled system. The stability of the controller is ensured as long as $0 \leq pole < 1$ [68]. To develop a more stable and robust controller, we again apply *weighted moving average* filtering technique used in time-series analysis for calculating control error before using it in Equation (5.4). At each control interval, *hybrid memory controller* tracks $\tilde{rt}$ by rejecting the influence of workload fluctuation on rt_i and withstands the control error e_i as long as it is insignificant. Finally, the controller's output $ctl \in (0, 1)$ is mapped to a memory size $m_i \in [m_{min}, m_{max}]$ using the previously presented mapping formula in Equation (5.2), where m_{min} and m_{max} are the minimum and maximum amount of VM memory sizes expressed by the number of memory units m_{unit}, which are allowed to be allocated, and m_i is the final output of *hybrid memory controller*.

5.5.4 Implementing and Integrating HMC

Hybrid memory controller is implemented in `Java`. In our implemented feedback control loop, the Xen hypervisor memory API[7] uses as the actuator to set the memory of the VM hosting the application. Similar to KVM hypervisor, Xen is also able to adjust the allocated memory of a VM on demand within a negligible delay (less than a second). Two sensors are developed for HMC: the application-level sensor, and the VM-level sensor. At each control interval, the application-level sensor observes the average RT and the VM-level sensor measures the average of memory utilization. The VM-level sensor monitors the memory utilization statistics over a control interval by using `/proc/meminfo`, and reports the average value of them to the controller. These monitoring values are used as decision making criteria in *hybrid memory controller* for the next control interval. Both sensors send their information via transmission control protocol/Internet protocol (TCP/IP) connection to the controller. The integration of the controller, client, and the sensors are realized in `Java` and are executed on a Linux server with `Java SDK 1.7`.

Experimental evaluation of HMC, are discussed in Section 7.4 of the evaluation chapter. We conduct an experimental setup using RUBBoS application deployed on top of Xen hypervisor. We compare and report the results achieved by HMC with *performance-based memory controller* (presented in Section 5.4 and evaluated in Section 7.3) and a capacity-based controller [134]. We validate our approach using synthetic traces generated based on open and closed system models along with the real-world traces, i.e., Wikipedia and FIFA WorldCup website.

5.6 Formal Assessment of the Controllers' Properties

In this section, the theoretical assessment of the proposed controllers from the control point of view is discussed. From the perspective of control engineer, a controller should be able to provide the following four main properties [70]:

[7]Xen memory API: `xm mem-set`

1. **Stability**. A system is stable as long as it tends to reach an equilibrium point, regardless of the initial conditions. In other words, the system output converges to a specific value as time goes to infinity. This equilibrium point should ideally be the desired output value.
2. **Absence of overshooting**. An overshoot happens when the controlled system exceeds the desired output before convergence. Designing controllers in a way to void overshooting can decrease unnecessary costs. For example, in our case overshooting represents the SLA violation, i.e., when the measured RT > desired RT.
3. **Low settling time**. Settling time refers to the time required for the controlled system to reach the stable equilibrium. It can be guaranteed to be lower than a specific value when the controller is designed.
4. **Robustness to model inaccuracies**. A robust control system converges to the desired output despite errors or variations in the initial captured system model. This property defines how well the controlled system reacts to disturbances so that to make correct decisions with inaccurate measurements.

First, let's consider PMC in which the system model is static, so the values of α does not change at runtime. In this case, these four properties can be analytically guaranteed, based on the mathematical definition of the controlled system. Since the control formula used in PMC is the same as the one proposed and formally assessed in [68], so the control properties are given. The choice of *pole* between 0 and 1 guarantees stability, absence of overshooting, robustness (the closer the value of *pole* to 1, the more robust) and a settling time (the closer the value of *pole* to 0, the fastest). This allows the controlled system to trade-off robustness for settling time, which is usually done in control theory.

In the case of HMC, where we also consider adapting the system model of the controller, what changes is the weight that is given to the error. That is indeed changed by a factor that is given by the changes of β and α. Since moving average technique is used for the control error values, although α values are dynamic and updated at runtime, they are not changed too fast, so it does not affect the stability of the closed loop system. Actually, it even improves the convergence property. This is because by using an updated value of α, the system model is a more precise representation of what is happening at the current time. On the other hand, the value of β in the used control formula (Equation (5.4)) is always between 0 and 1, therefore, it affects the settling time but not the stability. Clearly, if a controller acts on less than the error that it experiences, it is slower in reacting in case the model is correct. However, it also increases robustness, because the controller would take smaller steps toward the satisfaction of the goals. No matter how β is changed, the fact that its value is between 0 and 1 makes it possible to state that stability is preserved, settling time is increased, but with an increase in robustness. This is preferable because apparently the model may not be very precise around some of the operating points, e.g., due to memory saturation or some other runtime situation that happens to software systems.

CHAPTER 6

Coordination between Elasticity Controllers

Cloud elasticity provides an application with the ability to maintain a specified QoS level by automatically adjusting the host infrastructure on the fly. The most common elasticity strategy is horizontal in which the number of allocated VMs is increased or decreased under varying workload at runtime. However, utilizing vertical elasticity strategy in which the size of individual virtual machines (resource-wise) is adjusted is beneficial to achieve a higher resource utilization. In other words, while the horizontal elasticity is course-grain in provisioning or deprovisioning resources, vertical elasticity is fine-grained; individual fractions of a resource such as a CPU core may be allocated in a virtual machine for as little as a few seconds [112]. In the last decade, most attention has been given to research on the area of horizontal elasticity in cloud computing, somewhat due to the limitation of vertical elasticity on scaling outside a single physical machine [163] and more due to the fact that it used to require the system rebooting while hypervisors only recently have started supporting live vertical resource scaling.

While vertical elasticity has been recognized as a key enabler for efficient resource utilization of cloud infrastructure, only little research has been done to support vertical elasticity [134] where the focus is mostly on a single resource, e.g., CPU vertical elasticity [144, 103, 112, 170], or memory vertical elasticity [29, 49, 181, 134] but not both at the same time. The underlying assumption made in these research efforts is that the application is either CPU-intensive or memory-intensive, while in reality, an application may need various types of resources at different stages of its execution. Besides, the application performance objectives (e.g., response time) are rarely considered [53, 3]. Nonetheless, the existing techniques cannot be readily used as-is without proper coordination since they may lead to either under- or over-provisioning of resources and consequently result in undesirable behaviors such as performance disparity. Therefore, vertical elasticity may be required for different resources of one or more application tiers.

In this chapter, research question V is addressed by presenting contribution V. We first propose a coordination model in which two control strategies, i.e., vertical and horizontal elasticity, can be applied to achieve the performance objectives of a cloud application (Section 6.1). Based on the proposed model, we introduce several abstract coordination policies that can decide which elasticity strategy should be applied at any runtime control interval. The main goal of each policy is to meet the performance objectives of a cloud application by either changing the number of acquired virtual machines, or changing the size of the individual virtual machines at each control interval. Motivated by a motivation experiment (Section 6.2) where we show an application can need multiple resource intensive at runtime. Finally, we propose a novel fuzzy coordination approach at Sections 6.3 and 6.4 as the main focus on this chapter. In this approach a fuzzy controller acts as the coordinator between a CPU vertical controller and a memory vertical controller in order to satisfy the application response time at runtime without over-committing any of the resources.

6.1 A Coordination Model for Elasticity Strategies

The core of any elastic system is a controller. Such a controller can be used for providing elasticity of cloud infrastructure. Due to the scale and complexity of cloud applications, several distributed elasticity controllers are required to be utilized. As previously discussed, control theory is a promising fit and reliable solution to be used in autonomic controlling and dealing with unpredictable changes in the systems by introducing a principled way of designing feedback control loops. Each control loop can change the infrastructure based on its own control action without considering the consequences of other actions that has been made by other controllers.

In case of a cloud application, the coexistence of the horizontal elasticity strategy and the vertical elasticity strategy can lead to a better resource utilization and application performance. This can achieve by using the horizontal elasticity strategy to control the number of acquired VMs allocated to a cloud application, while vertical elasticity strategy is used to adjust the allocated resource (e.g. CPU, memory) of the VM. In this case, each control action comes with different costs. While acquiring or releasing VMs, as control actions of the horizontal elasticity strategy, can be more effective to handle the dramatic changes of workload at runtime, they may be unnecessary as it is more expensive and slower to be enacted in comparison with control actions of the vertical elasticity strategy. Indeed, increasing or decreasing the amount of allocated resources within a single VM, although may not be sufficient in some situations, is adequate to handle overload in many occasions and can avoid the performance degradation very quickly and with a much lower cost. Such a statement is also valid in an underload scenario when deciding which control action is more beneficial to be enacted, whether decreasing the size of a resource within the VMs or releasing the VMs.

This coexistence of horizontal and vertical controllers brings up the need for coordination in order to avoid in coherent control actions. For example, when a performance degradation is detected, horizontal controller may try to acquire more VMs, while at the

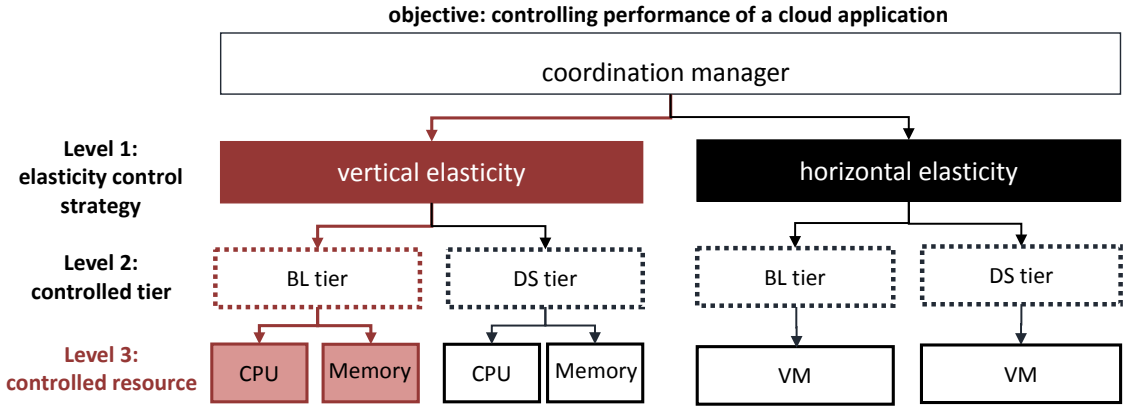


Figure 6.1: The coordination model for elasticity strategies

same time vertical controller is adding resources to the existing VMs. Nevertheless, due to either different control actions, or conflicting objectives, uncoordinated autonomic loops may lead to global sub-optimal or inconsistent states of a cloud application [52]. This section aims to initiate a research direction toward supporting the coordination of distributed autonomic controllers (elasticity controllers in our research) that all act toward achieving a common goal. Proposing effective coordination approaches can pave the way for autonomic control of large-scale and complex systems by using distributed controllers. In the remaining of this section, we first present a coordination model for elasticity control, and then briefly introduce several coordination policies that can be applied for orchestrating the control actions of the vertical and horizontal elasticity strategies.

6.1.1 Model Overview

The coordination model of elasticity strategies is depicted in Figure 6.1. The objective of this model is to control the performance of a cloud application by utilizing different elasticity control strategies (see Level 1 in Figure 6.1) applied to an appropriate application tier, labeled as controlled tier (see Level 2 in Figure 6.1). In other words, multiple tiers of a cloud application may be involved in processing user requests, thus, they may require different cloud resources, including VMs, CPU cores, memory, or a combination of these (see Level 3 in Figure 6.1). Therefore, horizontal or vertical elasticity may be required to apply on the VMs hosting different application tiers. In the remaining of this section, several policies for coordinating the vertical and horizontal elasticity strategies (see Level 1) are introduced.

6.1.2 Coordination Policies

Horizontal elasticity has been widely adopted by commercial clouds due to its simplicity as it does not require any extra support from the hypervisor. However, due to the static

nature of the horizontal elasticity in terms of the fixed VM size, applications cannot be provisioned with arbitrary configurations of resources based on their demands.

This usually leads to either inefficient resource utilization or SLA violation since the demand cannot always exactly fits the size of the VM. To efficiently utilize resources and avoid SLA violations, horizontal elasticity should be accompanied with a fined-grained resource allocation, applying the vertical elasticity strategy, where the VM sizes can be dynamically adjusted to an arbitrary value according to the runtime demands.

In this section several coordination policies are proposed, where each policy orchestrates the actions made by the horizontal and vertical elasticity controllers at each control interval.

- **Only vertical strategy**. In both overload and underload conditions, adjusting the size of the resource (e.g., CPU or memory) of the VMs hosting the cloud application is considered as the control action of the vertical elasticity controller. In case of considering the application of the control action solely on a single resource, this policy is similar to the two proposed vertical memory controllers of Chapter 5. In case of taking multiple resources scenario and the application of multiple vertical resource controllers, this policy also cover the fuzzy coordination approach that is presented in Section 6.3. Such a policy is effective for handling predictable and slow changes of workload at runtime, where the performance degradation can be compensated very quickly and with a much lower cost by changing the size of the VMs.
- **Only horizontal strategy**. Based on this policy, in both overload and underload conditions, the control actions of the horizontal strategy, adding or removing VMs, are applied to the infrastructure that hosts the cloud application. This policy also needs to balance the load among the VMs after applying the appropriate control action at runtime. Although this policy is more expensive and slower than the previous one, it is more suitable for handling dramatic change of workload at runtime.
- **Promoting horizontal strategy**. While in this policy both elasticity strategies can be used at runtime, it favors the control actions of the horizontal strategy in both overload and underload situations. More specifically, whenever the horizontal strategy gets to its maximum numbers of VMs which are allowed to be acquired at overload, or the minimum numbers of VM(s), which can be released under underload condition, the vertical strategy is applied as an alternative elasticity strategy.
- **Promoting vertical strategy**. In contrast to the previous policy, this one promotes the control actions of the vertical strategy for both over and underload conditions, i.e., increasing or decreasing of the allocated memory for the VMs. Whenever the maximum or minimum amount of resource (e.g., CPU or memory) is allocated, then the horizontal strategy is used as an alternative control strategy.
- **Promoting horizontal strategy in overload, but vertical strategy in underload**. Based on this policy, the horizontal strategy is preferred for overload conditions, while the control action of the vertical strategy is promoted to be

applied in underload conditions. The rationale behind this policy is to avoid the performance degradation by applying a more effective elasticity strategy (horizontal) at overload, while at underload trying to slowly decrease the size of the VMs.

- **Promoting vertical strategy in overload, but horizontal strategy in underload**. This policy is quite opposite to the previous policy in which the vertical strategy first tries to handle the overload conditions, since it is quicker and costs less. In the case of reaching the maximum capacity of the allocated resource for the VMs, then the horizontal strategy is used. The situation is reversed under underload conditions.
- **Prediction-based**. This policy uses a proactive approach for predicting the cloud application workload at runtime. Each elasticity strategy has its own advantages and disadvantages. While the horizontal elasticity is more costly and slower than vertical elasticity, it can be the only effective solution for handling the long-term and dramatic changes of workload. Therefore, using the proper strategy at the right time can lead to lower cost and achieving performance objectives. Since the booting time of the VMs is not instant and it varies between 60 to 600 seconds, while the workload contains many short duration spikes, the horizontal elasticity is only effective if the VM instances can be ready to use when they are needed to serve the workload.

  On the other hand, in the case of facing some long-term changes, using vertical elasticity, despite being fast in reaction, might not be sufficient and lead to SLA violation. Therefore, instead of making decisions based on current workload, in this policy, the workload trend is identified by employing a time-series forecasting technique to estimate the workload at the two future points in time, e.g., using double exponential smoothing [173]. If the slope between the two forecasted values are more than a specific threshold, the horizontal elasticity strategy is applied, otherwise, the vertical elasticity strategy is used.

The focus of the rest of this chapter is on the first introduced policy that allows the control actions of vertical elasticity strategy. In particular, we focus on the coordination of a CPU vertical controller and a memory vertical controller that adjusts the resources of the VM hosting the business logic tier of a cloud application. The red path of the proposed coordination model, Figure 6.1 shows the focus of this chapter. In the next section we experimentally motivate the need for such a coordination.

6.2 Motivation

As previously mentioned, during the lifespan of a running application, it may need various resources intensively depending on the nature of the workload. For example, a chat application may require techniques such as long-polling to immediately notify the user when a new message has arrived. Long-polling essentially delays the HTTP reply until there is an event to report, which in turn increases the number of connections on the server, hence increasing its memory requirements, without significantly increasing

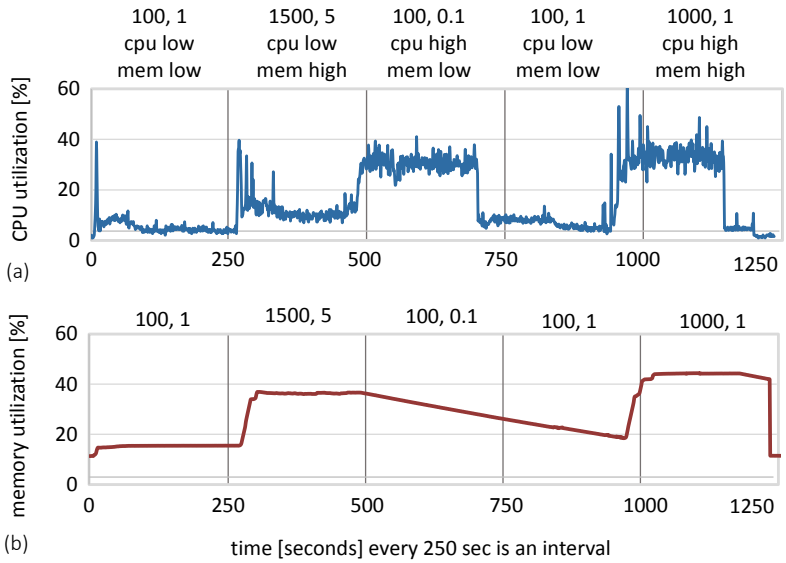


Figure 6.2: CPU and memory utilization of RUBiS under various workload patterns

CPU utilization. On the contrary, the chat application might include a *search in chat history* functionality, which is CPU intensive. Obviously, such an application needs to be scaled both CPU- and memory-wise. However, existing techniques cannot readily be used as-is for scaling multiple resources, e.g., both memory and CPU, at the same time, since uncoordinated control actions by different controllers may lead to inconsistent control actions. In this chapter, we propose a fuzzy coordination approach that dynamically coordinate two vertical controllers for CPU and memory to meet the performance objectives of a cloud application. In this section, we experimentally motivate the application need for multiple resources at runtime.

To show an application may require different resource configurations at runtime, we conduct an experiment on the RUBiS benchmark application [12] by injecting variable workloads at different time intervals, which induce the intended behavior. We deploy the BL and DS tiers of RUBiS on different VMs and over-provisioned both VMs (the VM hosting BL tier: 8 CPU cores and 4 GB memory and the VM hosting DS tier: 6 CPU cores and 10 GB memory). Then, by configuring a workload generator tool, `httpmon`, we define variable workload dynamics which put stress on the VM hosting the BL tier for either CPU, memory, or both resources during the application lifespan.

Figure 6.2 (a) and (b), respectively, depict the experimental results regarding the CPU utilization and memory utilization of the VM hosting `Apache` Web server as the representative of the BL tier. We set different configurations in `httpmon` to emulate different combinations of CPU or memory demands using a closed system model. To this aim, we vary the number of concurrent users and *thinktime* at each interval (i.e., every 250 seconds) shown as two values at the top of Figure 6.2, respectively. High *thinktime* resembles the case where many clients are doing long-polling, while high concurrency resembles the case where many clients are intensively interacting with the BL tier.

Therefore, changing the number of concurrent users and *thinktime* is a way to emulate variable CPU and memory requirements for the application. For example, to emulate an application with high CPU and low memory needs, one can set both the number of concurrent users and the *thinktime* to relatively low values (e.g., as in the first interval). On the other hand, to induce high memory and low CPU characteristics, one can set high values for both the number of concurrent users and the *thinktime* (e.g., as in the second interval).

As observed in Figure 6.2, there are some intervals where RUBiS needs less of both resources (the first and fourth intervals), more memory and less CPU (the second interval), more CPU and low memory (the third interval) as well as more CPU and memory (the last interval). The results clearly show that the application may need an arbitrary combination of these resources during its execution. Thus, an approach that supports the elasticity of multiple resources, in this case CPU and memory, to meet the application performance objectives needs to be designed. In the remaining of this chapter, a fuzzy coordination approach is proposed to address this need. Notice that the proposed solution targets applications that can benefit from live memory and CPU elasticity at runtime, i.e. application with a dynamic memory and CPU requirements.

6.3 Fuzzy Coordination Approach: Coordinating CPU and Memory Controllers

In this section, we first give an overview of the proposed approach, then briefly explain the two vertical controllers which are used in the proposed fuzzy coordination approach.

In order to address the deficiencies of the existing approaches, and to support the vertical elasticity of both CPU and memory while meeting the desired response time, in this chapter we propose *fuzzy coordination approach* that allocates the right amount of both resources. *Fuzzy coordination approach* is composed of three sub-controllers. The first is *fuzzy controller* (Section 6.4) designed based on fuzzy logic to infer the degree of contributions of CPU and memory to applications' performance changes. The other two controllers, are responsible for determining of the right amount of resources using the resource demand indicators generated by *fuzzy controller*. In general, *fuzzy controller* acts as a coordinator by looking at application's resource demand indicators such as the average CPU and memory utilization and response time so that the control actions of the vertical controllers complement each other to fulfill the application need.

6.3.1 Approach Overview

We consider a cloud infrastructure that hosts interactive applications, each with variable workload dynamics. Each service has an SLA that stipulates a desired value expressed as *mean response time*. The goal is to continuously adjust the allocated resources of applications without any human intervention, so as to drive applications' performance toward their objectives. Specifically, the desired *fuzzy coordination approach* should be capable of allocating just the right amount of resources for each application at

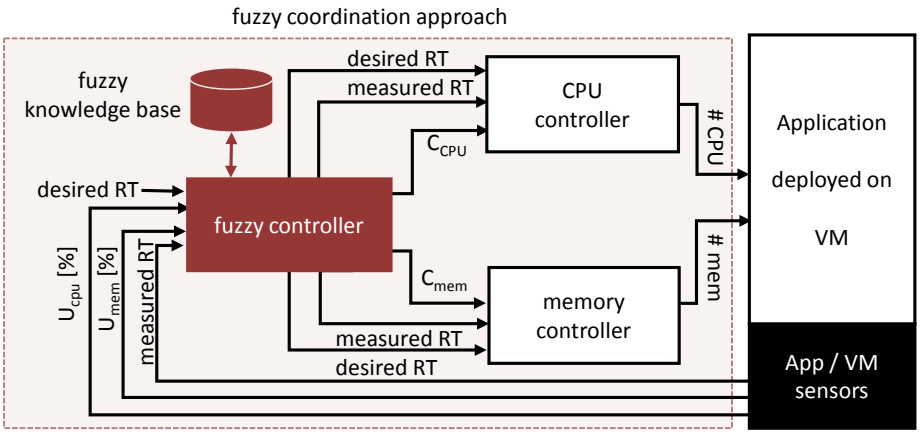


Figure 6.3: The architecture of the fuzzy coordination approach

the right time in order to meet its performance objective, avoiding both under- and over-provisioning.

Figure 6.3 shows the architecture of the proposed *fuzzy coordination approach*. The modules in red indicate the contribution of our work. This architecture loosely follows a MAPE loop (consisting of monitor, analysis, plan, execution) based on self-adaptive software terminology, introduced previously. Monitoring gathers information such as the observed RT, average CPU and memory utilization of the hosted infrastructure at each interval. During analysis, the resources required by an application are computed using the controllers in two steps.

In the first step, *fuzzy controller* infers the extent of the contributions of both CPU and memory to applications' performance change. More specifically, it generates a value $\in$ [-1,+1] indicating the degree of severity that the CPU and memory has on the performance of the application. A value close to -1 indicates resource over-provisioning, while a value close to 1 shows under-provisioning situation.

In the second step, using the values generated by *fuzzy controller*, the CPU and memory controllers, respectively, determine the number of CPU core(s) and the amount of memory that should be allocated using the application RT as a decision making criterion. Previous monitoring data is used to fit the model parameters, which are similar to the knowledge component of autonomic controllers, shown in Figure 6.3. Finally, during the planning and execution phase, hypervisor is configured to enforce the computed resources. A high level function of each component depicted in Figure 6.3 is described as follows:

- **Fuzzy controller**. It determines the coefficient with which CPU, memory, or both are responsible for the application performance change. Based on the fuzzy rules specified in the engine, it reasons about the coefficient values using the current state of the system. The output of *fuzzy controller* consists of two coefficients values, one is corresponding to the CPU and the other is related to memory, which indicates either to increase or decrease CPU, memory, or both (See Section 6.4).

- **CPU controller**. An adaptive controller that dynamically adjusts the CPU capacity that should be allocated to an application based on the values of measured and desired RTs as well as the CPU coefficient (C_{cpu}) received from *fuzzy controller*. It allocates the right amount of CPU cores for the application in order to guarantee its performance objective (See Section 6.3.2).

- **Memory controller**. It is an adaptive controller, similar to *cpu controller*, that dynamically tunes the amount of memory required for each application using the measured and desired RT values as well as the value of the memory coefficient (C_{mem}) given by *fuzzy controller*. The memory controller used in *fuzzy coordination approach* is *hybrid memory controller* previously presented in Chapter 5 (see Section 5.5 for more details), where a static value is assigned to the memory utilization parameter (β=1) in the control formula Equation (5.4). This way the controller decides solely based on the application response time at runtime.

- **Sensor**. This component gathers both the application- and VM-level performance information, including *mean response time*, average CPU utilization, and average memory utilization of the application and the allocated VMs, periodically. We refer to this period as the control interval. These monitoring values are used as decision making criteria in *fuzzy controller* for the subsequent control interval.

In the following, *cpu controller* is explained with more details, and *fuzzy controller* as the core of *fuzzy coordination approach* is elaborated.

6.3.2 CPU Controller

CPU controller is based on the work presented in [112]. The inverse relationship between *mean response time* rt_i of an application and the number of CPU cores cpu_i allocated to at each control interval is modeled as Equation (6.1).

$$rt_i = \beta / cpu_i \tag{6.1}$$

Where β is a model parameter. The parameter β can be estimated at runtime from the past measurements of mean response time and allocated CPU cores. To reduce the impact of measurement noise, we use a recursive least square (RLS) filter [120]. RLS is an adaptive filter which recursively finds the coefficients that minimize a cost function (control error in our case) relating to the input parameters. In essence, such a filter takes past estimations of β as well as the current product of $rt_i * cpu_i$ in order to generate a new value that minimizes the sum of the squares of the errors made in the results of every single calculation. In order to trade the influence of old values for up-to-date measurements, a *forgetting factor* of 0.45 is used. Note that in our experiments, based on the available resources, we define a boundary for the minimum and the maximum amount of allocated CPU.

6.4 Fuzzy Controller Design

Due to the non-deterministic behavior of software systems, it is almost impossible to know with a high degree of confidence the extent of the contributions of different resources to performance degradation of a software application and how much of each resource should be provisioned to alleviate the performance problem. Moreover, the measured data used as decision making criteria in the process of resource allocation such as RT, CPU and memory utilization may include sensory noise. If a coordinating approach does not pay attention to such uncertainties, it may cause the oscillations in resource allocations [96, 61]. To address these issues, we utilize fuzzy control [114] as it provides a means to reason about uncertainties using highly expressive languages where the treatment of uncertainty and approximate reasoning is performed in a natural and efficient way. With this in mind, in this section, we explain how we developed the fuzzy logic system (FLS) which is responsible for coordination and reasoning of the *fuzzy coordination approach*.

One of the most well-known applications of fuzzy logic is fuzzy control [114], in which the controller decides based on the defined fuzzy rules. A typical fuzzy *IF-THEN* (rule 6.2) is expressed as:

$$R: \quad \text{IF} \quad \underbrace{x_1 \text{ is } F_1 \ldots \text{ and } x_p \text{ is } F_p}_{\text{antecedent: input variables and fuzzy linguistic terms}} \quad \text{THEN} \quad \underbrace{y_1 \text{ is } G_l \ldots \text{ and } y_q \text{ is } G_q}_{\text{consequent: output control variable}} \tag{6.2}$$

Where antecedent is compound of a number of input variables, and the consequent is composed of a number of output control variables. In the case where there are multiple input and output variables, similar to our case, the system is called multi-input multi-output (MIMO) fuzzy system. In our approach, fuzzy rules are based on a data collection approach from a group of technical experts who has enough knowledge on cloud resource elasticity and application performance modeling.

6.4.1 Elasticity Reasoning using Fuzzy Logic System

A fuzzy knowledge-base has the information on how best scale the target system in terms of a set of linguistic rules (i.e., rule (6.2)). In our fuzzy logic system, average RT, average CPU utilization (U_{cpu}), and average memory utilization (U_{mem}) are the input variables, while CPU coefficient (C_{cpu}) and memory coefficient (C_{mem}) are the output control variables. Therefore, the fuzzy system used in our research is a MIMO FLS. In this work, the linguistic terms representing the values of the input variables are divided into three levels. For response time they are: slow (S), medium (M), and fast (F). Similarly, for CPU utilization U_{cpu} and memory utilization U_{mem} they are: low (L), medium (M), and high (H). As mentioned, the designed FLS consists of two output control variables, C_{cpu} and C_{mem}. These values indicate two numbers $\in [-1, 1]$ as the degree of severity that the CPU and memory effect on the application performance change at each control interval.

6.4.2 Extracting Fuzzy Rules

In general, for a fuzzy logic system, there are two ways to design a fuzzy knowledge-base including fuzzy rules and membership functions (MFs): (i) collecting data from the system behavior; (ii) using human experience. In our work, we follow the first based on the approach presented by Jamshidi et al. [96] to extract the fuzzy rules from the experts and identify the MFs and then we empirically update these values by carefully monitoring and analyzing the behavior of the system. In other words, our fuzzy knowledge-base is constructed based on the rules which are systematically obtained from the experts. To design the fuzzy rules, we collect the required data by performing a data collection among seven experts who have deep knowledge on cloud resource allocation and performance modeling. We prepare several questions to extract the required knowledge, such as the following sample question:

$$R^{25}: \quad \text{IF } \underbrace{RT \text{ is } slow \text{ and } U_{cpu} \text{ is } high \text{ and } U_{mem} \text{ is } low}_{\text{antecedent}} \\ \text{THEN } \underbrace{C_{cpu} \text{ is } +1 \text{ and } C_{mem} \text{ is } \text{-}0.3}_{\text{consequent}} \tag{6.3}$$

The experts are asked to determine the consequent (output control variables) using a number $\in [-1,1]$ for each fuzzy rule. We initially use the mean values from the experts' responses for our experiments, and then we empirically tune some of the rules based on analyzing the behavior of *fuzzy controller*. The final fuzzy rules used in our experiments are presented in Table 6.1.

6.4.3 Constructing Membership Functions

A membership function defines the degree of truth with a value between 0 and 1. In a fuzzy logic system a membership function is defined for each linguistic term used for presenting each input variable. In our FLS, there are three linguistic terms for each input variable, so in total we need to define nine MFs. We use both trapezoidal and triangular membership function types, as shown in Figure 6.4. We used trapezoidal MFs to represent "Low" ("Fast"), and "High" ("Slow"), and triangular MFs to represent "Medium". Similar to the fuzzy rules, we initially ask the experts to locate an interval $\subseteq$ [0,100] for each linguistic term used as input variable, and then we tune the extracted intervals empirically. The MFs are presented in Figure 6.4 for each input variable. While CPU and memory utilization do not need further normalization since the utilization for each resource is expressed in percentage, response time value should be mapped to a value $\in [0, 100]$. In order to do so, we normalize the measured response time with respect to a reference value as a coefficient of the desired RT. Measured response time values closer to this reference which are further up away from the target value are set to a value close to 100. On the contrary, the response time values further down are set to a value closer to 0 depending on the magnitude of the value relative to the reference.

fuzzy rule (R)	antecedents			consequent (mean value)	
	RT	Ucpu	Umem	Ccpu	Cmem
1	Fast	Low	Low	-1.0	-1.0
2			Medium	-1.0	-0.6
3			High	-1.0	0.0
4		Medium	Low	-0.9	-0.9
5			Medium	-0.5	-0.7
6			High	-1.0	0.0
7		High	Low	0.1	-1.0
8			Medium	0.1	-0.7
9			High	0.1	0.1
10	Medium	Low	Low	-0.9	-0.7
11			Medium	-0.8	-0.5
12			High	-0.8	0.5
13		Medium	Low	-0.6	-0.6
14			Medium	-0.9	-0.5
15			High	-0.9	0.5
16		High	Low	0.5	-0.8
17			Medium	0.4	-0.5
18			High	0.5	0.5
19	Slow	Low	Low	-1.0	-0.3
20			Medium	-1.0	0.7
21			High	-1.0	1.0
22		Medium	Low	0.6	-0.2
23			Medium	0.5	0.5
24			High	0.4	1.0
25		High	Low	1.0	-0.3
26			Medium	1.0	0.5
27			High	1.0	1.0

Table 6.1: The extracted fuzzy rules

input variable [range]	linguistic term	mean value		standard deviation	
		start (a)	end (b)	start (σ_a)	end (σ_b)
response time [0,100]	Fast	0	32	0	17
	Medium	24	66	13	12
	Slow	59	100	23	0
CPU utilization [0,100]	Low	0	41	0	17
	Medium	35	74	19	7
	High	68	100	24	0
memory utilization [0,100]	Low	0	45	0	22
	Medium	39	75	23	13
	High	69	100	27	0

Table 6.2: The fuzzy data regarding the used linguistic terms

triangular MF	trapezoidal MF
left = (a - σ_a) , 0)	lower left = (a - σ_a , 0)
middle = ((a + b) / 2), 1)	upper left = (a , 1)
right = (b + σ_b , 0)	upper right = (b , 1)
	lower right = (b + σ_b , 0)

Table 6.3: Locations of the main points of the used membership functions [96]

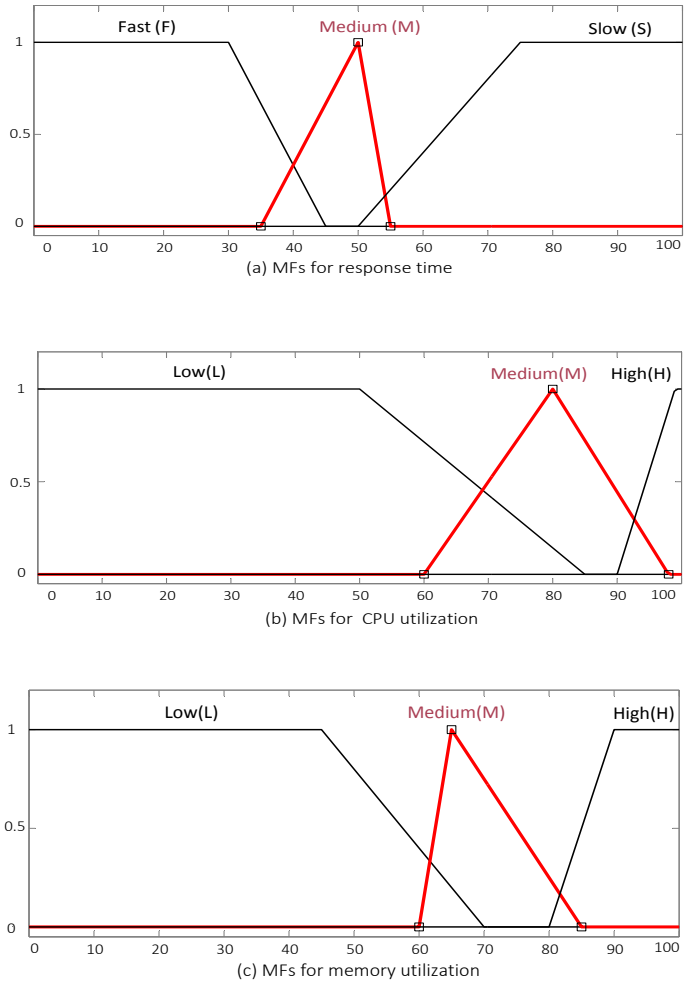


Figure 6.4: Membership function of each linguistic term for the three input variables

6.4.4 Reasoning with the Fuzzy Controller

Having designed the fuzzy logic system with the membership functions and the fuzzy rules, the controller can then perform the elasticity reasoning. *Fuzzy controller* works based on the following steps: (i) the measured values of input variables (i.e., RT, U_{cpu}, and U_{mem}) are first fuzzified using the defined MFs (shown in Figure 6.4); (ii) the FLS inferences and reasons according to the given fuzzified input variables using the designed fuzzy rules (Table 6.1) to produce the outputs consisting C_{cpu} and C_{mem}; (iii) the output control variables, C_{cpu} and C_{mem}, are fed into *cpu controller* and *memory controller*, respectively, to compute CPU cpu_i and memory mem_i for the next control interval. Figure 6.5 (a) and (b) show the outputs of *fuzzy controller* (C_{cpu} and $C_{mem} \in$ [-1, 1]) results in a hyper-surface corresponding to all possible normalized values of the input variables (RT, U_{cpu}, and $U_{mem} \in$ [0,100]). Notice that the three inputs are modeled

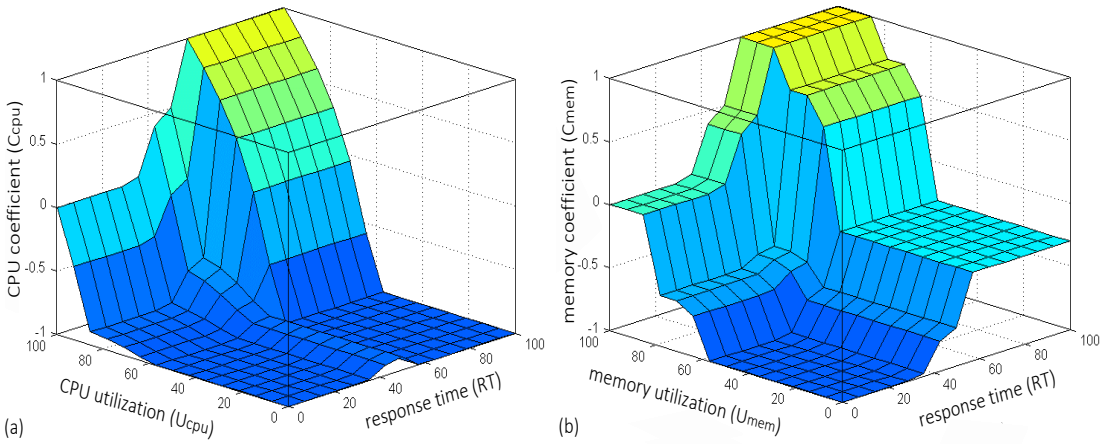


Figure 6.5: The fuzzy controller's output variables according to the input variables

in two the separated diagrams each with two inputs, for the sake of visibility. These diagrams reveal a more conservative behavior of *fuzzy controller* for controlling the memory allocation in comparison with CPU due to the fact that the application may crash as a result of memory shortage.

As shown in Table 6.2, we receive seven different intervals from the seven experts. We then calculate the mean and deviation values of the end points as presented in Table 6.2. We use these values to construct the MFs based on the guideline given in [96]. In this table, consider a and b as the mean values of the linguistic term end points, while σ_a and σ_b are the standard derivations of them correspondingly. The locations of the main points of each MFs are calculated based on the formulas presented in Table 6.3, using the values in Table 6.2.

6.4.5 Coordination Process

CPU and memory are allocated to each virtual machine with seamless coordination of the two controllers. *Fuzzy controller* generates coefficients that signify the extent to which each resource is needed by the application. Using CPU coefficient C_{cpu} and memory coefficient C_{mem}, the other two controllers determine the actual amount of each resource that should be allocated to meet the application's performance objective. Afterwards, *cpu controller* and *memory controller* first compute the values of cpu_i, and mem_i. Finally, both controllers apply the coefficient values to calculate the respective resource using Equations (6.4) and (6.5). In these equations, mem_i and cpu_i are the outputs of *memory controller* and *cpu controller*, respectively, while CPU_i, and MEM_i are the final amount of resources that should be allocated at the control interval i. Indeed, *fuzzy controller* mitigates the change on the number of CPU cores or the amount of memory, as the direct outputs of *cpu controller* and *memory controller*, at the current interval relative to the previous one by looking at the utilization of both resources as well as the measured RT at the same time.

$$CPU_i = CPU_{i-1} + C_{cpu} \cdot |cpu_i - CPU_{i-1}| \tag{6.4}$$

$$MEM_i = MEM_{i-1} + C_{mem} \cdot |mem_i - MEM_{i-1}| \tag{6.5}$$

To clarify the reasoning behind values of C_{cpu} or C_{mem}, a brief highlight is provided. Zero value for C_{cpu} or C_{mem} implies that the performance objective is met, thus, there is no need to change resources and the actions made by *cpu controller* and *memory controller* are bypassed. A value of 1 or -1 for C_{cpu} or C_{mem} indicates that the values computed by these two controllers should be fully allocated to the VM hosting the application. However, when $-1 < C_{cpu} < 1$, or $-1 < C_{mem} < 1$, the actual CPU core(s) or memory size allocated to the VM is proportional to the respective coefficient values produced by *fuzzy controller* instead of allocating the amount of resources computed by these controllers directly. In general, a positive coefficient value indicates the need for additional resources, while a negative coefficient value indicates that the need to reduce resources.

6.4.6 Implementation and Integration

The proposed *fuzzy coordination approach* and its included *cpu controller* are implemented in `c++`, while *fuzzy controller* is implemented using `fuzzylite` library [7]. *Memory controller* is implemented in `Java`. The application- and VM-level sensors are both implemented in Linux `shell scripts`. Both sensors send their information via TCP/IP connection to *fuzzy controller* at each control interval. The Xen API for CPU and memory are used as actuators for *cpu controller* and *memory controller*, respectively. The integration of the three controllers, client, actuators, and the sensors are realized in `c++` and are executed on a Linux server with `Java SDK 1.7`.

The proposed approach is evaluated in an experimental setup using three different interactive benchmark applications under workload traces generated based on open and closed system models. The results are compared with a baseline approach which supports vertical memory and CPU elasticity. The evaluation results are discussed in Section 7.5 of the evaluation chapter.

CHAPTER 7

Evaluation

In this chapter, we present the evaluations of the contributions proposed in the scope of this thesis, elaborated in Chapters 3 to 6.

Section 7.1 covers the evaluation of the proposed cost-aware VM placement approach used for managing geographically distributed data centers (explained in Chapter 3). To this aim, we first introduce the simulation setup, and then discuss and compare the achieved results of the proposed approach with the results of the two baseline algorithms.

In Section 7.2, we follow a simulated-based evaluation of the proposed multi-cloud service selection approach (presented in Chapter 4) by modeling a realistic environment using a CAD application as the use case scenario. Since prospect theory as the used theory in our approach, is the alternative of utility theory, we compare and discuss the results of the multi-cloud service selection algorithm with the results achieved by a utility-based selection algorithm.

Sections 7.3 and 7.4 present the experimental evaluation of the two proposed vertical memory elasticity controllers, elaborated in Chapter 5. First, the experimental setup of each controller is explained. It then follows with a detailed discussion of the results achieved by each controller based on the time-series and aggregate analysis in comparison with the baseline controllers. RUBBoS as an interactive benchmark application and a set of synthetic and real-world workload traces are used for the experiments.

Section 7.5 reports the experimental evaluation of the proposed fuzzy coordination approach (presented in Chapter 6). A thorough experimental evaluation is performed and the results are discussed, using the three different interactive benchmark applications, RUBBoS, RUBiS, and Olio [9], under workload traces generated based on open and closed system models.

#DCs	#PMs	#VMs	**Bronze** (penalty)	**Silver** (penalty)	**Gold** (penalty)
5	25	25	0.05$/% violation	0.1$/% violation	0.2$/% violation

Table 7.1: CloudNet configuration parameters

policy name (abbreviation)	**definition of the policy**
last workload (BN-LW)	next value is estimated equal to the last one
trend workload (BN-TW)	next value estimation follows a certain linear trend
linear regression WL (BN-LRW)	applying linear regression on the historical data

Table 7.2: Workload prediction policies used in the VM placement approach

7.1 VM Placement across Geographically Distributed Cloud Infrastructures

In this section, the proposed VM placement approach (contribution I), presented in Chapter 3, is evaluated using the designed Bayesian network based on the data extracted from the real-world traces.

7.1.1 Setting up the Simulation Framework (CloudNet)

To evaluate the proposed algorithms based on the regional electricity prices and temperature differences, we first configure the previously introduced simulation framework, *CloudNet*, with the values presented in Table 7.1. In our simulation, we have one virtual machine type (1000 MIPS, 768 MB of memory) and one physical machine type in terms of the resource capacity (3000 MIPS, 4 GB of memory). Furthermore, each VM has an availability-related SLA metric which is defined as the time of the overall downtime per billing period. Note that the used billing period is one month in our evaluation. As shown in Table 7.1, our approach supports three SLA priority levels, namely *Gold*, *Silver*, *Bronze*, with different SLA penalty costs.

In order to evaluate the proposed algorithms, we consider a combination of various real-world time- and location-dependent inputs such as electricity price, power outage statistics, cooling models, and temperature (introduced previously in Chapter 3). Conducting the evaluations, we simulate one month (Jan-Feb 2013) operation of running data centers, geographically distributed in five locations (i.e., Brazil, Canada, Norway, Austria, Japan), with the management interval of one hour.

7.1.2 Baseline Algorithms

The proposed VM placement approach use the three previously workload prediction policies, summarized in Table 7.2. The proposed approach is evaluated in comparison with the two baseline approaches: (i) a heuristic first fit decreasing (FFD) approach that supports both VM allocation and migration. To get comparison results with FFD approach, we use it under two resource allocation policies. The first policy, *first fit*

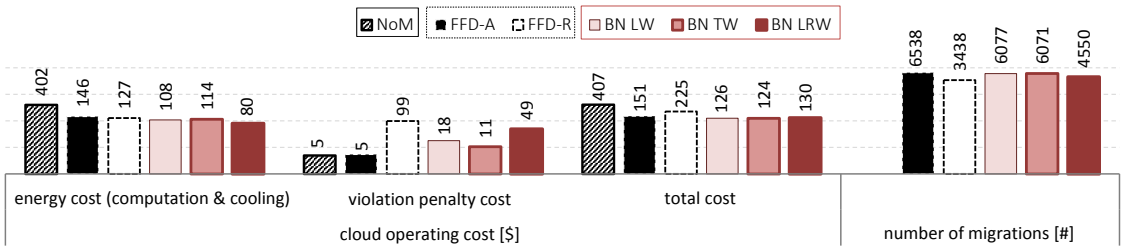


Figure 7.1: Aggregate results throughout one month of simulated evaluation

decreasing agreed (FFD-A), statistically allocates the amount of resources agreed by the SLA, while the second policy, *first fit decreasing requested* (FFD-R), is more dynamic in allocating the amount of resources required by the VM at runtime; (ii) an approach named NoM that follows a first fit VM allocation strategy, without supporting the VM migration. The rationale behind having such a baseline is to show whether the migration strategy can have an inverse effect on the VM placement in terms of SLA violation.

All approaches have the same set of input parameters and simulated configurations. The goal of the evaluation is to show the ability of the proposed approach to use the designed Bayesian network model which includes the extracted information about the cloud infrastructure in order to perform more efficient decisions concerning placement of VMs across geographically distributed data centers. An approach is said to be better if the total cost, including the energy cost and the penalty cost of the SLA violation is minimized.

7.1.3 Cost and QoS as Evaluation Metrics

The following metrics are considered as the evaluation metrics that represent the cost and QoS aspects of each approach:

- **Energy cost** presents the total operating costs, including the computational cost and the cooling cost of all data centers.
- **SLA violation penalty cost** is a cost that has to be paid by the cloud provider in the case of SLA violation. We define an SLA violation on a case that the VM unavailability exceeds a certain time threshold throughout a billing period.
- **Number of migrations** that represent the migration actions triggered during the evaluation whether to avoid the SLA violation due to the power outage or due to the consolidation. In general, the migration action should be applied by a cloud provider in a case that it has an appreciable impact on the operating costs. A lower value of this metric is preferable if it has a negative effect on the SLA violation and consequently the penalty cost.

7.1.4 Aggregate Results

Figure 7.1 shows the aggregate evaluation results obtained during the whole simulation run. The usage of the proposed approach (reddish plots) leads to better results with

all the three used policies, while under BN-TW policy the proposed approach achieves the best result. BN-TW policy decreases the total cost by up to 69% (124$ vs. 407$) in comparison with NoM approach. This improvement is by up to 45% (225$ vs. 124$ total cost) in comparison with FFD-R, which has the lowest number of migrations, and by up to 18% (151$ vs. 124$ total cost) in comparison with FFD-A, which has a higher number of migrations.

The results reveal that the usage of a more enhanced prediction policy, e.g., linear regression (BN-LRW policy) in our approach, increases the cost efficiency in the terms of energy cost. However, it causes more SLA violations in comparison with the other used policies (i.e., BN-LW, BN-TW). Moreover, the results in the case of NoM approach in comparison with FFD and the proposed approach to highlight the necessity of supporting migration strategy while managing distributed data centers in order to decrease the operating costs. The reduction of operating costs is achieved because of taking into account the time- and location-dependent input parameters for the management decisions such as applying the migration actions at a suitable time of the day or to a proper data center location. Furthermore, since by using *CloudNet* we simulate data centers with frequent power outages, approaches like NoM not only suffer from a high energy cost, but also leads to a high number of SLA violation as they cannot handle situations like a power outage.

In comparison with the FFD approach (i.e., FFD-A and FFD-R), the proposed VM placement approach gains less energy cost while keeping the penalty cost under control in a way that the achieved total operating costs is less for all the three used policies, as shown in Figure 7.1. The reason is due to the utilization of the prediction workload policies, using the extracted knowledge of cloud management, and considering the input parameters such as power outage statistics of cloud data centers modeled as Bayesian networks, the effectiveness of multi-criteria decision analysis method applied on Bayesian network reasoning, and supporting different SLA models. In summary, the proposed cost-aware VM placement approach under all workload prediction policies achieves better results in terms of both operating costs in comparison with the two baseline approaches.

7.2 SLA-based Multi-Cloud Service Selection

In this section, we evaluate the proposed multi-cloud service selection approach (contribution II), explained in Chapter 4, in comparison with a utility-based selection algorithm. The evaluation setup along with the discussion of the results are presented in this section.

7.2.1 Simulation Setup

In order to model heterogeneous cloud service offerings in a multi-cloud environment, we simulate 12 commercial IaaS providers, each with one or multiple data centers distributed along different geographical locations (four continents). In our simulation environment,

parameter	definition	unit or range
IaaS type	$VM_{small/medium/large}$, Storage	-
Location	data center region	1-4
Reputation	provider reputation rank	1-10
Availability	up-time of service	% in three months
Response Time	time to fully receive an answer	sec
Throughput	download a few large files	Mb/sec
$Cost_{VM}$	leased $VM_{small/medium/large}$	$/one small per hour
$Cost_{storage}$	stored data	$/one GB per month
$Cost_{traffic}$	download rate	$/one GB per month

Table 7.3: Functional and non-functional parameters offered by IaaS providers

(millisecond)	**USA**	**Europe**	**Asia**	**Australia**
USA	25	150	250	100
Europe	150	25	150	200
Asia	250	150	25	500
Australia	100	200	500	25

Table 7.4: Latency matrix

each provider offers a set of functional and non-functional parameters as well as pricing models for the offered services. As depicted in Table 7.3, each provider either provides cloud storage, or cloud virtual machine with three sizes: small, medium, and large. The VM budget is given based on its computation units. One, two and four computation units are respectively assigned to the small, medium and large VM size. We assume that the computation units of different IaaS providers for the same VM size are similar. The pricing model for each provider is acquired by using the pricing model of the corresponding real-world IaaS provider, while the values of QoS attributes are gathered experimentally by running a set of cloud tests using *CloudHarmony* Web service[1] on the chosen public cloud providers using a single client hosted in Austria. This specification allows us to model a realistic simulation environment.

Furthermore, the reputation value for each provider is simulated considering the reputation ranking model introduced in [94]. Due to lack of free access to the information regarding the latency between different data centers (available on *CloudHarmony)*, a latency matrix are defined with synthesized values by considering the geographical distances, shown in Table 7.4. The latency between host services within the same data center is assumed to be 10ms.

Supported meta-SLA parameters as well as their aggregate function are presented in Table 7.5. In this table, budget is the customer willingness to pay for leasing the cloud infrastructure including the leasing virtual machine and storage cost, as well as the data

[1]CloudHarmony Web service: `http://cloudharmony.com`

SLA parameter	unit	definition	aggregate function
Budget	$	total cost	$Cost_{agg} = \sum$ {VM+storage+traffic}
Availability	%	up-time	$Ava_{agg} = \prod_{i=1}^{n} Ava(S_i)$
Throughput	Mb/s	downloaded data	$Th_{agg} = min_{i=1}^{n} Th(S_i)$
Latency	ms	latency	$Lat_{agg} = \frac{\sum_{i,j=1}^{n} W(S_{ij}).Lat(S_{ij})}{\sum_{i,j=1}^{n} W(S_{ij})}$
Reputation	1-10	providers' rank	$Rep_{agg} = \frac{\sum_{i=1}^{n} Rep(S_i)}{n}$

Table 7.5: Meta-SLA parameters and their aggregate functions

parameter	unit	standard	professional	enterprise
contract duration	hour	720	720	720
Budget for VM	(small) $/hour	0.07	0.1	0.14
Budget for storage	(1 GB) $/month	0.1	0.12	0.15
Budget for traffic	(1 GB) $/month	0.1	0.12	0.15
Availability	%	96	98	99.8
Throughput	Mb/sec	5	5	15
Latency	ms	100	100	40
Reputation	1-10	3	3	5

Table 7.6: Meta-SLA requested values for the three different software editions

parameter	sub-SLA$_{UI}$	sub-SLA$_{BL}$	sub-SLA$_{DS}$
Edition	sand./ prof./ enter.	sand./ prof./ enter.	sand./ prof./ enter.
Type	virtual machine	virtual machine	storage
Size	small/ medium/ large	large/ large/ large	100 / 500 / 1000 (GB)
Number	1 / 1 / 1	1 / 2 / 5	1 / 1 / 2
Location	-	-	Europe

Table 7.7: Sub-SLA requested values for the three different software editions

traffic cost based on the defined units. For the virtual machine, a cost unit is expressed based on the coefficient value of a small VM per hour, while for the storage and traffic, a cost unit is represented based on the coefficient value of storing or transferring 1GB data per month. The calculation of the total budget in our simulation is based on these three cost units as well as the duration of the contract.

As depicted in Table 7.5 (previously introduced in Chapter 4), for each non-functional parameter of meta-SLA, there is a corresponding aggregation function, which calculates the aggregate values of composite service non-functional parameters based on its constituent services. In the aggregate latency formula $W(S_{ij})$ is the connectivity value of each edge in the connectivity graph, shown in Figure 7.2. While $Lat(S_{ij})$ is gathered from the latency matrix, showed in Table 7.4, based on the data center geographical location of the included services of the composite service. In order to show the different aspects of

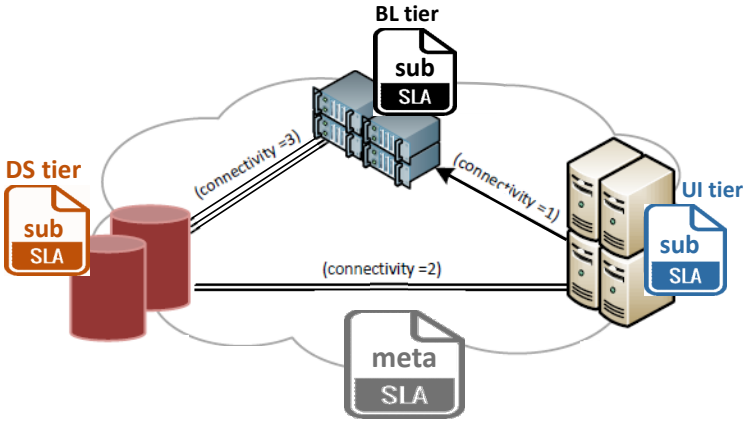


Figure 7.2: Connectivity graph of the CAD-aaS use case

our approach, we run the experiments of three sets of SLAs for three CAD-as-a-service software editions (standard, professional, and enterprise), each with different meta-SLAs and sub-SLAs as depicted in Tables 7.6 and 7.7, respectively.

Recalling the CAD-aaS use case (Figure 7.2) and its three application tiers, we divide the sub-SLAs into three logical groups. The first group of requested services belongs to the infrastructures that hosts the application UI, called sub-SLA_{UI}. The second group is related to the infrastructures that hosts the BL tier of the application for computing and rendering the images of CAD-aaS, called sub-SLA_{BL}. Finally, data storage tier is hosted in the cloud storage that stores and maintains the CAD models, called sub-SLA_{DS}. Although our approach is flexible enough to receive three different sets of sub-SLAs {sub-SLA_{UI}, sub-SLA_{BL}, sub-SLA_{DS}} for each software edition, we consider the same set for all editions. The rationale behind it is that each sub-SLA expresses the functional and non-functional requirements related to the application tiers (UI, BL, and DS). While it is possible to have different functional needs such as type, size and number of requested infrastructure services for each application tier, the non-functional requirements of various components in different software editions are almost the same.

7.2.2 Utility-based Selection Algorithm as the Baseline

In this section, we first introduce a utility-based algorithm [101] as the baseline approach. Then, we compare the features of the proposed approach, named as a prospect-based, with the so called utility-based baseline approach, the discussion of the evaluation results are given in the following sections.

The utility-based algorithm uses a quasi-linear utility-function adopted from the multi-attribute auction theory [25] to calculate the customer utility, taking the cloud customer leasing budget in focus. The utility for each composite service is calculated by subtracting the total service usage costs (including virtual machine, traffic, and storage) from its monetized usage benefit. The former is calculated for each customer by multiplying the overall score for the SLAs with the maximal payment for a perfect service.

factor	prospect-based algorithm	utility-based algorithm
cost impact	predefined	customer weight similar to QoS
weighting method	dependent W: $\sum W_{\text{QoS}} = 1$	independent W: $W_{\text{QoS}} \in (0, 1]$
selection strategy	maximize customer utility	maximize customer satisfaction
fitting function	customer value for each QoS	gain and loss (flexible scoring)
SLA support	only meta-SLA	meta-SLA and sub-SLA

Table 7.8: Comparison between prospect-based and utility-based algorithms

The overall SLA scores (a normalized value between 0 and 1), is defined as the sum of the weighted single SLA parameter scores, express the overall customer satisfaction for a certain service quality. For the calculation of the SLA score, the algorithm uses so called fitting functions that map each SLA metric value to a satisfaction level between 0 and 1.

A Theoretical Comparison: Utility-based vs. Prospect-based Algorithms

The key differences of the proposed prospect-based algorithm, with the utility-based algorithm is summarized in Table 7.8 and are explained as follows:

- **Cost impact**. The two algorithms have a different perspective on the cost. In the prospect-based algorithm, the cost has a flexible impact on service scoring based on the weight that each customer can assign, similar to other QoS parameters. While in the utility-based algorithm, the cost has a fixed influence with a predefined impact, which cannot be defined by each customer.
- **Weighting method**. The weighting model of these two algorithms is different. In the prospect-based algorithm, the weight of each QoS parameter is chosen within (0,1] independent of other parameters. However, in the utility-based algorithm the weights are dependent to each other within (0,1] and the sum of them should be 1.
- **Selection strategy**. The utility-based algorithm selects a composite service as the target, which maximize the customer utility. While, our algorithm selects a composite service that best satisfies both the meta-SLA and the sub-SLAs, i.e., maximize the customer satisfaction. In both algorithms, functional parameters must be fulfilled and both aim to find the best set of non-functional parameters. However, contrary to the utility-based algorithm, our algorithm can also support hard non-functional parameters that can be treated as the same way as the functional parameters.
- **Fitting function**. The fitting function used in the utility-based algorithm is quite similar to the satisfaction scoring function used in our algorithm. However, our algorithm is more flexible as it is based on the weights, while the utility-based algorithm needs to define separated fitting function for each QoS. Therefore, supporting more SLA parameters in our algorithm needs less effort.
- **SLA support**. The fitting functions in utility-based algorithm is only used to score the aggregated QoS based on the SLA for the composite service, while our algorithm

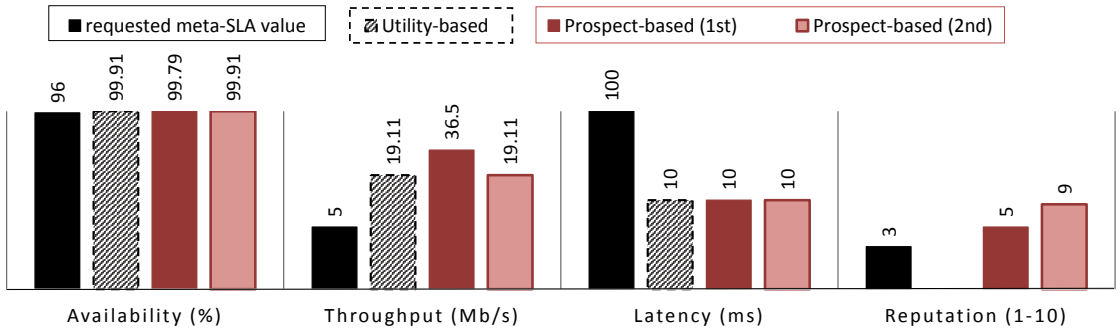


Figure 7.3: Meta-SLA of standard edition and aggregate results: QoS of selected services

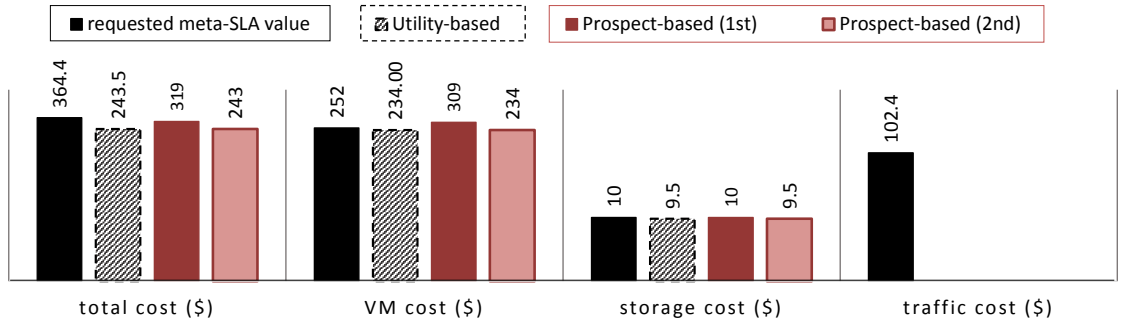


Figure 7.4: Meta-SLA of standard edition and aggregate results: Cost of selected services

supports scoring each single service based on the proposed concept of sub-SLA, and composite service based on the concept of meta-SLA using the satisfaction scoring function.

7.2.3 Evaluation Results

In this section, the service selection results achieved by both algorithms are discussed using several evaluation scenarios. The requested values for the meta-SLA and sub-SLAs parameters are shown in Tables 7.6 and 7.7, discussed previously. These values are different for three software editions of the CAD-aaS use case. The scenarios in this section are designed in a way to highlight the different aspects of the proposed approach.

Impact of cost

As the first scenario, minimizing the cost is the main objective of the customer and the QoS of the composite service is more important than the quality of individual services. To simulate this scenario, we assign the weights as: $W_{subSLA} = 0.1$ and $W_{metaSLA} = 0.9$, to dominate the influence of meta-SLA on sub-SLAs in the service selection algorithm. Furthermore, in meta-SLA, $W_{Cost} = 0.8$ and for other non-functional parameters $W_{NF} = 0.25$ to make cost as the most influential factor for selecting services.

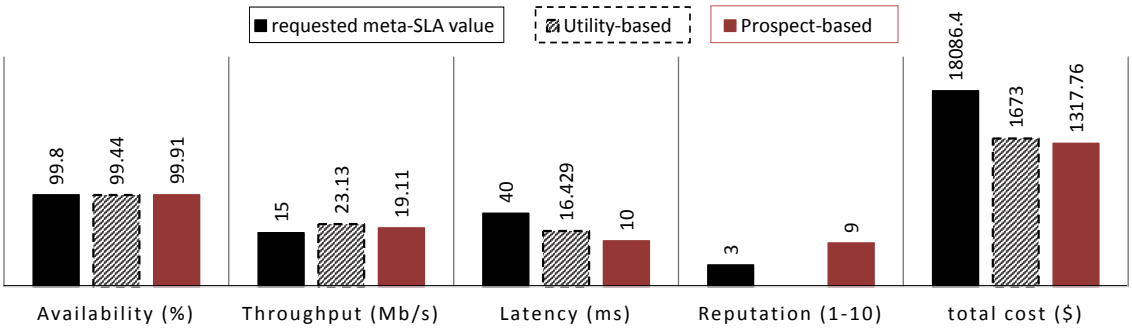


Figure 7.5: Meta-SLA of enterprise edition and aggregate results: QoS and total cost of selected services

Figure 7.3aggregate depicts the aggregate results of the achieved QoS values for the chosen services in the both algorithms. In this scenario, the selected services of the prospect-based algorithm are exactly the same as the selected services of the utility-based algorithm, i.e., both set of services are chosen from the same cloud provider and located at the same data center, so the achieved latency is 10ms, as shown in Figure 7.3aggregate. Furthermore, both algorithms are successful in satisfying the requested non-functional parameters of meta-SLA including cost (virtual machine, storage, and traffic), as illustrated in Figure 7.4.

The results of the first scenario demonstrate the tendency of both algorithms to select services from a single cloud provider data center, if this does not lead to SLA violations. The first ranked services of the prospect-based algorithm are chosen from AmazonEC2 (Europe data center) while the utility-based algorithm and our algorithm (the 2nd round) select all services from the Voxel[2] (Europe). This is reasonable, since the data traffic is free of charge and latency is negligible when all the services are located at the same data center. Note that in this scenario, the results achieved in the professional and enterprise editions are also similar to the achieved results of the standard edition.

Impact of meta-SLA

As the second scenario, we evaluate the proposed algorithm in a condition that the importance of a QoS parameter (availability) is equal to the importance of the cost to the customer, i.e., both have assigned by higher priorities compared to other non-functional parameters. We realize this scenario by setting the weight of availability more than the weights of other meta-SLA parameters (i.e., throughput, latency, and reputation) for the enterprise software edition. Moreover, in the utility-based algorithm, we set the weight equal to 0.5 for the availability and 0.25 for latency and throughput. By applying these settings, we achieve the results depicted in Figure 7.5. Our algorithm chooses AmazonEC2 (Europe), while the utility-based algorithm selected Voxel (Europe) for all requested infrastructure services. The plots of Figure 7.5 show that our algorithm

[2]Voxel IaaS provider: `http://www.voxel.net`

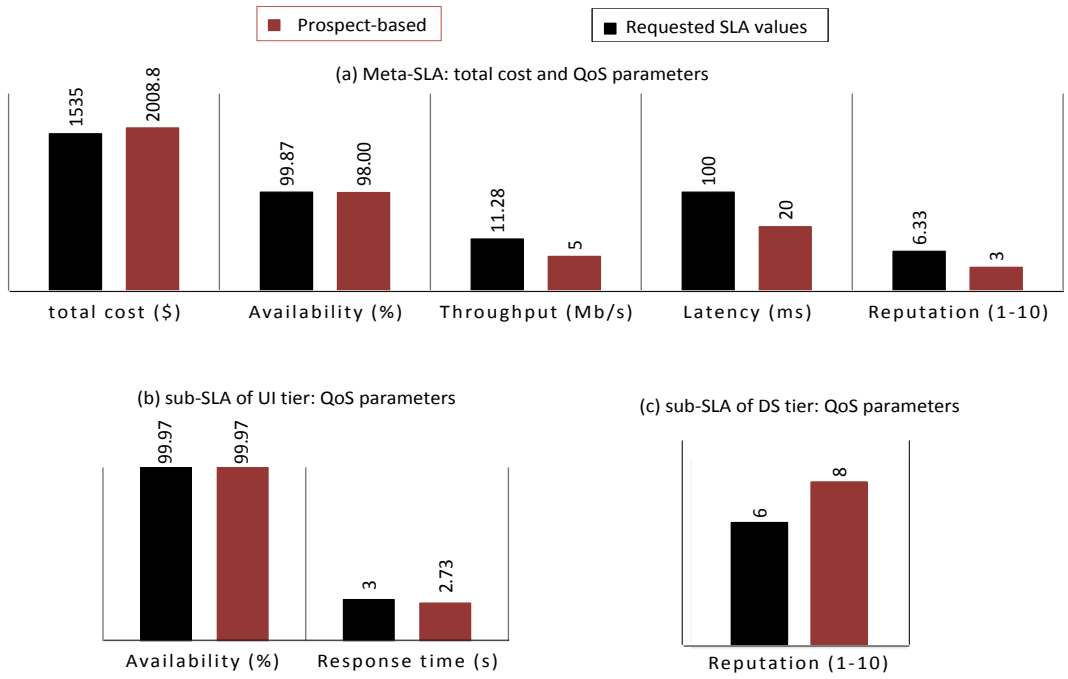


Figure 7.6: Meta-SLA, sub-SLA$_{\text{UI}}$, sub-SLA$_{\text{DS}}$, and aggregate results of professional edition: QoS and total cost of selected services

performs well in terms of: (i) not violating any SLA parameters; (ii) giving the priority to the parameters that are most important for the customer (cost and availability).

Moreover, our algorithm chooses the services that have the closest quality values to the requested values by the customer to maximize the satisfaction. This can justify the idea of applying prospect theory in the service selection. It other words, the best score is given to a service that more closely satisfy the requested SLA parameters with higher weights, contrary to the services that have the best quality. The throughout plot of Figure 7.5 demonstrates that the utility-based algorithm tries to maximize the values of throughput, while the requested availability is violated. This scenario shows that the utility-based algorithm focuses more on keeping the cost under the requested budget as the cost impact is not configurable in the algorithm. While in the prospect-based algorithm by assigning suitable weights and accepted boundaries for each SLA parameters, we can obtain the ranking score that maximizes the customer's satisfaction.

Impact of sub-SLA

As the last scenario, we consider that the CAD-aaS provider (i.e., the cloud customer) has specific non-functional requirements for each tier (UI, BL, DS) of the CAD application. The CAD-aaS provider wants to provide a professional software edition in which, for the security reasons, it is required to use cloud storage that: (i) the location of the data center providing the service is in Europe; (ii) the service is offered by a high reputed cloud provider (i.e., location = EU and reputation=6 in sub-SLA$_{\text{DS}}$). Furthermore,

the CAD-aaS provider wants to keep the response time of the deployed UI tier under 3sec, and makes it highly available (response time = 3sec and availability = 99.98% in sub-SLA_{UI}) with a high priority weight (0.75). The priority weights of other meta-SLA parameters are as default (W=0.5).

The plots shown in Figure 7.6 depict the comparison results between the QoS parameters requested in the meta-SLA and sub-SLAs, and the QoS values of the selected services (Rackspace[3] for deploying the UI and DS tiers, Citycloud[4] for deploying the BL tier both in Europe) by the prospect-based algorithm. As shown, the algorithm finds a suitable set of services that satisfies not only the requested meta-SLA parameters for the composite service, but also all the requested sub-SLAs' parameters are satisfied.

7.2.4 Discussion

From the first scenario, we conclude that although the impact of cost in the prospect-based algorithm is not a predefined like the utility-based algorithm, by assigning proportional weights to the service selection factors, the proposed algorithm can behave the same as a utility-based method which is a widely accepted cloud service selection from the efficiency point of view.

The second scenario, in which another QoS parameter is as important as the cost for the customer, shows that the prospect-based algorithm outperforms the utility-based algorithm. It selects a more suitable set of services which do not violate any requested meta-SLA parameters. Moreover, it chooses the services with the closest quality values to what are requested by the customer.

The third scenario highlights one of the main features of the proposed algorithm. The proposed algorithm not only tries to find an optimum composite service with accepted aggregated QoS values to satisfy the meta-SLA but also consider the sub-SLA satisfaction for the services that host the distributed application tiers. Nevertheless, existing service selection algorithms that support composite service selection, only focus on finding a set of services that satisfies the aggregated QoS parameters and neglect the individual included services. We can cover this need easily by the proposed sub-SLA concept, as represented at the third scenario.

To sum up, according to the evaluation results, when we are dealing with the quality parameters of a service as a whole, any algorithms may be adequate to find the most suitable services among the candidates. The main problem here is that in such a situation, the services provided by the same provider are preferred. This diminishes the benefits of a multi-cloud environment. Our algorithm comes into play when cloud customers want to take the advantage of being able to combine the services offered by different providers available in a multi-cloud. By introducing the sub-SLA concept, so the customer can defines more details regarding each individual service inside a composite service. Thus, as can be seen in the results, leads to a better customer satisfaction. It is worth mentioning that supporting parameters such as reputation is also significant in our work.

[3]Rackspace IaaS provider: `http://www.rackspace.com`
[4]Citycloud IaaS provider: `https://www.citycloud.com`

7.3 Performance-based Memory Elasticity Controller

In this section, we illustrate the experimental evaluation to show the potential of the implemented PMC, *performance-based memory controller*, explained in Chapter 5 as contribution III. In what it follows, we first describe the experimental setup, then we discuss the evaluation results.

7.3.1 Experimental Setup

Figure 7.7 depicts the overview of the experimental setup. The experiments are conducted on a single physical machine, Linux Ubuntu 14.04 server, equipped with 16 processors[5] and 32 GB of memory. To emulate a virtualized environment and perform vertical memory elasticity, we use KVM hypervisor. The used benchmark application is deployed in two VMs. As shown in Figure 7.7, VM_1 runs Apache 2.0 Web server with PHP enabled, and VM_2 runs the application database, MySQL.

Apache configuration. To emulate long connections that induce memory-intensive behavior on VM_1, as would be the case with techniques such as long-polling, we set the Apache parameters *keep alive timeout* to 10 seconds. To avoid VM_2 being a bottleneck, we provide sufficient memory and CPU cores for that during the experiments. Besides, we set our experimental setup in a way that there is no memory consumption limit for Apache running on VM_1 and the amount of allocated memory to this VM is dynamic and is managed by the controller. We use Apache *multi-processing module (MPM) prefork*, which is thread safe and therefore suitable to be used with PHP applications. We set the values of parameters regarding the Apache processes to relatively high values (2000 in our experiments), for example *MaxClients*[6] and *ServerLimit*[7]. These values need to be well above the number of concurrent requests that `Apache` has to deal with during any of the experiments in our work.

Benchmark application. We use RUBBoS, an open source, multi-tier benchmark application previously introduced in Chapter 5. It provides the essential bulletin board features of Slashdot site[8] and has been widely used in cloud research [49, 76]. It includes two tiers: business logic tier, which is a Web server using PHP server side scripts; a data storage tier that stores user information, stories, and related comments. The benchmark application includes two kinds of workload modes: browse-only and write interaction. We use browse-only workload in our experiments to more greatly put stress on the business logic tier by sending read-only HTTP GET requests.

Client. User interaction with RUBBoS is emulated using the previously introduced tool, *httpmon*, by sending or receiving HTTP requests. By utilizing the features of this tool, such as the ability to set the number of concurrent users at runtime, we configure *httpmon* so that in each interval, a specific number of concurrent users request random stories. To this aim, we dynamically set the *concurrency option* at runtime to

[5]Intel Xeon E31220
[6]Max number of child processes that are launched to serve requests
[7]Max value of MaxRequestWorkers for the lifetime of the Apache process
[8]Slashdot website: `http://slashdot.org`

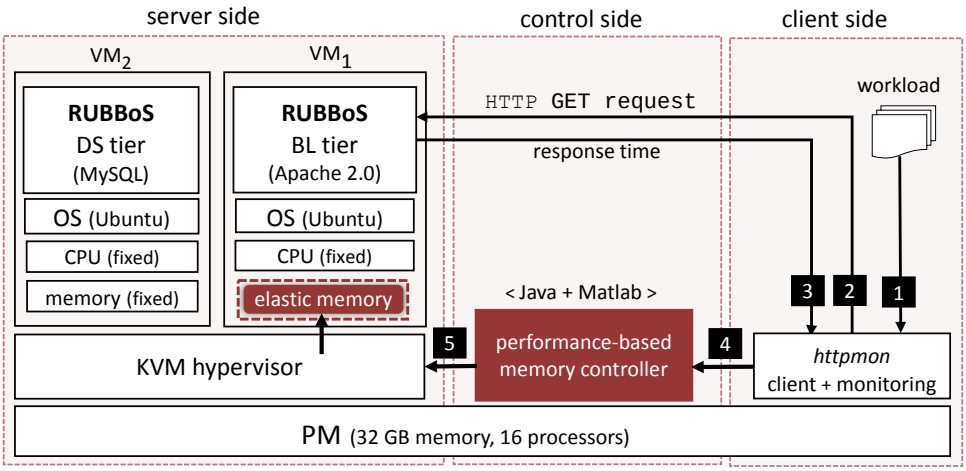


Figure 7.7: Overview of the experimental setup for evaluating PMC

emulate the used workload patterns. As for the monitoring, *httpmon* is also used in our experiments. A master thread at each interval collects data from the concurrent clients and periodically measures and sends statistics for the previously elapsed seconds of execution to *performance-based memory controller* via a TCP/IP socket.

Performance metric. In our experiments with RUBBoS, we focus on response time as a performance metric. Since users of such applications often base their Web experience consideration on how long it takes to browse stories, response time has a great influence on Web user satisfaction [108, 76]. In our scenario, response time is defined as the time difference between sending the first byte of the HTTP request to receiving the last byte for each user request. The mean, 95^{th} percentile, or maximum response time values can be also configured and used in *performance-based memory controller*, depending on the user requirements. In our experiments of this section, we use the *mean response time* as the performance metric.

Workload preparation. In our evaluation, we use Wikipedia and FIFA WorldCup website workload traces. Figure 7.8 depicts the scaled patterns of these traces; see the workload details in Table 7.9. Horizontal and vertical scaling of these workload traces are done according to the capacity of our experimental setup, while still keeping the original patterns. These two patterns then are mapped to the concurrent users that send HTTP GET request. To facilitate the reproducibility of our research, we release the source code (in `Python`) that prepares these two traces[9].

Experiment process. As shown in Figure 7.7, the experiment is started from the client side, *httpmon* tool, which is responsible for the load generating and monitoring. The server side is RUBBoS application controlled by the control side consists of *performance-based memory controller*. In this figure, the numbers indicate the sequence of the experiment process. After feeding the workload traces into *httpmon* (1), and based on the workload values at each control interval, *httpmon* emulates a specific number

[9]The preparation code of Wikipedia `http://goo.gl/iy36Pg`, FIFA `http://goo.gl/iUGF6O`

workload	duration	H-scale	V-scale	ratio	details
Wikipedia	24 hour	1000 sec	1000 users	10% of all req.	German language
FIFA	40 days	1000 sec	1000 users	all requests	WorldCup 1998

Table 7.9: Specification of the Wikipedia and FIFA WorldCup website workload traces

evaluation	workload	$\tilde{rt}$ (ms)	*pole*	$\text{interval}_{\text{control}}$ (s)	$\text{interval}_{\text{WMA}}$ (s)
experiment	Wikipedia	600	0.99	10	10
	FIFA	300	0.999	5	5
simulation	Wikipedia	500	0.5	10	10
	FIFA	300	0.999	10	20

Table 7.10: Configuration parameters for different evaluation scenarios

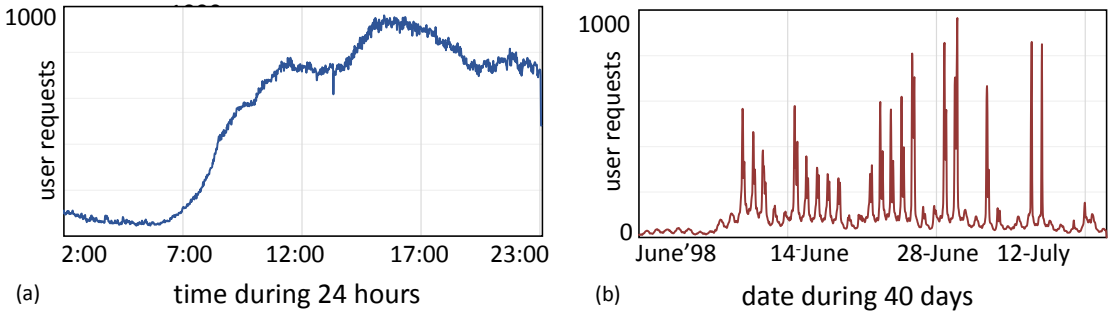


Figure 7.8: Wikipedia and FIFA WorldCup website workload traces

of concurrent users to send the HTTP GET requests to RUBBoS (2), and meanwhile measures RT of the sent requests (3). A control loop is triggered at this point after receiving the measured RT by *httpmon* at each control interval (4). The aim of our experiment is to evaluate the effect of memory elasticity on the application RT. Therefore, at each control interval, the controller calculates the control error and based on that, it measures the required memory to meet the application desired RT, and then it invokes the KVM memory API as the actuator to either increase or decrease memory size of VM_1 at runtime (5).

A thorough evaluation should be done under a variety of conditions, and then statistical analysis be applied to the achieved results. However, relying only on statistical results may hide details about the behavior of the system in different phases; therefore, in the following sections, we first explain the time-series analysis of the controller under different evaluation scenarios and then discuss the aggregate analysis to show the effectiveness of the controller. We compare the results of a self-adaptive RUBBoS (i.e., equipped with the controller) with two non-adaptive RUBBoS (i.e., without using the controller) under two provisioning policies: over-provisioning and under-provisioning. Table 7.10 shows the experimental setting parameters for all the scenarios, including the desired RT, *pole*

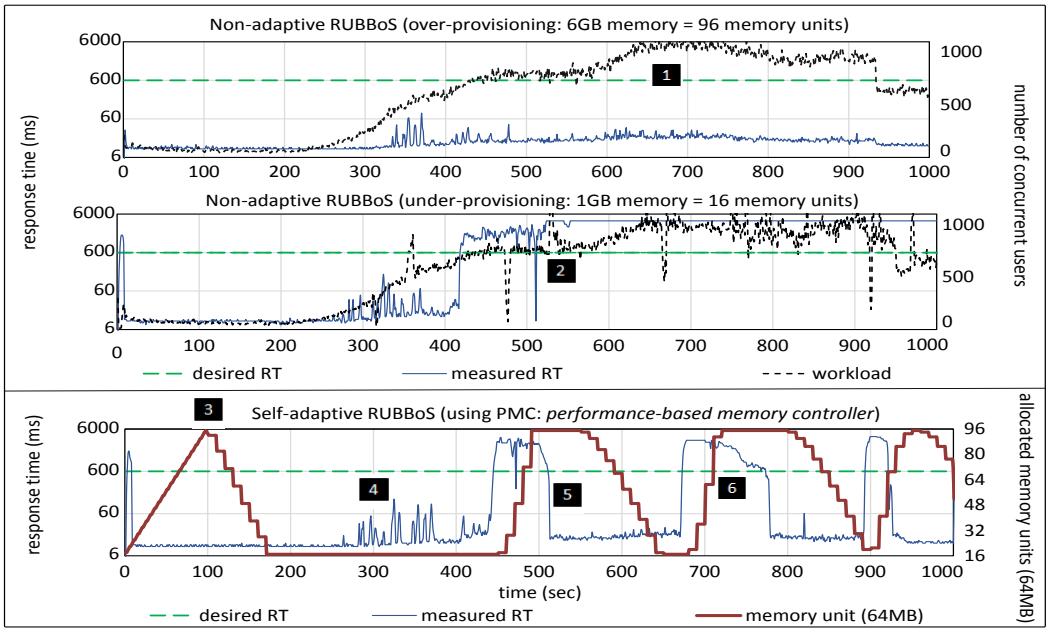


Figure 7.9: Time-series analysis results of non-adaptive and self-adaptive RUBBoS under Wikipedia workload

value, control interval, and the weighted moving average period ($\text{control}_{\text{WMA}}$).

7.3.2 Experimental Results: Time-Series Analysis

In this section, for each workload trace, we explain time-series analysis of the results, depicted in Figures 7.9 and 7.10. We discuss how *performance-based memory controller* adjusts the allocated memory of the VM hosting the application business logic tier at each control interval to maintain the desired response time.

Wikipedia workload traces

Diagrams of Figure 7.9 show the behavior of RUBBoS under Wikipedia workload traces. The first two diagrams are related to non-adaptive scenarios. The aim of these two scenarios is to show the application response time under the runtime workload change with over-provisioning and under-provisioning of memory. The first diagram shows that over-provisioning the memory, i.e., allocating 6GB (96 units of memory), is enough to handle the workload peak, denoted by marker 1, and the measured RT stays lower than the desired RT throughout the entire experiment. On the other hand, the result shown in the second diagram as a representation of the under-provisioning scenario, allocating 1GB (16 units of memory) reveals that the application is unable to respond quickly and handle the peak periods, so all requests are timed-out, denoted by marker 2. More precisely, under sever lack of memory and a large amount of user requests, the Apache server enters a state where it is unable to respond to any single request, i.e., seems

freezing. If this happens, the reaction of the VM hosting BL tier to any change in the allocated memory is much slower than normal. Avoiding this situation is a challenge for the memory controller.

The third diagram depicts the result of a self-adaptive RUBBoS in which the allocated memory is adjusted according to the workload change at runtime in a way to keep the measured RT under the desired RT. Note that the workload is the same as two previous diagrams, but for the sake of readability we does not show it in the last diagram. The first 100sec is the period that the controller is learning the behavior of the application, denoted by marker 3. During this period, the controller is exploring the effect of memory increment on the measured RT to capture the system model. Afterward, the controller has the system model and is able to calculate the suitable amount of allocated memory at each control interval. Marker 4 shows the period when the workload is still low; therefore, the controller assigns the minimum amount of memory to VM_1 while keeping the measured RT less than the desired RT ($\tilde{rt} = 600ms$). At the time 450sec, the workload is increasing, so the controller increases the allocated memory, denoted by marker 5, until the measured RT becomes less or equal to the desired RT. Marker 6 shows the situation in which the system is under the workload peak, and the maximum amount of memory has been allocated. Even though all of memory is allocated, for a short period (from time 700sec till 780sec), the measured RT is higher than the desired RT.

The behavior of the controller at the time 800sec, shows that it is fully capable of reacting to decrease the allocated memory which is not necessary any more while keeping the measured RT within range of the desired RT. Notice that in this scenario, as the workload change is slow and almost persistent, we set the controller in a way to react more slowly in a more stable manner. Hence, we configure the controller with the following parameters, shown in Table 7.10: $pole = 0.99$; the control interval of 10sec; the moving average period of 10sec. The values of these configuration parameters are extracted empirically after carefully examining the controller's behavior with a range of values.

FIFA workload traces

Similarly, Figure 7.10 depicts the results regarding the FIFA workload traces in three scenarios. The first diagram shows that the over-provisioning policy (2GB memory) is quite enough for VM_1 to process all user requests in less than the desired RT ($\tilde{rt} = 300ms$). The second diagram reveals that the application cannot handle the frequent and sudden peaks in workload, denoted by marker 7, while following under-provisioning policy (512MB memory); therefore, all measured RTs are timed-out, i.e., $rt_i > 4sec$ after this time. The last diagram shows the result of a self-adaptive RUBBoS equipped with PMC configured by values mentioned in Table 7.10 such as $pole = 0.999$, control interval and moving average period both are 5sec. These values are chosen empirically after examining the results with several values for the situation in which the workload fluctuations are high and temporal in comparison with the previous scenario. With a shorter control interval the controller is more sensitive to react to sudden changes, and with a higher *pole* value, it reacts less aggressively. As depicted, after capturing the system model of the controller,

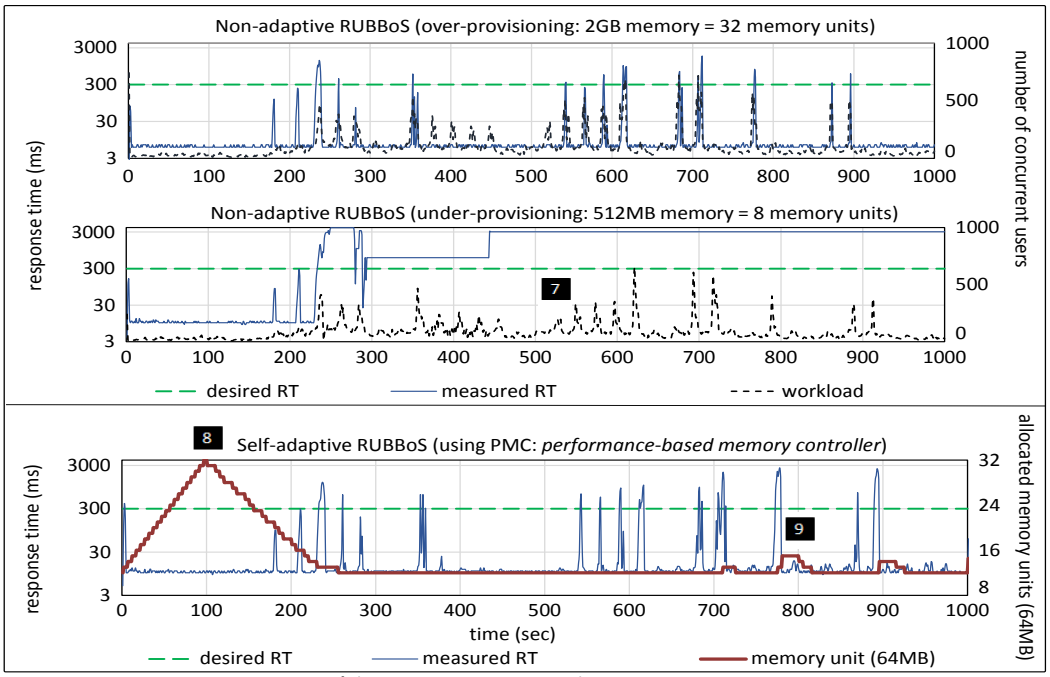


Figure 7.10: Time-series analysis results of non-adaptive and self-adaptive RUBBoS under FIFA workload

denoted by marker 8, as far as the measured RT is less than the desired RT or even more than it, but for a temporary short period (e.g., at 360sec), the minimum memory is allocated to VM_1 by the controller. At 800sec, denoted by marker 9, the measured RT is more than the desired RT for a relatively longer period than other violation periods, so the controller reacts quickly and increases the allocated memory to some extent to handle the workload peaks. Lighter scenarios are repeated at 710sec and 900sec, and the controller reacts very well, according to the measured RT at each period.

7.3.3 Experimental Results: Aggregate Analysis

In this section, we aim to show the effectiveness of PMC on satisfying the desired RT and minimizing the resource cost via a set of experiments. As factors for measuring the effectiveness of the controller, we consider: (i) the average of all measured RTs, which represents our performance metric; (ii) the average number of memory units used over time, which in our case, determines the variable part of the ownership cost. These criteria cover the main aspects of elasticity comprising: scalability; cost; and time efficiency. The goal is to meet the desired RT, which is set as $\tilde{rt} = 600ms$ in the experiment associated with the Wikipedia and $\tilde{rt} = 300ms$ for the FIFA, while keeping the memory usage as low as possible. The drop in memory usage represents the potential capacity to be released back to the cloud to save on cost. To this aim, we compare the experimental results of a self-adaptive RUBBoS equipped with the controller, with two introduced non-adaptive scenarios, under the workload traces. A visual summary of the aggregate

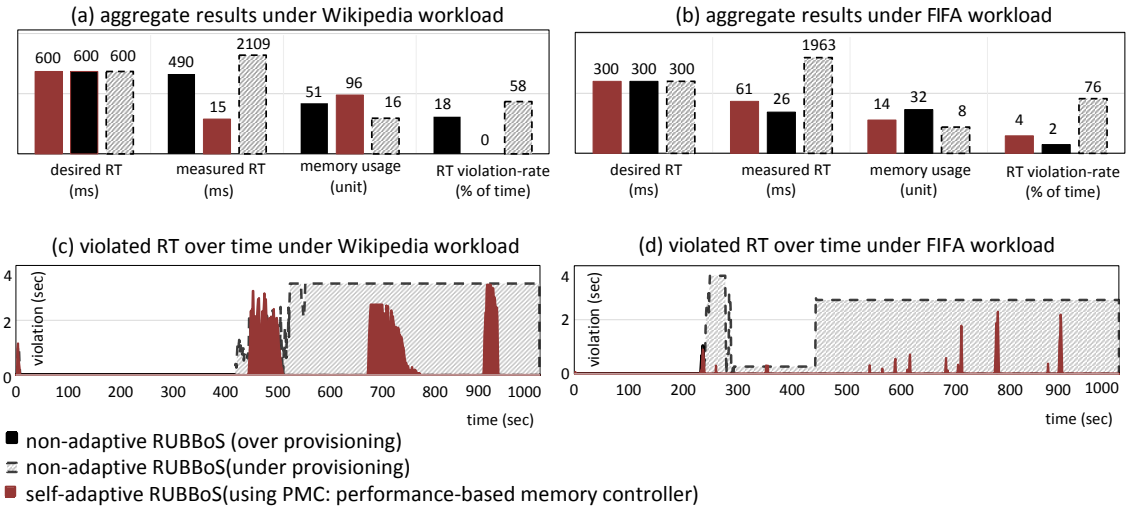


Figure 7.11: Aggregate results of non-adaptive and self-adaptive RUBBoS under Wikipedia and FIFA workloads

results is depicted in the diagrams of Figure 7.11 and is discussed in the following sections, categorized by the workload traces.

Wikipedia workload traces

As shown in Figure 7.11 (a), in comparison with the over-provisioning policy, the controller acquires less memory as the representative of cost, so decreasing the cost by 47% (51 memory unit vs. 96 memory unit). In comparison with the under-provisioning policy, the controller is significantly better in terms of the achieved RT (490ms vs. 2109ms), giving the cloud application' owner a better chance to guarantee the performance metric. To be more precise, beside the total average of measured RT, we also show the percentile of time in which the violation is occurring during the experiment, shown as RT violation-rate in Figure 7.11 (a). Figure 7.11 (c) depicts the distance between the desired RT and the measured RT when the violation is occurring throughout the whole experiment. In other words, this diagram projects the area of the last diagram of Figure 7.9 in which the measured RT is above the desired RT. In an ideal situation, a controller should be able to minimize this area and keep the violation rate close to zero. As shown, although the controller is able to keep the total average of the measured RT less than the desired RT (490ms vs. 600ms), at 18% of the time, the measured RT is larger than 600ms.

FIFA workload traces

For the results shown in Figure 7.10 (b), the situation is much better. In comparison with the over-provisioning policy, the controller has allocated less memory, so consequentially decreasing the memory usage by 57% (14 memory unit vs. 32 memory unit). In comparison with the under-provisioning policy, it behaves better in terms of the average of measured RT (61ms vs. 1963ms). As for the violation rate during the experiment, Fig-

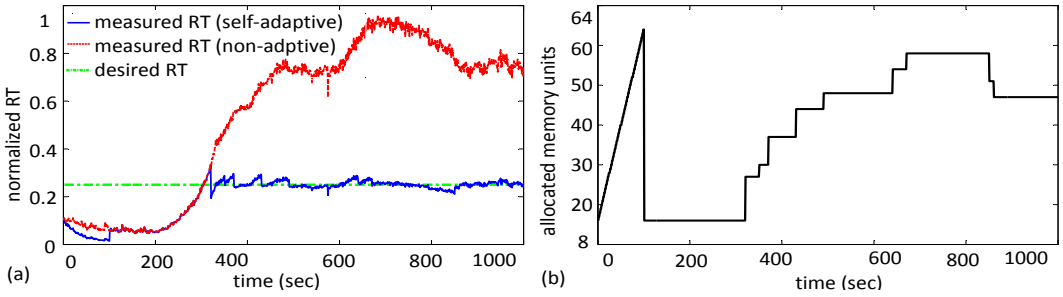


Figure 7.12: Simulation results of the controller's behavior under Wikipedia workload

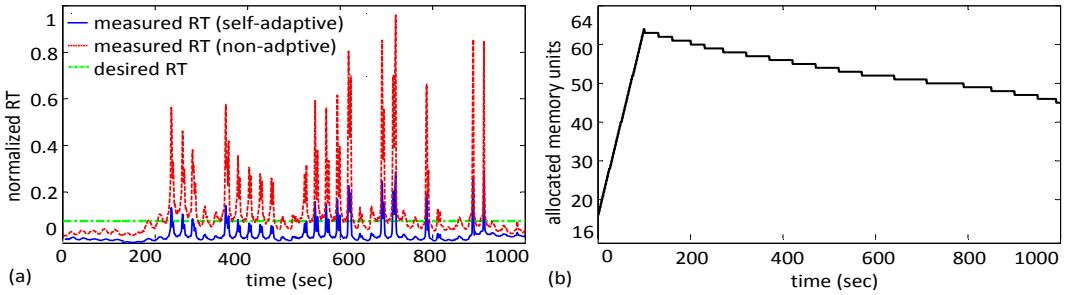


Figure 7.13: Simulation results of the controller's behavior under FIFA workload

ure 7.11 (d) depicts the comparison of the percentile of time in which there is a violation among self-adaptive and non-adaptive scenarios. In this workload, the violation rate is much less (4%), so in total, the benefit of the controller can be more significant for this type of workload that has temporal peaks.

7.3.4 Simulation Results

Although our evaluation is based on the discussed experiments, in order to fine-tune *performance-based memory controller*, figure out its behavior, and make it ready for the real experiments, we first run two sets of simulations under the same workload traces. The controller used in the simulation is the same as in the one used in the real experiment. We define a method to estimate the measured RT as a function of the allocated memory and the number of user requests at each interval during 1000sec simulation run. Figures 7.12 and 7.13 show the corresponding results of a non-adaptive versus a self-adaptive simulated application under the used workload traces. Note that in non-adaptive scenarios, the allocated memory is fixed, so the workload is the only variable factor which influences the estimated measured RT; therefore, the diagrams of the measured RT (Figures 7.12 and 7.13 (a)) look similar to the used workload patterns. Figures 7.12 and 7.13 (a) show the estimated measured RT of the simulated application at each iteration, while Figures 7.12 and 7.13 (b) represent the memory units allocated by the controller at each iteration; see Table 7.10 (simulation part) for the configuration parameters. Based on the simulation results, we can see, apart from the instabilities that happen in a real environment, the stability, responsiveness and

robustness of the controller is sufficient to cope with the used workload patterns. Similarly, the first 100sec is the period that the controller is capturing the system model, then the controller allocates the proper amount of memory to cope with the varying number of users during the simulation run. The root mean square error (RMSE), Equation (7.1), is a frequently used measure as the evaluation factor in the control theory domain which also is reported in the following simulation results. It represents the sample standard deviation of the differences between measured RT and desired RT.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\tilde{rt} - rt_i)^2} \tag{7.1}$$

The RMSE of *performance-based memory controller* for the presented simulation results are 0.101 (Wikipedia) and 0.057 (FIFA), which are quite good in the control context.

7.3.5 Discussions

The aggregate results, recalling diagrams of Figure 7.11, show the benefit of the controller to save the memory usage (cost) by 47% in the case of Wikipedia, and 57% in the case of FIFA workload traces in comparison with the over-provisioning policy. In both cases, the controller could satisfy the desired RT with the total average of measured RT (490ms vs. 600ms, and 61ms vs. 300ms). However, as we use the real-world workload instead of using simple step workload, which is quite common to benchmark a controller, at 18% of the time the measured RT is more than the desired RT in the case of Wikipedia, and 4% in the case of FIFA workload traces. The trade-off between the importance of saving more at memory usage or having less violation is subjective and up to the owner of the cloud application. Moreover, the behavior of the controller under two sets of experiments as well as the simulation run, reveals that *performance-based memory controller* behaves as intended. After identifying the system model during the learning phase, at the control phase the proper amount of memory for the VM hosting BL tier is adjusted in order to keep the measured RT close to the desired RT. Regarding the experimental results, the two used error smoothing techniques in our controller, *support vector regression* and *weighted moving average*, previously explained, helped the controller to be able to cope with the workload fluctuations at runtime.

The evaluated *performance-based memory controller* utilizes a static system model that does not support any explicit update mechanism during the control phase. Therefore, if the application enters a new phase with a different load pattern, it is required to continually update the system model. Moreover, while increasing or decreasing the memory, *performance-based memory controller* does not have an insight to the current value of resource utilization, there it may lead to over-committing memory under some runtime conditions. These limitations are addressed in the scope of this thesis by the proposed hybrid memory controller, which is evaluated in the next section.

7.4 Hybrid Memory Elasticity Controller

In this section, we present the experimental evaluation of the implemented HMC, *hybrid memory controller*, presented in Chapter 5 as contribution IV, and compare it against two baseline controllers. In the following, we first describe the baseline approaches, then the experimental setup, and finally discuss the evaluation results.

7.4.1 Baseline Approaches

In this section, we briefly introduce the two baseline controllers, performance-based and capacity-based memory controllers for evaluating the proposed HMC.

Performance-based memory controller (PMC)

The memory controller used as the baseline is *performance-based memory controller* previously presented in Chapter 5 (see Section 5.4 for more details), and evaluated in Section 7.3. As a brief recalling, it follows a control synthesis technique. At each control interval, PMC's output $ctl \in [0, 1]$ is mapped to a memory size $mem_i \in [mem_{min}, mem_{max}]$. The controller takes the application response time as the decision making criterion to adjust the amount of allocated memory.

Capacity-based memory controller (CMC)

As a capacity-based approach, CMC [134] uses memory utilization as the decision criterion to adjust the allocated resource. CMC consists of one elasticity rule that is applied based on memory overprovisioning percentage (MOP) $\in [0, 1]$. The idea behind MOP is to avoid thrashing so that to keep the VM memory size beyond the memory used by the application. The used elasticity rule enables CMC to decide when to scale up or scale down by monitoring the value of memory usage ($mem_{used} \in [0, 1]$), and the corresponding calculated MOP at each control interval (every 5 sec in our experiment). As shown in Equation (7.2), mem_{used} is estimated by using the values reported at `meminfo`[10] file, which includes the information about the Linux system's memory at runtime. Similar to HMC and PMC, we define a minimum amount of memory where the controllers cannot shrink the VM memory size below this amount. This allows the guest operating system to properly operate and avoid experiencing unexpected application crashes due to the lack of memory.

$$mem_{used} = mem_{total} - (mem_{free} + mem_{cached} + mem_{buffers}) \tag{7.2}$$

The elasticity rule is applied if the free virtual machine memory ($mem_{free} = 1 - mem_{used}$), is smaller than 80% or greater than 120% of the MOP. Under such conditions, CMC dynamically adjusts the memory size of the virtual machine using Equation (7.3).

$$mem_{size} = mem_{used} \cdot (1 + \text{MOP}) \tag{7.3}$$

[10]Linux memory information file: `/proc/meminfo`

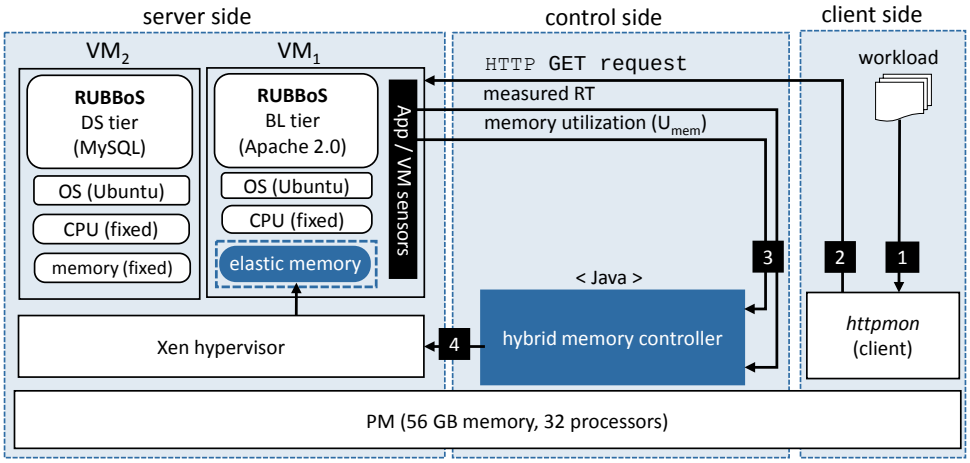


Figure 7.14: Overview of the experimental setup for evaluating HMC

This rule implies decreasing or increasing the memory size depends on the behavior of the application deployed in the VM, and the magnitude of the memory changes depends on how fast or slow the application requests or releases the memory. A lower value of MOP aims at reducing the unused memory of the VM, i.e., achieving higher utilization, but has a higher chance to incur thrashing (rapidly exchanging data in memory for data on disk, to the exclusion of most application-level processing), if the application memory consumption grows faster than the rate at which CMC increases the memory size. In contrast, a higher MOP aims at reducing the chance of thrashing if the memory consumption grows rapidly, but surely at the expense of wasting more memory. Based on our experimental results and the type of the application used in our evaluation, we set 0.1 as a value for MOP, as a suggested value in the original work [134].

7.4.2 Experimental Setup

As shown in Figure 7.14, the experiments are conducted on a physical machine equipped with 32 cores[11] and 56 GB of memory. To emulate a typical virtualized environment and apply the vertical memory elasticity, we use Xen hypervisor. The benchmark application is deployed in two separate VMs, where VM_1 runs a Web server, Apache 2.0 with PHP enabled, and VM_2 runs MySQL as the application database. VM_2 is provisioned by sufficient memory and CPU cores to avoid becoming a bottleneck during the experiments. The Apache configuration is exactly as what was described previously in Section 7.3.1, so it is not repeated here.

Benchmark application. Similar to the evaluation of PMC, we again use RUBBoS as the benchmark application, introduced previously in Section 7.3.1.

Workloads. Experiments are performed using different workloads to characterize the controller's responses to performance changes. We evaluate HMC by using workload

[11]Two AMD Opteron™6272 processors, 2100 MHz, 16 cores each, no hyper-threading.

generated based on the open and closed system models [159]. A closed system model is defined when the arrival of new requests is only triggered by previous request completions, followed by *thinktime*. While, an open system model is defined when new requests arrive independently of the previous request completions. For open clients, we change the arrival rate and inter-arrival time during the course of the experiments as required to stress the system. For the closed model, *thinktime* of each client as well as the number of concurrent users are varied. The change in arrival rate or number of users is made instantly. This makes it possible to meaningfully compare the system behavior under these two client models.

To emulate the users accessing the applications, under the generated workload, we use the previously introduced workload generator tool, *httpmon*, which is able to generate both open and closed system model client behaviors. By using *httpmon*, a large value of *thinktime* with a high number of concurrent users would induce a high memory utilization while keeping the CPU utilization low. We also keep constant the number of requests for some time to study the behavior of the models under both the steady and transient states. In addition, to validate the applicability of our approach against real-life situations, we also use real-world workload traces extracted from Wikipedia and FIFA WorldCup website. These traces are selected due to their complementary nature. While the Wikipedia workload shows a steady and predictable trend, the used FIFA workload has a bursty and an unpredictable pattern. Figure 7.15 depicts the patterns of these traces. Then, these two patterns map to concurrent users who send HTTP GET requests.

Performance metric. The response time of a request is defined as the time elapsed from sending the first byte of the request to receiving the last byte of the reply. In this work, we are mostly interested in the *mean response time* over 20 seconds (4 control intervals), which is a long enough period to filter measurement noise, but short enough to highlight the transient behavior of an application. As pointed in [8], good elasticity metrics are the ones which are related to a single application tier. Such metrics can be tracked, their patterns can be learned, their statistics can be calculated, and then an intelligent elasticity controller can use them to ensure the user satisfactions in terms of QoS. This is one of the reasons that in our work we only focus on the BL tier of the application; therefore, based on the presented experimental setup, the defined evaluation metric is only dependent on the memory resource of the VM hosting BL tier of the application.

Experiment process. Figure 7.14 depicts the experimental setup for evaluating HMC. Numbers in this figure indicate the experiment's process. The experiment starts when the workload traces are fed into *httpmon* (1), and based on the values of workload traces at each control interval, *httpmon* emulates a specific number of concurrent users to send HTTP GET requests to the application under test (2). At each control interval, 5 seconds in our experiments, the application sensor observes the mean response time and the VM sensor measures the average of memory utilization in the course of this period. Both sensors send their monitored information via TCP/IP connection to the controller (3). *Hybrid memory controller* dynamically determines the required memory

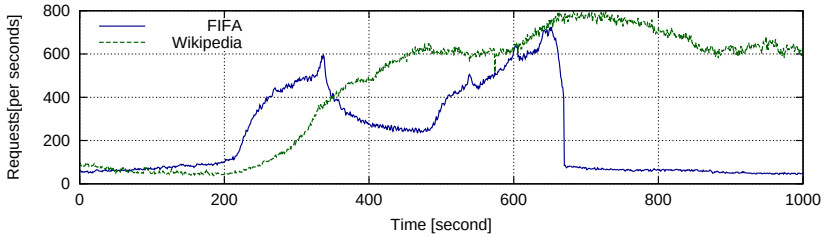


Figure 7.15: Workload patterns from the real-world traces: unpredictable–FIFA-based and predictable–Wikipedia-based

size of VM_1 in order to meet the desired RT. Finally, the controller invokes the memory actuator, i.e., the Xen API for memory allocation, to either increase or decrease memory size of VM_1 at runtime (4).

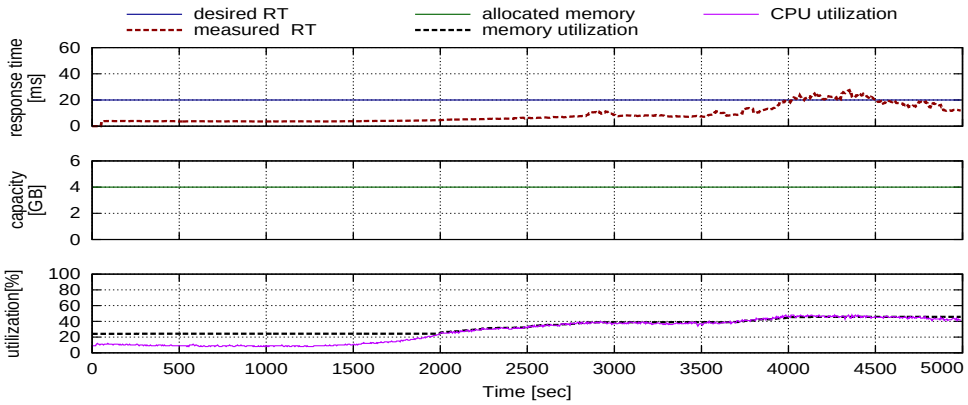
7.4.3 Experimental Results

In this section, the evaluation results are presented and discussed as follows:

- **Non-adaptive scenarios**. Analyzing the time-series results of a non-adaptive RUBBoS, i.e., without any controllers, under Wikipedia workload. Non-adaptive scenarios consist of both memory over- and under-provisioning.
- **HMC time-series analysis**. Analyzing the time-series results of HMC under various workload traces, including open, and closed system models, Wikipedia, and FIFA WorldCup website traces. The goal here is to show that the proposed controller is able to meet the desired RT without over-provisioning while achieving relatively high memory utilization.
- **Time-series comparison**. Presenting the results related to the behavior of CMC and PMC under Wikipedia workload traces and comparing them with the results achieved by the proposed controller, HMC, in a similar experimental setup.
- **Aggregate analysis**. Reporting the aggregate results related to a self-adaptive RUBBoS equipped with the three controllers under different scenarios as well as a non-adaptive RUBBoS covering both memory under- and over-provisioned scenarios.

All diagrams presented in this section are structured as follows. Each figure consists three diagrams which show the results of a single experiment. Note that we perform a number of experiments and find similar patterns in the results and then presents one of them. The bottom x-axis represents the time elapsed since the start of the experiment. In each figure, the upper diagram plots *mean response times*, the bottom diagram plots the memory and CPU utilization of the VM hosting the BL tier (i.e., VM_1) of the application under test. The rationale behind reporting the CPU utilization is to show that in all experiments, the CPU has not been the bottleneck resource. Finally, the middle diagram plots the amount of memory required in GB computed by the respective controller and allocated to VM_1 over the next 5 seconds as the default control interval.

(a) Over-provisioning the memory



(b) Under-provisioning the memory

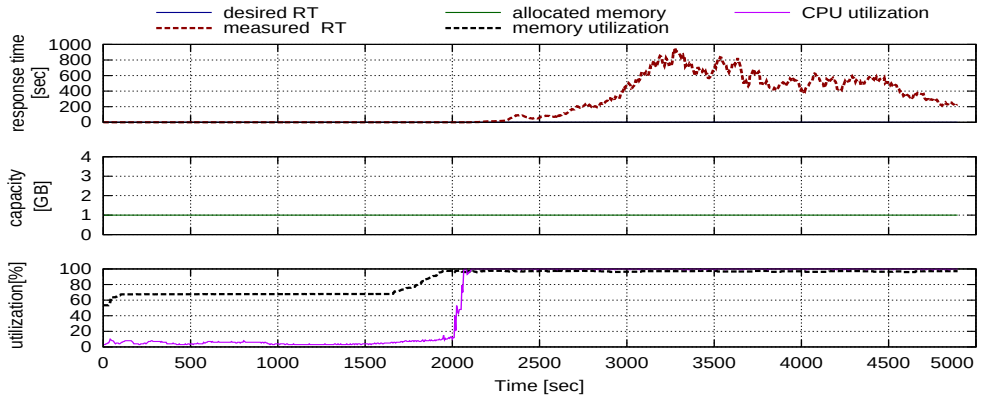
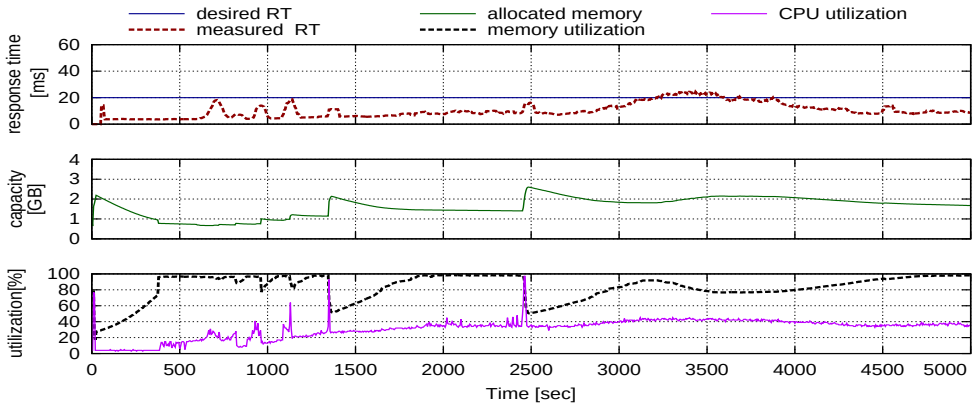


Figure 7.16: Non-adaptive RUBBoS, under Wikipedia workload, 20ms desired RT

Non-adaptive scenarios

Figures 7.16 (a) and (b) present the behavior of a non-adaptive RUBBoS, under over-provisioning and under-provisioning of allocated memory, respectively. The aim of these two scenarios is to show the application measured RT when the allocated memory is static and no controller is involved. It can be observed that in the over-provisioning case, Figure 7.16 (a), the desired RT ($\tilde{rt} = 20ms$) is easily met with allocating 4 GB of memory, but with the expenses of wasting the memory during the experiments and consequently causing a high resource cost, while achieving a low resource utilization. On the other hand, in the case of under-provisioning experiment, Figure 7.16 (b), the measured RT roughly follows the workload pattern and it is much higher than the desired RT from when the workload started to increase (time 2000 sec), while achieving a very high resource utilization. However, the application is unable to respond quickly and handle the peak periods, so all requests after this time have higher measured RTs values than the desired RT. In general, these results reveal the need for self-adaptive solutions, i.e., using memory controllers that dynamically adjusts the allocated memory.

(a) Wikipedia workload, 20ms desired RT, 0.99 pole



(b) FIFA workload, 15ms desired RT, 0.9 pole

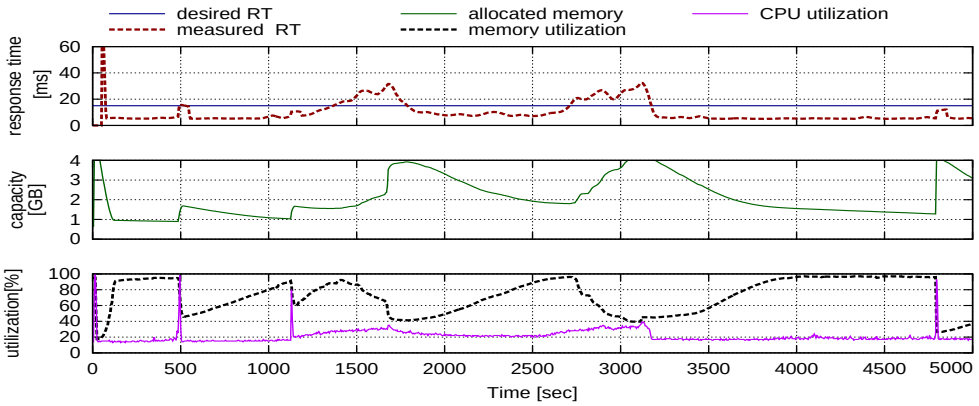


Figure 7.17: Self-adaptive RUBBoS equipped with HMC, under real-world workload traces, 5sec control interval

HMC time-series analysis

To show the behavior of HMC under various conditions, Figures 7.17 and 7.18 show the results for scenarios in which RUBBoS is equipped with HMC under different workloads: open and closed system models ($\tilde{rt} = 30ms$), as well as the Wikipedia ($\tilde{rt} = 20ms$) and FIFA ($\tilde{rt} = 15ms$) traces. In general, the measured RTs remain lower than or relatively close to the desired values under both system models (see diagrams of Figure 7.18) and the measured RTs converge to the desired RT immediately, mostly without being noticed, after detecting a sudden increase or decrease in the workload, e.g., from the first interval (200 requests/ sec) to the second interval (500 requests/ sec). This can be also seen in scenarios with real-world workloads (see diagrams of Figure 7.17, taking Figure 7.15 as the workload patterns). The used *pole* value for all scenarios is 0.9, except the Wikipedia. In the case of Wikipedia workload due to its slow incremental nature, it is not required to have a quick reaction, so the *pole* value is set to 0.99. Empirically, we observe having this value leads to a lower control error for this workload type.

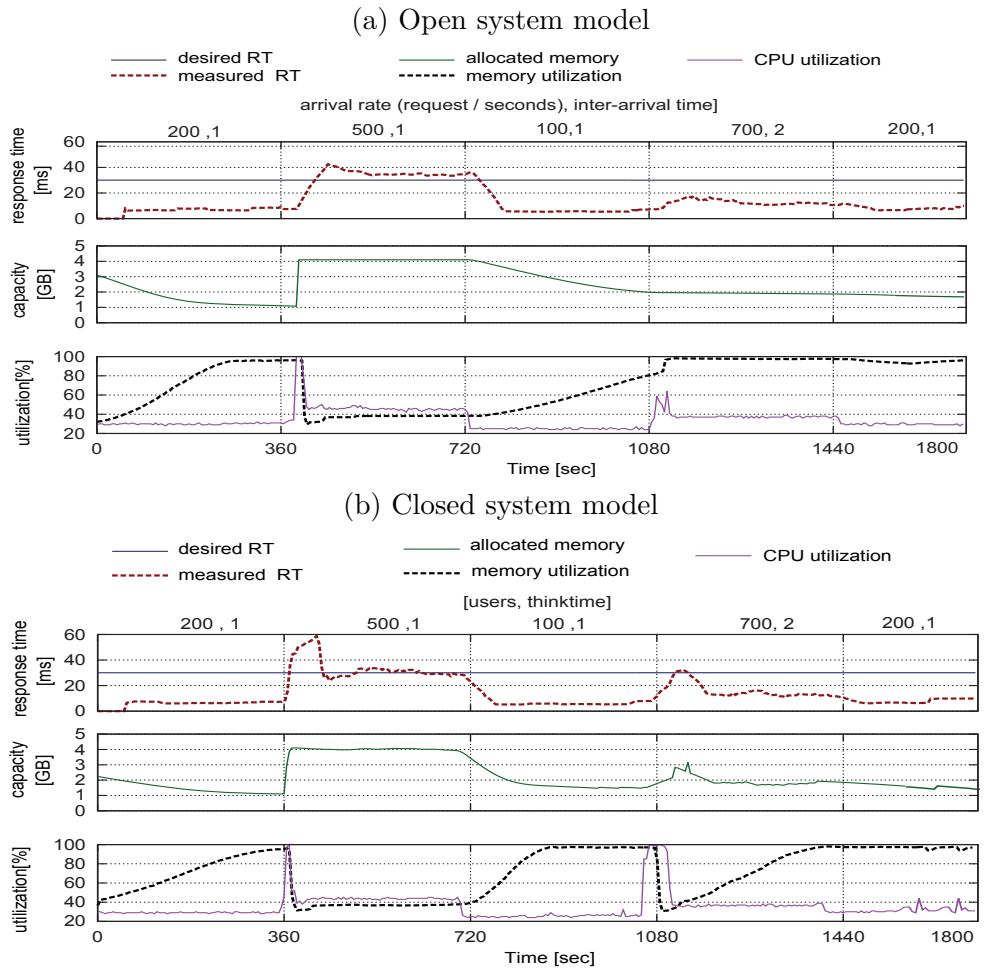
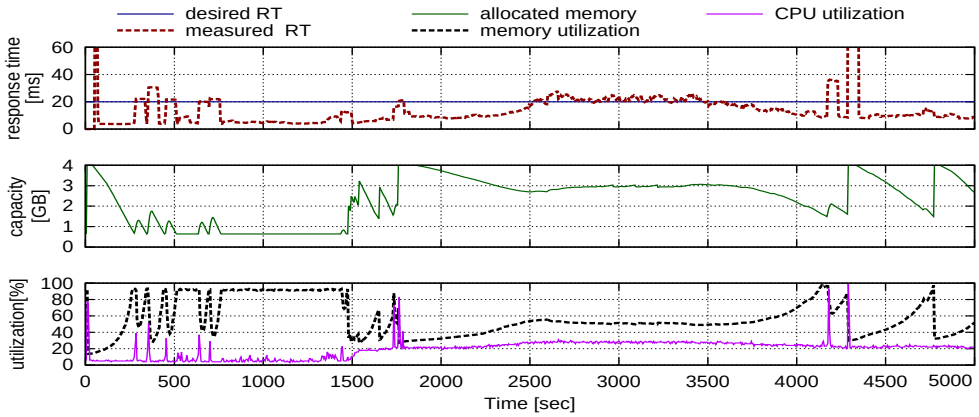


Figure 7.18: Self-adaptive RUBBoS equipped with HMC, under synthetic workload traces, 30ms desired RT, 0.9 pole, 5sec control interval

The other important point to note is that HMC properly detects and adapts to the memory capacity required to meet the desired RTs for both open and closed system models. Indeed, as it can be observed from the plots presented in Figure 7.18, a close observation of the results reveals that the allocated memory is slightly higher under the open system model compared to the closed system model for the similar configurations (i.e., under the same workload that is labeled on top of each interval for the first diagram in Figure 7.18 (a) and (b)). This is because the number of created Apache processes is slightly higher under the open system model than the closed system model. Moreover, there is a slight increase in memory usage as the number of users or arrival rates increase because of the equivalent number Apache processes created. However, memory is not immediately released unlike CPU cores as the number of users or arrival rates decrease, since the idle Apache processes are not garbage collected, immediately.

(a) PMC: 20ms desired RT, 0.99 pole



(b) CMC: MOP 0.1

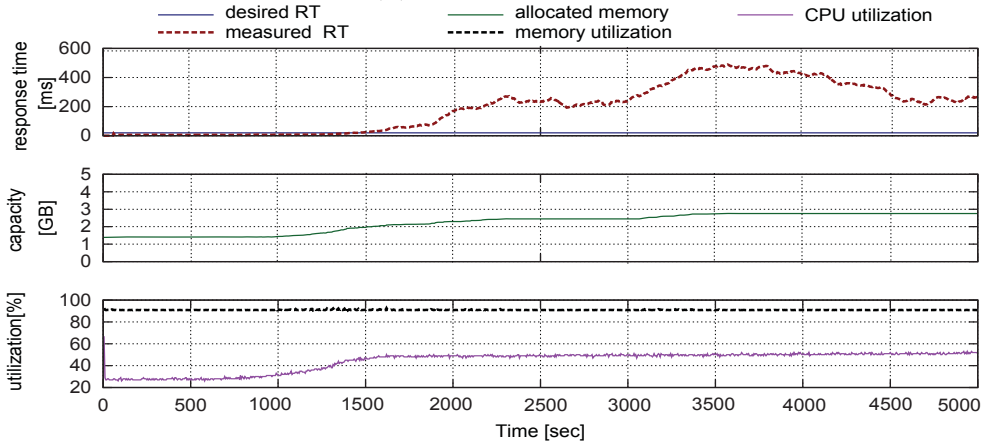


Figure 7.19: Self-adaptive RUBBoS equipped with PMC and CMC, under Wikipedia workload, 5sec control interval

Time–series comparison of CMC, PMC, and HMC

To compare the behavior of the proposed *hybrid memory controller* with the two baseline controllers, Figures 7.19 (a), (b), and Figure 7.17 (a), present the best results achieved by using these three controllers with the same benchmark application under Wikipedia workload traces. As shown in Figure 7.19 (a), while CMC, as a capacity-based controller, is able to highly utilize the allocated memory, this amount of memory is obviously inadequate to ensure the application performance. Based on the results of Figure 7.19 (b), the values of the measured RT exactly follow the application workload pattern (see Figure 7.15 for the Wikipedia workload pattern). This is because the capacity-based approaches are oblivious to the observed performance of services and only try to adjust the resources in a way to achieve high resource utilization. Therefore, such a controller may lead to SLA violations, and it is not appropriate to be used for applications with sensitive performance requirements such as interactive applications.

The result of PMC, Figure 7.19 (a), reveals that taking the application response time as an indicator of the memory scarcity is enough to meet the application performance requirements. However, a performance-based approach such as PMC sometimes decides inefficiently as it does not have any insight into the resource utilization at runtime. This can cause either over- or under-provisioning the resource. A common problem of such approaches is when the application performance is close to the saturation point, but still the measured RT is far better that the desired RT, therefore, a performance-based approach such as PMC decides to decrease the allocated memory to avoid the memory wastage. While the controller in this situation is oblivious about the memory utilization, depends on the intensity of the workload at that moment, any decrements of memory can suddenly enter the application into a memory saturation circumstance. Consequently, it can cause a sudden peak at the measured RT and sometimes cause the SLA violation. Indeed, the explained scenario is observed in Figure 7.19 (a) at the time 4200sec while the application measured response times face a sudden peak.

Nevertheless, results of Figure 7.17 (a) show that HMC remains stable in terms of both maintaining the application performance objectives and avoiding resource over- and under-provisioning. In compared to CMC, based on the utilization diagrams (the last diagram of Figure 7.19 (b) and Figure 7.17 (a)), while it is clear that the memory utilization achieved by using HMC is relatively lower than by the one by using CMC but only in the case of HMC the measured RT is maintained lower than the desired RT. In comparison with PMC (Figure 7.19 (a)), HMC is able to meet the desired RT with less oscillations, low values for the standard deviation of response time as reported in Table 7.11, and achieving a higher memory utilization. These comparisons are more obvious when analyzing the aggregate results at the following section.

Aggregate analysis

To assess the aggregate behaviors of HMC in comparison with PMC, and CMC over the course of the experiments, in Table 7.12 we report the mean values of the allocated memory, memory utilization, and CPU utilization, as well as the values for the mean and standard deviation of response time for different scenarios. Moreover, we also present two control theoretic metrics, integral of squared error (ISE) and integral squared of timed error (ISTE), which represent the observed error during the life span of the system under test. These metrics are computed as shown in Equations (7.4) and (7.5) [125], where $e(t) = \tilde{rt} - rt(t)$. ISE metric reports how much the measured RT is close to the desired RT. ISTE is a timed variant of the ISE, which weights the error over time and reduces the effect of the initial transient phase. ISTE would result in better values for a control theoretical solution, where the transient phase is the price to pay for modeling and controlling the system [125]. Note that although achieving a lower measured RT seems preferable from the end users' point of views, an ideal elasticity controller should be able to achieve a measured RT that is close to the desired RT, not far better since this can implicitly indicate the resources over-provisioning.

$$ISE = \frac{1}{n}\sum (e(t))^2 \tag{7.4}$$

scenario	workload ($\tilde{rt}$)	controller	ISE	ISTE	mem ($mean$) [GB]	RT ($mean$) [sec]	RT (SD) [sec]	U_{cpu} [%]	U_{mem} [%]
non-adap.	Wiki (20ms)	over-pro	0.0286	0.019	4	0.0087	0.0007	27.97	33.67
		under-pro	0.0944	0.0856	1	0.234	0.3053	60.45	85.97
HMC	open (30ms)	HMC	0.0816	0.056	2.42	0.015	0.014	34.63	69.62
	closed (30ms)		0.0825	0.055	2.15	0.013	0.011	34.91	68.85
	FIFA (15ms)		0.0396	0.0245	2.098	0.014	0.0713	21.04	70.91
comparison	Wiki (20ms)	HMC	0.0364	0.0235	1.595	0.0105	0.0069	31.05	83.36
		PMC	0.0381	0.0253	2.3219	0.015	0.0647	19.47	59.74
		CMC	0.0305	0.0223	2.0569	0.176	0.1732	42.02	90.97

Table 7.11: Self-adaptive and non-adaptive aggregate results

$$ISTE = \frac{1}{n}\frac{\sum \left(t \cdot e\left(t\right)\right)^2}{\sum t} \tag{7.5}$$

Table 7.11 shows these aggregate results of non-adaptive RUBBoS application without using any controller, as well as self-adaptive RUBBoS equipped with the three different controllers under various workload scenarios. As shown, the first set of results is related to non-adaptive experiments where no controller is used. The goal of such experiments is to show that with a static memory allocation policy, i.e., over- or under-provisioning the memory, either the application performance should be sacrificed (see under-provisioning results), or the desired performance can be met with the expense of memory wastage to be able to handle the workload peak (see the over-provisioning results). The rationale behind having a relatively high CPU utilization in the case of under-provisioning in compared to all other scenarios is that, when the memory is not sufficient for the application to handle the workload, the VM starts using thrashing which is a CPU intensive task.

The second set of scenarios (the second row of Table 7.11) is related to HMC under open and closed system models as well as the FIFA workload. The aggregate results using HMC reveal that *hybrid memory controller* is able to keep the measured RT lower than the desired RT, while it uses a reasonable memory with relatively lower error values, and high memory utilization (close to 70% in all three experiments). Note that achieving low CPU utilization in all scenarios indicates that the CPU is not a resource bottleneck in any of the scenarios (i.e., based on the experimental setup CPU has been over-provisioned).

The last set of experiments (the last row of Table 7.11), compare the aggregate results of HMC to the two baseline controllers, PMC, and CMC, under the Wikipedia workload with similar test conditions. A controller is said to be better if the desired RT is met with lower error and without memory over-provisioning. Note that, except the other metrics in this figure, in the case of memory utilization achieving a higher value is preferable. Based on the results of these three experiments, while HMC uses the lease amount of memory (1.59 GB), it can meet the application desired performance with low control errors and high memory utilization (83%). Although among all, CMC achieves the highest value of memory utilization (90.97%), the measured RT is very high in this case (220ms) compared to the desired RT (20ms), and it even uses more memory (2.06 GB)

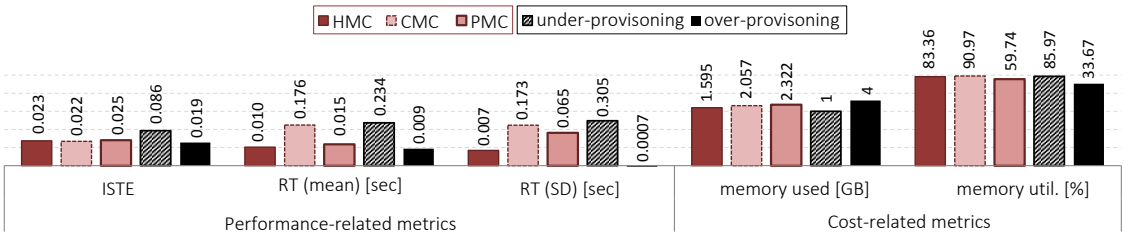


Figure 7.20: Aggregate results of self-adaptive and non-adaptive RUBBoS, under Wikipedia workload

than the proposed HMC.

Figure 7.20 visualizes the aggregate results, reported in Table 7.11, and explained above. Specifically, this diagram depicts the aggregate results obtained under Wikipedia workload for five different experiments, including self-adaptive scenarios, (using HMC, CMC, and PMC presented with reddish plots), as well as non-adaptive scenarios (under- and over-provisioning, presented with black/ gray plots) in terms of either performance-related metrics, or cost-related metrics. Figure 7.20 is illustrated in details in the following discussion section.

7.4.4 Discussions

Recalling Figure 7.20, in general, the usage of the proposed *hybrid memory controller* (the dark red plots) leads to better results in both performance and cost aspects. More precisely, the results achieved by HMC compared to the results using one of the baseline controllers reveal that while the observed control error (ISTE) is almost comparable throughout the experiment, the stability of RT is lower for CMC compared to HMC (0.173 SD vs. 0.0069 SD) by even allocating more memory (2.05 GB vs. 1.59 GB), leading to SLA violation. This shows that a capacity-based approach such as CMC in spite of achieving a higher resource utilization (90.97 % vs. 83.36%) compared to our proposed hybrid-approach, is oblivious to the application performance, achieving at least 10 times higher measured RT compared to HMC (0.176sec vs. 0.01sec), and it obviously violates the SLA ($\tilde{rt} = 20ms$).

On the other hand, a performance-based approach such as PMC can achieve almost the same measured RT compared to the proposed hybrid approach with comparable control error (0.023 vs. 0.025), but with higher memory usage (2.32 GB vs. 1.59 GB), lower memory utilization (59.74% vs. 83.36%), and much lower stability in measured RT based on the value of SD (0.065sec vs. 0.006sec). The most stability is achieved by the hybrid approach in comparison with the performance-based approach due to the insight that it has into resource utilization, while PMC only decides based on the application performance, i.e., response time, at runtime. That is why HMC achieves the best stability (i.e., the lowest SD value) in RT among all other baseline approaches.

Results related to the comparison between two non-adaptive scenarios and self-adaptive RUBBoS equipped HMC show the benefit of leveraging elasticity controllers

that dynamically adjusts the allocated resources, rather than allocating static amount of resources. Compared to HMC, under the over-provisioning scenario, the memory utilization is lower (33.67% vs. 83.36%), and much more memory is allocated on average (4 GB vs. 1.59 GB). Meanwhile, it achieves the lowest average RT (9ms), which is far better than the desired RT (20ms), with the expense of wasting resources. On the other hand, in the under-provisioning experiment only 1 GB of memory is allocated statically achieving a high utilization (90.97%), but due to the lack of memory during the workload peak, the mean and standard deviation values of the measured RT are not acceptable (see Figure 7.20) and it causes an SLA violation.

In summary, the trade-off between the importance of saving more at memory usage (i.e., achieving a higher memory utilization) or having less performance violation is subjective and is up to the owner of the cloud application. Moreover, the behavior of the hybrid controller under all experiments reveals that HMC behaves as intended. After identifying the system model and updating it, if it is required, at each control interval, it adjusts the right amount of memory in order to keep the measured RT close to the desired RT without over-committing the memory. It takes into account both the memory utilization and the application response time as decision making indicators.

7.5 Coordinating CPU and Memory Elasticity Controllers using Fuzzy Control

In this section, we present the experimental evaluation of the proposed *fuzzy coordination approach*, presented in Chapter 6 as contribution V. To this aim, we compare the results of our approach using *fuzzy controller* (FC) to an approach that does not have such a coordinator, i.e., *non-fuzzy controller* (NFC). NFC is based on by simply running the previously introduced independent work for *cpu controller* [112] and *memory controller* (see Chapter 6 for more details) in parallel without having any synchronization between the two controllers during the resource provisioning decision. Indeed, the outputs of these two controllers are directly applied on the VM without any influence of *fuzzy controller*'s outputs. In this case, the inputs for both *cpu controller* and *memory controller* are only the desired RT and the measured RT. In the evaluation scenario for NFC, it is assumed that an application is either CPU- or memory-intensive, so there is no need for coordination between the controllers which control these resources separately. The goal of the evaluation is to show which approach, FC or NFC, meet the desired RT by predicting the right amount of resources and avoiding under- or over-provisioning. A controller is said to be better if the desired RT is met without over-provisioning any of the resources. In what it follows, we first describe the experimental setup and then, we report and discuss the evaluation results.

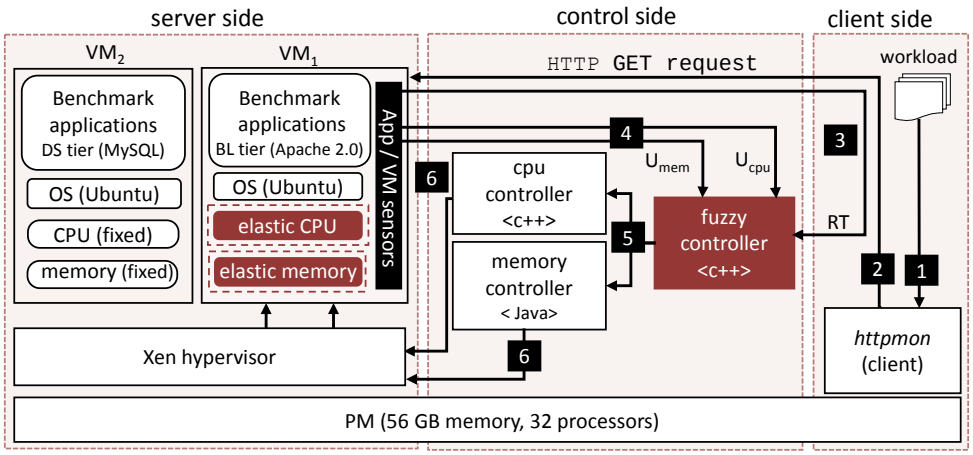


Figure 7.21: Overview of the experimental setup for evaluating the fuzzy coordination approach

7.5.1 Experimental Setup

The experiments are conducted on a physical machine equipped with 32 cores and 56 GB of memory. To emulate a typical cloud environment, and apply the vertical elasticity action, we use Xen hypervisor. Benchmark applications, as shown in Figure 7.21, are deployed in two separate VMs. VM_1 runs the application BL tier, Apache 2.0 with PHP enabled, and VM_2 runs the application DS tier, MySQL. To avoid VM_2 being a bottleneck for any resources, sufficient memory and CPU cores are allocated during the experiments. The Apache configuration is similar to what was previously described in Section 7.3.1.

Benchmark applications. To test the applicability of the proposed approach with a wide range of interactive applications, we perform experiments using three benchmark applications: RUBiS, RUBBoS, and Olio [9]. These applications are widely-used cloud benchmarks (see [78, 165, 176]) and represent an eBay-like e-commerce application, a Slashdot-like bulletin board, and an Amazon-like book store, respectively.

Workloads. Experiments are performed using different workloads to characterize the controller's responses to performance changes. We evaluate the controller using workload generated based on the open and closed system models, previously explained. The generated workloads gave us freedom of evaluating different parameters. For instance, to increase the number of requests by five-folds or ten-folds to understand the behavior of our solution; to induce memory and/or CPU intensive behavior by varying different parameters such as *thinktime*, and the number of concurrent users accessing the systems. To emulate the users accessing the applications, based on the generated workload, *httpmon*, previously introduced, is utilized. By using *httpmon*, a low *thinktime* would induce high CPU utilization. On the contrary, a long *thinktime* with a high number of concurrent users would induce a high memory utilization while keeping the CPU utilization low. We test the controller under more extreme scenarios than can be found in real-world

traces to more stress-test the system. For open clients, the arrival rate and inter-arrival time are changed. For the closed model, *thinktime* of each client as well as the number of concurrent users are varied.

Performance metrics. In our evaluation, we are interested in the *mean response time* over 20 seconds that is equal to 4 control intervals, which is long enough to filter the measurement noise, and at the same time short enough to reveal the transient behavior of the application.

Experiment process. As shown in Figure 7.21, the experiment starts with feeding the workload into *httpmon* (1), and based on the workload at each control interval, *httpmon* emulates a specific number of concurrent users to send HTTP GET requests to the benchmark application (2). At each control interval, 5 seconds in our experiments, the application-level sensor observes the average of measured RT (3), while the VM-level sensor measures the average of CPU and memory utilization (4). These sensors send their monitored information via TCP/IP connection to the controller. *Fuzzy controller* computes the corresponding coefficients for CPU C_{cpu} and memory C_{mem}, and they are fed to the respective controllers (5). Then the CPU and memory controllers compute the amount of the respective resource. Finally, each controller invokes the corresponding actuator, i.e., the Xen API for CPU and memory allocation, to adjust size of memory or CPU of VM_1 at runtime (6, 6').

7.5.2 Experimental Results

To evaluate the controllers, we inject a variable load, so as to test how each controller reacts during sudden workload spikes (i.e., under extreme conditions) under both open and closed system models. Furthermore, the desired RTs used in the experiments are varied from relatively high to small target values in order to assess the controller's behavior under different scenarios. We discuss the results based on time-series analysis and aggregate analysis in the following sections.

Time-series analysis

The plots in this section are structured as follows. Each figure shows the results of a single experiment. Note that we perform a number of experiments and find similar patterns in the results and then presents one of them. The bottom x-axis represents the time elapsed since the start of the experiment. The upper graph in each figure plots *mean response time*, while the lower graphs plot the number of CPU cores and the amount of memory required in GB as are computed by the respective controllers and allocated to the VM hosting the BL tier (i.e., VM_1) of the application under test over the next 5 seconds as the control interval.

Figures 7.22 to 7.24 show the results for FC and NFC scenarios with different desired RTs under open and closed system models for Olio application. In general, the measured RTs remain stable and close to the desired values under both system models. Moreover, the RTs converge to the desired values immediately, mostly without being noticed, after detecting a sudden increase or decrease in workload for FC and NFC

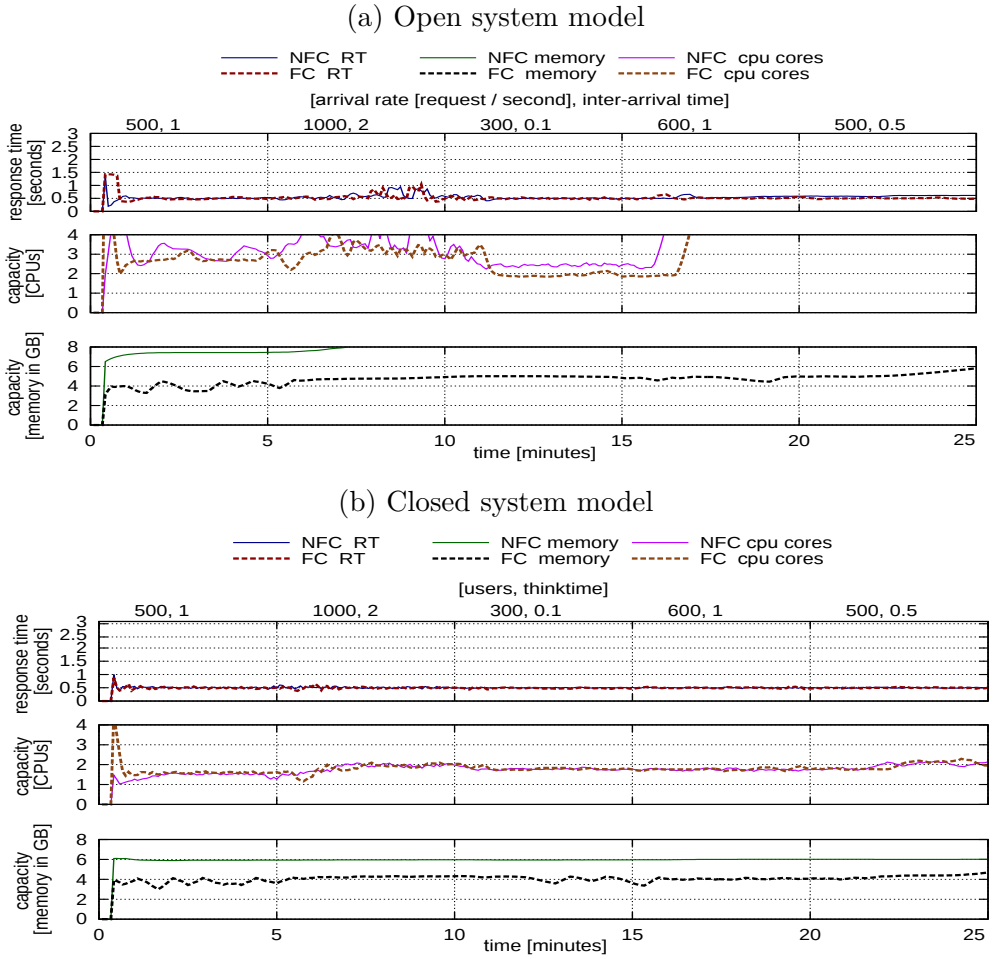


Figure 7.22: Olio, under open and closed system models with 0.5sec desired RT

scenarios. However, NFC most of the time over-provisions both memory and CPU. This is due to asynchronous decision making by the CPU and memory controllers. As a result, each controller assumes that the performance degradation is due to the lack of resource controlled by itself. Since *cpu controller* is more reactive than *memory controller* due to the different nature of the controlled resources, it can adapt the allocated CPU more quickly with the right number of cores. While *memory controller* is more conservative for memory adjustment and it remains in over-provisioning state because it needs to consider the performance stability of its decision. In general, FC allocates the right amount of both resources without over- or under-provisioning any of the resources.

The other important point to note is that both FC and NFC properly detect and adapt to the CPU capacity required to meet the target RTs for both open and closed model systems. Indeed, as it can be observed from the plots presented in Figures 7.22 to 7.24, the open system model requires more capacity compared to the closed system model for similar configurations. This is because the closed system model waits for the

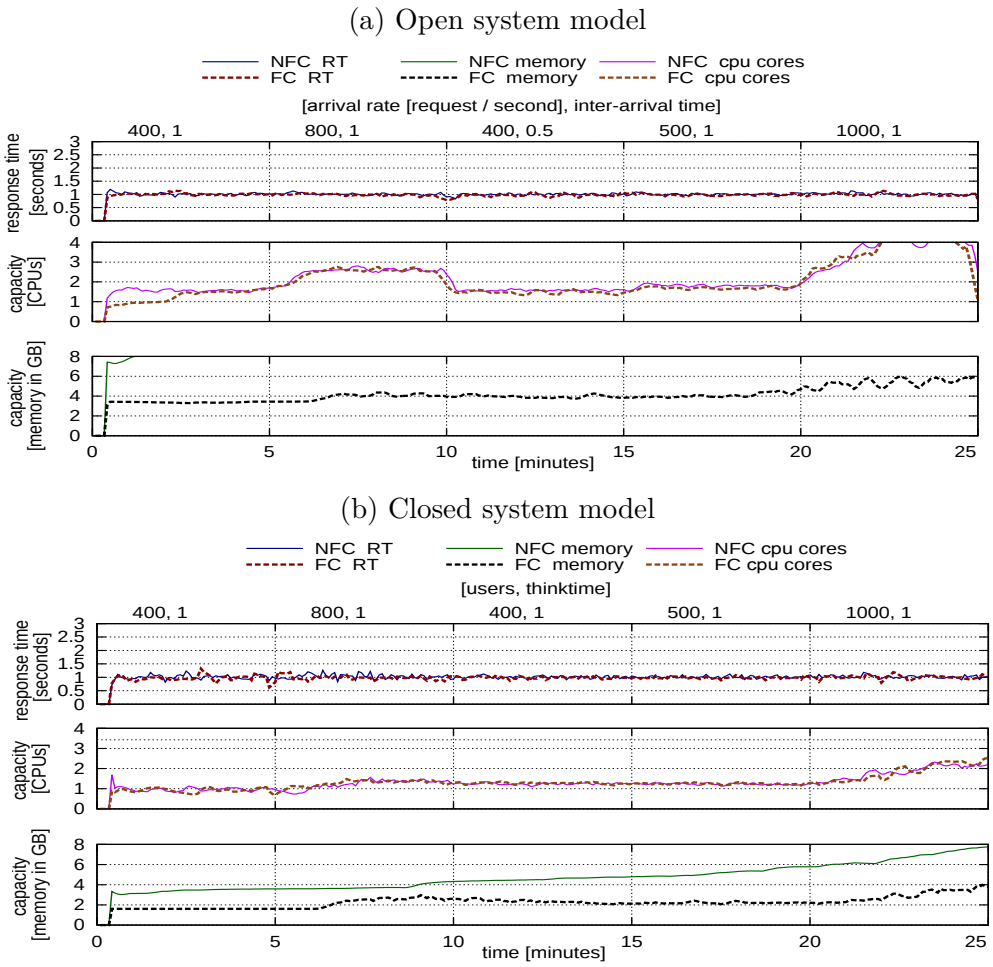


Figure 7.23: Olio, under open and closed system models with 1.0sec desired RT

thinktime after getting a response from the system which reduces the intensity of the workload. However, memory is over-provisioned under NFC for the reason explained above. On the contrary, FC allocates the right amount of memory required to meet the target RT for both open and closed system models due to the coordination of the memory and CPU controllers.

We also run the experiments with RUBiS and RUBBoS applications to observe the behavior of the proposed coordination approach with applications which have different types of resource needs. However, we only present time-series plots for desired RT of 0.5sec and 1.0sec for RUBiS and RUBBoS, respectively. As can be observed from Figures 7.25 and 7.26, while NFC does not behave well, since it over-provisions memory for RUBiS application, FC remains stable since provisioning of the resources is synchronized. The over-provisioning of both CPU and memory is lower in the case of RUBBoS.

In general, FC remains stable both in terms of achieving performance targets and

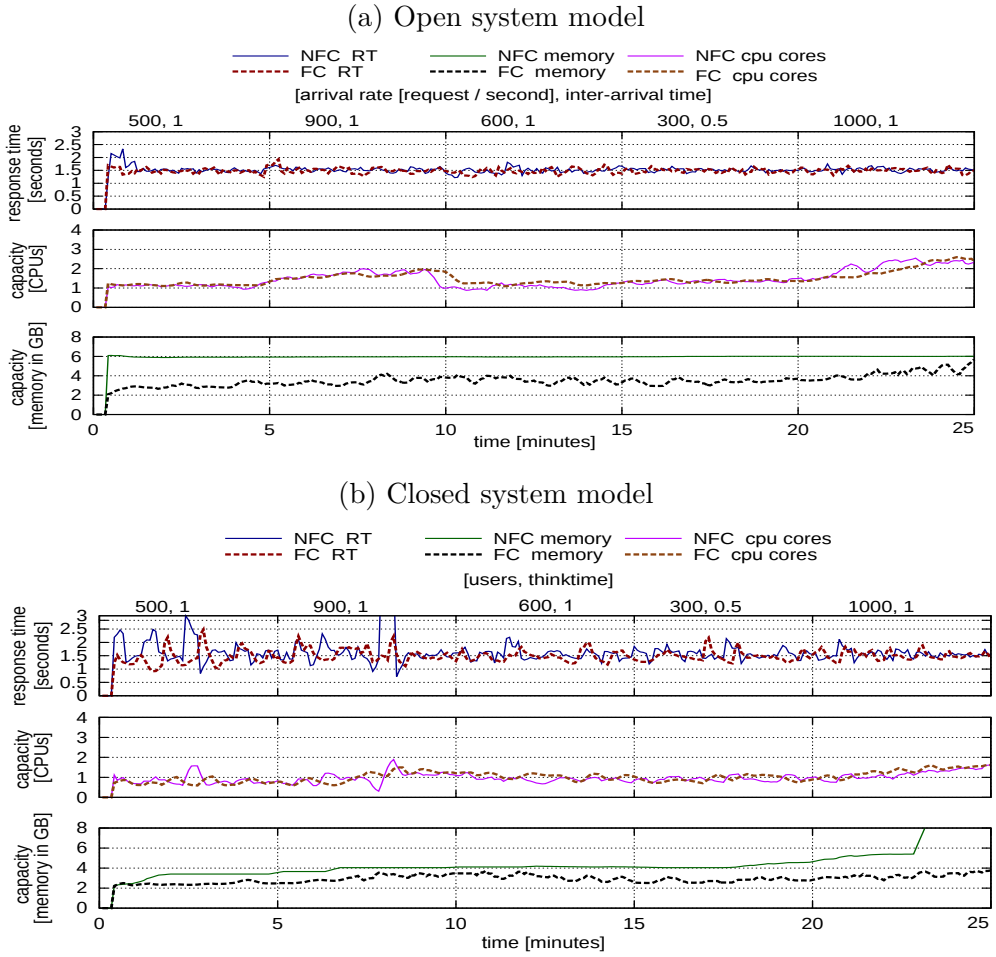


Figure 7.24: Olio, under open and closed system models with 1.5sec desired RT

avoiding resource over- and under-provisioning irrespective of the target values under both system models. On the other hand, most of the time, NFC makes inconsistent decisions. In general, it is only *cpu controller* that does its job under NFC while *memory controller* is usually over-provisioning the memory because of its slow reaction due to the fact that memory is released or reclaimed by the application slowly. Moreover, there are some instances where the experiments are not able to be completed under NFC as a result of the application crashing due to low memory allocation (under-provisioning). Thus, the behavior of NFC is non-deterministic from one run to the other for the same workload pattern and with the same configurations.

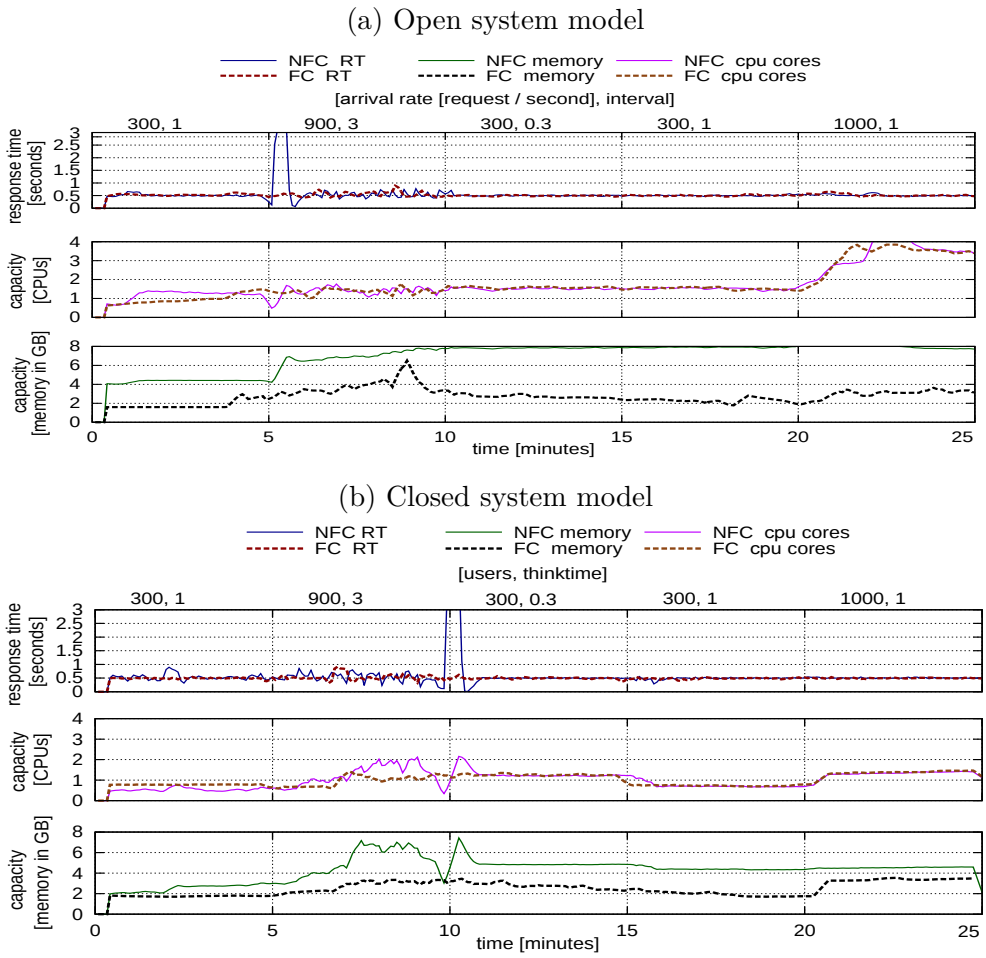


Figure 7.25: RUBiS, under open and closed system models with 0.5sec desired RT

Aggregate analysis

To assess the aggregate behaviors of FC in comparison with NFC over the course of the experiments, we report the mean values for the CPU and memory allocations along with the values for the mean and standard deviation of RT. Moreover, we also present two control theoretic metrics which measure the total error observed during the lifespan of the system under test. These metrics are ISE, presented previously in Equation (7.4), and integral of the absolute error (IAE) computed as shown in Equation (7.6), where $e(t) = \tilde{rt} - rt(t)$.

$$IAE = \sum |e(t)| \tag{7.6}$$

Table 7.12 shows the aggregate results of the three applications under different desired RTs and system models. As reported in Table 7.12, for almost all the benchmark applications the ISE and IAE are relatively smaller for FC compared to NFC under both system models. This implies that our novel FC takes better decisions than NFC. In some of the instances where NFC has smaller values of the aggregate errors, it has higher

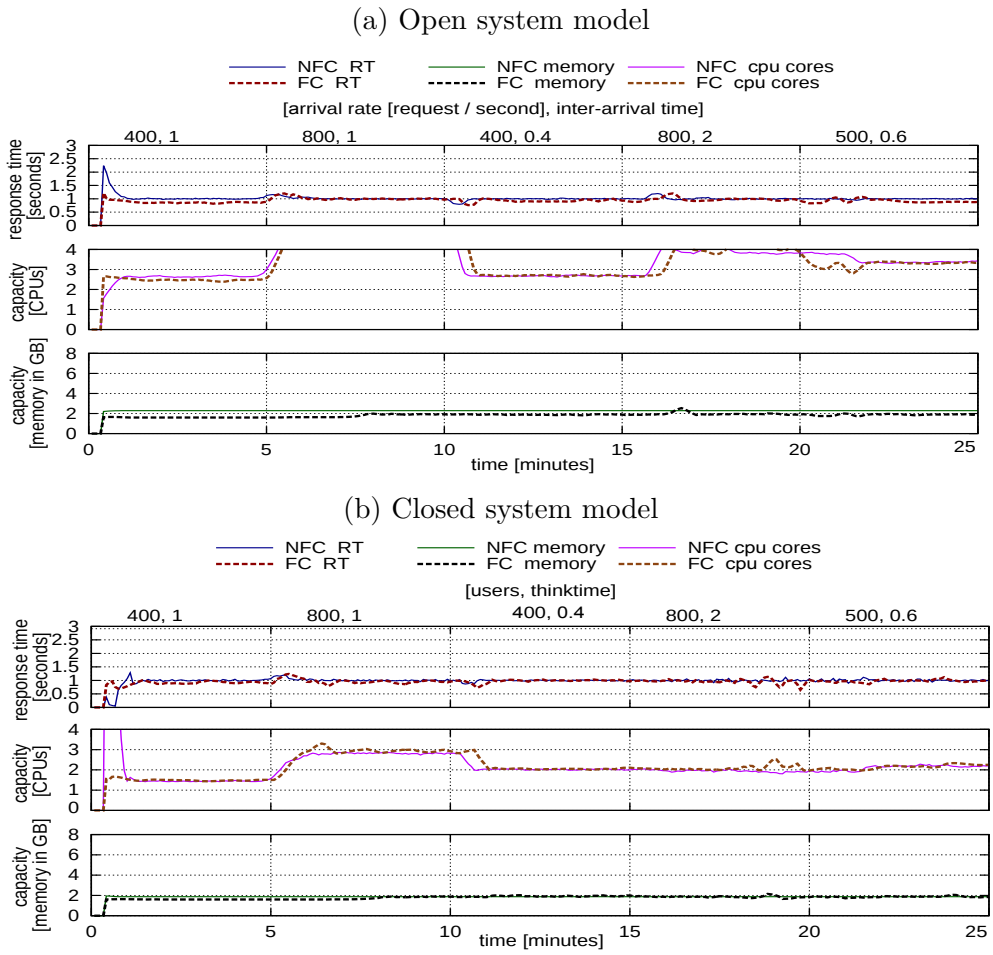


Figure 7.26: RUBBoS, under open and closed system models with 1sec desired RT

resource allocations and standard deviation indicating that the allocations are either over-provisioned or under-provisioned during the experiment. Generally, the average resources allocated under FC are almost always less than under NFC. This difference is significant in some experiments for either of the resources under FC in comparison with NFC. Figure 7.27 presents the percentage of improvement for FC compared to NFC in terms of allocated resources and stability of the RT, i.e., lower value of the standard deviation. In general, it shows that FC is more stable and allocates less resources compared to NFC.

As shown in Figure 7.27, under RUBiS open system model experiment, the allocated memory under FC is less by 60.76% (2.67 GB vs. 6.81 GB) compared to NFC for similar workload pattern while having 64.78% more stability in measured RT. In the case of CPU, while the values for both FC and NFC are comparable, we observe less CPU cores in some experiments under FC such as Olio (0.5 sec, open model) which is allocated 56.51% (4.44 CPU cores vs. 10.21 CPU cores) less CPU cores compared to NFC while again having improved in both allocated memory (18.98%) and stability of RT (29.68%).

application ($\tilde{rt}$)	system model	control mode	ISE	IAE	mem [GB]	CPU [core]	RT [sec]	RT (SD) [sec]
Olio (0.5sec)	open	FC	16.65	41.68	3.70	4.44	0.56	0.22
		NFC	34.69	56.10	4.57	10.21	0.59	0.32
	closed	FC	14.32	34.50	4	1.99	0.50	0.21
		NFC	24.03	52.47	4.73	2.04	0.52	0.27
Olio (1sec)	open	FC	7.60	38.15	3.70	2.21	0.98	0.15
		NFC	37.39	48.15	4.57	2.34	1.01	1.29
	closed	FC	28.12	66.11	4.00	1.39	0.98	0.29
		NFC	34.90	75.21	4.73	1.37	1.04	0.33
Olio (1.5sec)	open	FC	40.45	86.24	3.93	1.86	1.46	0.35
		NFC	34.07	74.91	8.08	1.92	1.50	0.32
	closed	FC	42.36	89.69	2.25	1.80	1.47	0.36
		NFC	39.48	84.17	4.81	1.86	1.51	0.35
RUBiS (0.5sec)	open	FC	25.25	42.43	2.67	1.70	0.55	0.27
		NFC	190.66	60.10	6.81	1.75	0.56	0.76
	closed	FC	16.52	43.15	2.19	0.91	0.50	0.22
		NFC	355.89	81.94	4.19	0.90	0.56	1.04
RUBBoS (1sec)	open	FC	7.04	32.51	1.78	3.50	0.93	0.14
		NFC	10.95	21.32	2.23	3.50	1.02	0.18
	closed	FC	13.04	40.52	1.76	2.16	0.95	0.20
		NFC	15.61	39.45	1.83	2.33	0.98	0.22

Table 7.12: Aggregate results under the two system models for Olio, RUBiS and RUBBoS

Besides, even though the aggregate mean RTs are relatively comparable for both FC and NFC, the standard deviation for FC is relatively smaller than NFC, i.e., more stability in the case of RT. This implies that the resources allocated under NFC are less well matched with the needs compared to FC. Moreover, it also indicates that there are more oscillations under NFC than FC due to improper predictions of the resources.

In general, in all scenarios under NFC, more CPU and memory are allocated on average during the experiment than with FC under similar configurations despite the fact that the aggregate mean RTs are comparable. These results clearly reveal that by using our novel FC the target RTs are met with lower oscillation (lower values for the standard deviation of RT) while avoiding under- and over-provisioning of either of the resources. This happens due to *fuzzy controller* that coordinated the level of requirements of each resource. Thus, FC is an effective approach that meets the target RTs slightly better, using a substantially lower amount resources.

7.5.3 Discussion

Experiments highlight the need for coordination of the two vertical controllers acting on different resources for the same goal, i.e., meeting the application performance objectives. Specifically, coordination among different controllers prevents conflicting decisions such as under- or over-provisioning of one or more of the resources while avoiding performance violations. The experimental results reveal that under the open system model relatively more improvement is achieved in terms of allocation of both resources and stability in term of application RT (see Figure 7.27). On the contrary, uncoordinated decision making can lead to non-deterministic behavior due to the fact that each controller makes a decision in isolation without considering the effect of the other. That is why the novel FC

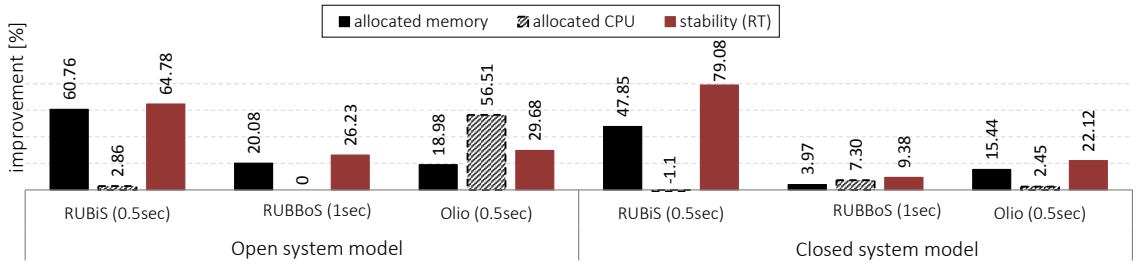


Figure 7.27: Aggregate results: Improvement achieved by using FC compared to NFC

is able to predict the right amount of resources required to satisfy applications' demands under a variety extreme conditions using workloads generated based on open and closed system models. On the other hand, the behavior of NFC, as a baseline approach, is non-deterministic leading to either resource over-provisioning or under-provisioning. In general, FC is able to adapt the resources by observing the applications' performance, and resources utilization without needing to be explicitly notified about changes in the workload patterns. In summary, the experimental results reveal these *key findings*:

- The behavior of NFC is non-deterministic leading to either over provisioning or under-provisioning of memory. The application performance is met during over-provisioning of memory as the application performance is literally controlled by *cpu controller*. However, this leaves ample unused memory that could have been used by other VMs of the same PM. On the other hand, the most serious issue is when memory is under-provisioned which leads the application to crash.

- FC is able to meet the application performance while resources are efficiently utilized. This is because FC is able to reason about the contribution of each resource under uncertainty (using fuzzy logic) by observing the corresponding average utilization values and application RT.

- FC maintains a high utilization of resources. The proposed FC not only guarantees the application performance, but also achieves high utilization of resources as they are used as the decision making criteria. Thus FC provides a win-win scenario for both application owners and also cloud infrastructure providers.

In summary, depending on the nature of the workload, an application can intensively need arbitrary combinations of resources (e.g., CPU or memory) at different stages of its execution. Therefore, uncoordinated deployment of resource elasticity controllers that control different resources for the same application may lead to unpredictable behavior such as resource over-provisioning, which forces customers to pay for unused resources, or application crashing due to severe resource shortage as a result of conflicting decisions. To overcome such issues, careful coordination of controllers ensures achieving the application performance objective with optimal amount of resources, preventing over- and under-provisioning.

CHAPTER 8

Related Work

In this chapter, we present the related work, organized into the following structure: Section 8.1 presents the literature on managing cloud data centers as well as the simulation tools that can be used in this area. Section 8.2 describes the existing work related to the service selection and SLA management in a cloud environment and in particular multi-cloud. Section 8.3 presents related work on cloud resource elasticity, including vertical elasticity of CPU, memory and multiple resources. Since control theory is used in our work to realize the resource elasticity, in Section 8.4, we carefully explore the most relevant control-theoretic approaches, including fuzzy control that have been applied to enable resource elasticity in the cloud computing domain.

8.1 Cloud Infrastructure Management

In this section we first give an overview of existing approaches on cloud management and then focus on existing cloud simulation frameworks.

Although various techniques have been devised for efficient cloud resource management, an effective solution for governing cloud resources in geographically distributed data centers is still an open issue. The current work suffers from shortcomings such as: ignoring the expert knowledge and thereby loosing important information for building efficient system models; not fully addressing cloud management problems, i.e., VM placement [117, 129], temperature-aware energy usage [188], and VM migration [16]. In other words, there is a lack of research that tries to model a combination of these problems while taking into consideration their interconnections and dependencies.

Beloglazov et al. [32] propose a green cloud solution that not only allows to minimize operating cost but also to reduce the environmental impact. Li et al. [116] present a consolidation and forecast-based resource provisioning algorithm that utilizes Bayesian networks. Calcavecchia et al. [38] consider dynamic nature of the incoming stream of VM allocation requests and propose a technique called backward speculative placement

(BSP) that projects the past demand behavior of a VM on a candidate target host. Song et al. [168] state that the key improvement of resource utilization and service throughput depends on using an optimized dynamic resource allocation method. They propose a two-tier resource allocation mechanism consisting of local and global resource allocation with feedback to provide capacities of concurrent applications. A recent research by Lućanin et al. [124] propose the usage of location- and time-dependent factors to in controlling distributed data centers to enable flexible energy-efficient cloud management via SLA models. Altmann et al. [21] propose a cost model along with a model-based service placement optimization algorithm. Their approach takes into consideration the total cost of all possible service placement options in federated hybrid cloud environments and identifies the optimum placement decision.

Despite the mentioned work, there is not enough research attention on the identification of causal relationships hidden in expert knowledge which can enable a more cost-aware VM placement across geographically distributed data centers. This is the motivation behind modeling such relationships using Bayesian networks and applying multi-criteria decision analysis method to be able to make management decisions under uncertainty.

Cloud infrastructure simulation frameworks

CloudSim [39] provides a tool-kit for modeling and behavior simulation of various cloud components such as data centers, PMs, and VMs. It includes typical cloud features, i.e., VM allocation, cloud federations, and dynamic workloads. It is mainly used to evaluate cloud resource provisioning strategies in a controlled simulated environment. *D-Cloud* [27] is a dedicated test environment, build upon Eucalyptus [140], an open-source software for building cloud computing environments. *D-Cloud* allows to simulate different regular faults in a cloud environment, and to inject them into host operating systems. *PreFail* [99] is a framework for systematic and efficient failure exploration, and validation of correctness of cloud recovery protocols. In comparison with *D-Cloud* that provides simulated actual faults, *PreFail* inserts a failure into the target system or the operating system library. Using such frameworks, cloud testers can flexibly set up different failure scenarios, which is addressed in the scope of this thesis. However, none of the existing tools support features for simulating geographically distributed cloud data centers.

8.2 Multi-Cloud Service Selection

The issues of SLA-based service selection has been widely investigated in both Web service and cloud computing domains in recent years. A thorough comparison of existing approaches dealing with SLA in cloud computing has been done in [110]. Among the research projects introduced in this report, the Contrail project [40, 45] has similar goals to our research on SLA management for composite services in a multiple cloud environment with different resource types. However, our focus is on multi-cloud, while this project works on the federated-cloud model. Moreover, aside the different needs

for SLA interoperability in these two models, the main goal of the Contrail project is to allow cloud providers to seamlessly integrate resources from other clouds with their own infrastructures, and breaks the current customer lock-in situation by allowing live application migration from one cloud to another. However, our goal is to provide a framework, as a middleware for cloud customers, for minimizing the infrastructure leasing cost and SLA violation rate as well as maximizing the customer satisfaction level by utilizing the infrastructure services from a multi-cloud environment.

A comparative review of existing approaches on service selection for composite Web services is given in [133]. In Web service domain, Stephen et al. [190] propose a QoS-based Web service ranking and selection approach. Their approach calculates the satisfaction score of the user for each QoS parameter based on the basis of prospect theory and then aggregates the scores in order to select the service with the highest overall score. Their approach has some significant advantages over other existing work, such as selecting the service that best satisfies QoS requirements concerning by the user, instead of the service with the best QoS, which may lead to an over-qualification and can improve utilization of services. Moreover, by using prospect theory, they model the relation between service QoS parameters and the user satisfaction more precisely. However, in their work, they only focus on a single service selection. In the research carried out in this thesis, we similarly use the principle of prospect theory, but to rank both single cloud services and the composite cloud services.

For a proper service selection in multi-cloud, a methodology is needed to compare cloud services based on the various criteria such as the cost and QoS parameters for different user profiles [148]. In addition, because of the SLA heterogeneity in this environment, SLA management from both customer and provider perspectives is challenging.

Most of the research efforts that focus on the SLA-based service selection and allocation in clouds are focused only on maximizing the customer profit [48] and [44] or cloud infrastructure provider profit [115]. The work in [184] is one of the first research attempts dealing with resource allocation from the cloud customer's perspective. This work is enhanced to support both the end user and the cloud customer (e.g., the application owner) profits in [185]. The authors propose an allocation strategy for the cloud customers to maximize their profit and satisfaction level when deploying their applications in cloud infrastructure services. Their approach also supports the dynamic changing of customer requests with the goal to minimize the number of used VMs. However, response time and service initialization time are the only parameters considered in their proposed SLA used in the service selection. In addition, their evaluation is limited to one cloud with a single requested VM per Service.

Similar work has investigated service allocation in the cloud [166, 58] by providing an SLA-driven resource allocation scheme that selects a proper data center among globally distributed data centers operated by a provider. In contrast, in the research carried out in this thesis, the composite multi-cloud services are proposed which supports more SLA parameters such as availability, latency, reputation, throughput, and cost. The concept of sub-SLA and meta-SLA are first mentioned in the Grid computing domain [143]. The authors use these concepts to schedule jobs in Grids by utilizing a multi-agent system

and an SLA negotiation protocol. The usage of these concepts in a multi-cloud, as the focus of this thesis, significantly differs from this one due to the differences between Grid and cloud business models.

Compared to the previous research approaches, in the scope of this thesis, we propose an approach to assist the cloud customer (an application owner) to select the most suitable cloud infrastructure services from a multi-cloud model while handling the SLA hierarchy and heterogeneity. Here goal is to maximize the cloud customer profit by maximizing its SLA satisfaction level, which can be achieved by utilizing prospect theory in the computation of customer satisfaction.

8.3 Vertical Resource Elasticity

The field of elastic systems in general and elasticity approaches, in particular has been gaining momentum in cloud computing [96], and several approaches based on different frameworks, models and techniques have been applied, turning it into a mature field. In this section, instead of reviewing the breadth of this field, which has been reviewed in [73, 75, 50, 41], we summarize the work on vertical elasticity in cloud computing. In theory, any resource could be elastic, however, the practical exploitation depends on the type of the resource, cost and complexity of the implementation. Since the focus of this thesis is on the vertical elasticity of memory and CPU, in this section, we review work related to these resources.

8.3.1 Memory Elasticity

Baruchi et al. [29] compare two techniques for memory elasticity: (i) based on the concept of moving average (ii) based on Page Faults. They experimentally show that when Page Faults are used to scale memory, the performance is improved in comparison with the exponential moving average technique. Dawoud et al. [49] propose the concept of Elastic VM that supports dynamic resource elasticity feature without rebooting the system. They experimentally demonstrated that Elastic VM architecture requires less consumption of resources and avoids scaling-up overhead while guaranteeing SLAs. The method is claimed to be more suitable for memory elasticity with lower costs and complexity.

Wang et al. [181] use the ability of dynamic memory scaling to improve the performance of data deduplication (a specialized data compression technique for eliminating duplicate copies of repeating data). The amount of memory is dynamically set in accordance to a sampling technique to guarantee the performance of the whole system. Germán et al. [134] use elasticity rules to adapt the VM memory size of the application need. A mechanism is proposed to monitor the VM memory and apply vertical elasticity rules in order to dynamically change the memory size by using the memory ballooning technique provided by KVM hypervisor. Similar to our research in this thesis, their method shows that by adapting the VM memory size, the performance level of the running application can be met. Molto et al. [134] present a mechanism for adapting the VM memory size to the memory consumption pattern of the application by using a simple elasticity rule.

Spinner et al. [169] proposes a proactive vertical memory approach which takes the application performance and the change at the workload in order to adjust the allocated memory at runtime.

The proposed approaches on vertical memory elasticity in this thesis have several distinguishing aspects: (i) because of the special challenges in memory elasticity, research on this topic is scarce compared to other resource elasticity research; (ii) among the work exploring memory elasticity, most of them [85, 118] look at the data storage tier, as the effect of memory on retrieving data is clear; therefore, being concerned with the memory elasticity of the business logic tier (e.g., Apache Web server) has not yet been well investigated; (iii) we look at memory elasticity from different point of view, taking the application response time, instead of memory utilization as the most commonly used indicator of the memory scarcity [29]. In other words, we consider the application performance as a decision making criterion to scale up or down the memory; (iv) applying a generic control design process that guarantees the stability of a controlled system to realize vertical memory elasticity.

8.3.2 CPU Elasticity

Yazdanov and Fetzer [191] develop a specific solution for vertical elasticity of CPU resources. Their solution is built on top of Xen hypervisor using the combination of on-the-fly plugging CPU and tuning virtual CPU power to provide a finer grain control on the physical resources associated to the virtual machine. Spinner et al. [170] propose a model-based approach that uses the relationship between the resource allocation and the observed application performance to automatically extract and update the model using resource demand estimation techniques. This model is then used in a feedback controller to dynamically adapt the number of virtual CPUs of individual virtual machines.

Kalyvianaki et al. [103] supports vertical elasticity by adopting Kalman filtering and statistical approaches to track and control the CPU utilization in virtualized environments in order to realize capacity allocation. Pradeep et al. [144] use two layers of controllers, one to regulate the relative utilization for each tier of a Web application, and a second one to further adjust the allocations in cases of CPU contention. Lakwe et al. [112] present two generic response time performance models, queue length based and inverted response time, which map performance to capacity and provide performance guarantees for interactive applications deployed in the cloud. Their proposed requires only minimal training or knowledge about the hosted applications while simultaneously reacting as quickly as possible to changes in workloads.

8.3.3 CPU and Memory Elasticity

Lu et al. [121] develop a tool to automatically set resource control for both VMs and resource pools to meet performance of the application level, as well as resource pool level. For the former, they translate performance objectives into the appropriate resources, consisting memory and CPU, by controlling the setting of the individual VMs hosting the application. At the resource pool level, they ensure that all important applications

within the resource pool can meet their performance targets by adjusting controls at the resource pool level.

Yixin et al. [53] design a MIMO controller to regulate server CPU and memory utilization within specified QoS value for Apache Web server. They show that the MIMO control technique is able to handle the trade-offs between the speed of metric convergence and sensitivity to random fluctuations while enforcing the desired policies. Apache CloudStack [3], as a recent open source software, tries to add the ability to scale up CPU or memory for running virtual machines based on the predefined compute offerings for different hypervisors (Xen, VMware [156], and KVM).

Some more recent approaches which consider both CPU and memory [53, 3] use resource utilization as a decision making criteria which is oblivious to application performance. Although, Lei et al. [121] propose an application-driven model that tries to ensure response time below a certain threshold, their proposed approach may lead to resource over-provisioning.

8.4 Control-Theoretic Approaches for Cloud Elasticity

Since control theory is the main solution domain that is used to realize vertical elasticity in this thesis, in this section we carefully look at the most relevant control-theoretic approaches that have been applied to enable elasticity for cloud applications. By following this approach, not only can we position the research done in the scope of this thesis into this narrow filed in cloud computing, but we can also provide a clear view of the impact of control theory in this field. Moreover, we briefly introduce work on fuzzy control used in this thesis.

Controller synthesis

A simple yet general controller synthesis approach proposes in [68, 70], which reduces the need for strong mathematical background to devise ad-hoc control solutions. Filieri et al. [67] highlight potential solutions toward application of control theory in construction of adaptive software. They focus on adaptation of a specific class of models and control to achieve reliability properties. Maggio et al. [126] study different self-adaptation strategies and discuss that adaptive and model predictive control systems outperform other approaches in performance aspect on a-priori unknown situations. There are some approaches based on control theory (e.g., [55, 108]) that can enhance cloud applications with the capability to adjust their resources based on changing environmental conditions. Patikirikorala et al. [146] investigate the benefits and limitations of applying feedback controllers in cloud computing platforms, and to this aim, they briefly highlight a few system design requirements. These approaches typically synthesize an elasticity controller to automatically decide when to activate some optional features. The benefit of such approaches is that they allow guaranteeing some specific desirable properties. Although such controllers are resilient against stationary noises, they are proved to be robust against non-stationary uncertainties.

Classic control

Some approaches (e.g., [127, 194]) employ classic control techniques, such as proportional integral derivative (PID), to construct autonomic controllers for adjusting resources. These controllers depend on simple mathematical models and provide formal guarantees on the properties of the controller, but they need to consider some assumptions that constrain their adoption in highly dynamic and volatile environments such as cloud.

Advanced control

Other approaches (e.g., [76, 125, 145]) try to build on classic control theory or classic queuing models, by proposing parametric models where part of the parameters are unknown at design time and can be adjusted by adopting adaptive filters such as Kalman filtering.

Adaptive control

Adaptive control addresses some of the shortcomings of fixed gain controllers by dynamically estimating the model parameters and adjusting the gains of the controller to better estimate the control reference. Therefore, changes in the system model are detected on the fly and incorporated into the controller. A relevant example of such adaptive controller has been employed in [147].

Black box control

Classic control approaches rely on the use of mathematical models that is inherently limited to the domain where it is possible to accurately define a model structure and estimate model parameters. Black box and surrogate models address this challenge by constructing the models from input-output data collected over time, and thus obtaining models that resemble the system by construction. Interestingly such black box approaches has been adopted quite a lot in the context of cloud elasticity, e.g., [172, 74].

Online learning approaches

Some other established techniques, such as machine learning, have been exploited to enhance classic controllers. Classic controllers augmented with learning capabilities increase the adaptability of the underlying static technique. Such an approach allows the control solutions to deal with unseen and emerging behaviors that may differ from the design-time assumptions. Machine learning approaches can be categorized as model-based and model-free, depending on the use of analytical models. The most popular model-based approaches use artificial neural networks [125], while popular model-free techniques use clustering to discover new control rules [189]. In model-based approaches, the accuracy of the control actions is proportionally related to the model structure and the training data [125]. In model-free solutions, the accuracy depends on the learning rate and the size of the action-configuration space [75].

Fuzzy control

The main difference between the traditional model-based control theory approaches and knowledge-based control approaches is that model-based approaches assume that a precise mathematical model of the system to be controlled is explicitly available. Whereas, the knowledge-based control does not make such an assumption but rely on expert knowledge [96]. Deriving an accurate mathematical model of the underlying software is a daunting task due the non-linear dynamics of real systems [89, 194]. Fuzzy control is a known knowledge-based control approach which has been applied for dynamic resource allocation in cloud [189]. In fuzzy control, which is typically called as model-free approach, such non-linear functions of the target system is implicitly constructed through fuzzy rules and fuzzy inference by imitating human control knowledge. Although this facilitate knowledge elicitation from users, such approaches are still dependent on users' inputs. Some approaches tackle this problem by entangling the fuzzy control with machine learning techniques [154, 113].

CHAPTER 9

Conclusion

In this chapter, we present the main conclusion of our research by highlighting the significance of this thesis in terms of summarizing the contributions and their implications for the advancement of QoS control in cloud environments (Section 9.1). The limitations of the thesis are discussed in Section 9.2. Finally, Section 9.3, states the ongoing trends and open topics in related research areas for future research to build upon the contributions presented in this work.

9.1 Summary

This thesis addresses controlling the trade-off between QoS and cost in cloud environments from two different perspectives: (i) the cloud infrastructure provider, who aims to provide high quality services while trying to minimize the operating costs of cloud infrastructure; (ii) the cloud customer, who targets the cloud to achieve the cost and QoS benefits. The introduced contributions of this thesis are aligned with these two perspectives.

First, we tackle the problem of managing geographically distributed cloud infrastructures in a QoS-aware and cost-effective manner, taking into account the time- and location-based parameters in such infrastructures, such as regional electricity prices and temperature. To this aim, a VM placement approach consisting of VM allocation and consolidation algorithms is proposed to enable cloud providers in reducing the operating costs while providing high QoS for their customers. Then, by taking the cloud customer viewpoint, e.g., an application owner, we propose a multi-cloud service selection approach, which aims to maximize the benefit of the customer in terms of QoS and cost. To this aim, the proposed approach selects and composes the best combination of services in a multi-cloud model. Furthermore, we argue that the existing cloud providers do not offer any performance guarantees for their services, while for a cloud customer a poor application performance can easily reduce the revenue by killing the satisfaction of end users [136, 79, 119]. We address this gap by proposing solutions in which the

cloud application can become self-adaptive. This is realized by coupling the application with elasticity controllers, which continuously monitors the application performance and dynamically adjusts the resource allocations according to the varying workload in order to meet the application performance objectives.

In the remaining of this section, we provide more details in order to summarized the contributions that previously elaborated in Chapters 3 to 6, and evaluated in Chapter 7.

Controlling the trade-off between QoS and cost for the cloud provider

We propose and implement a novel virtual machine placement approach that enable the provider who operates geographically distributed infrastructures to control the operating costs (i.e., power, cooling, and SLA violation penalty) while keeping the customers satisfied in terms of the provided QoS. This approach creates a decision model using Bayesian networks, and applies multi-criteria decision analysis method along with two proposed algorithms for the VM allocation and consolidation. For the evaluation, we focus on geographically distributed cloud data centers that experience frequent power outages. The proposed approach is evaluated in a simulation setup in comparison with two state-of-the-art baseline algorithms. The results show that the proposed approach decreases the energy cost by up to 69% in comparison with the first baseline approach, and by up to 45% compared to the second baseline approach.

Controlling the trade-off between QoS and cost for the cloud customer

To enable the cloud customer to have a wider range of QoS and cost, we propose a multi-cloud service allocation framework includes both design-time and runtime activities. In the scope of this thesis, we mainly focus on developing a design-time multi-cloud service selection approach. By using the proposed approach, the cloud customer can find and compose the best set of services from multiple cloud providers with respect to the QoS and cost requirements. The proposed service selection approach utilizes prospect theory to score the alternative service offerings based on the given customer SLA. For the evaluation, the proposed algorithm is compared with a state-of-the-art utility-based algorithm as the baseline in a realistic simulation environment using a computer-aided design application use case. The evaluation results show that the proposed approach effectively selects and composes a set of services that best satisfy the SLA within a specified leasing budget, requested by the cloud customer.

In order to make cloud applications self-adaptive, we follow a control design process used in the control theory domain, to synthesize a feedback loop controller, named as memory controller. The designed memory controller auto-scale the allocated memory of the virtual machine hosting the cloud applications to guarantee the application performance objectives while provisioning memory as the required cloud resource according the application demands at runtime. We experimentally evaluate the designed controller in a virtualized environment using a cloud benchmark interactive application. We conduct a set of experiments under two real-world workload traces. We compare the application response time and the resource usage of a self-adaptive application (equipped with the

proposed controller) with a non-adaptive application (under- and over-provisioned the memory). The experimental results reveal that the controller is able to efficiently save at least 47% memory usage while meeting the application performance objectives.

Furthermore, following the same control design process, we improve the memory controller by the idea of taking both application performance and resource utilization at runtime as decision making criteria for auto-scaling the allocated memory, named as hybrid memory controller. The aim of the hybrid memory controller is to satisfy the application performance objectives while achieving a high resource utilization at runtime in spite of varying workloads. For the evaluation, the results achieved by the hybrid controller are compared in an experimental setup with the results of two baseline controller using a cloud benchmark interactive application deployed in a virtualized environment. The results reveal that the hybrid memory controller achieves a relatively high memory utilization (close to 84%), while allocating the lowest amount of memory, and having a high performance stability (i.e., standard deviation of response time) compared to the two baseline controllers. Generally speaking, such a controller can be used to make a cloud application self-adaptive and to guarantee the application performance objectives while decreasing the cost in terms of resource usage, i.e., achieving a high resource utilization for the application owner.

Afterwards, in order to meet the application performance objectives by auto-scaling of multiple resources at runtime, we propose a fuzzy coordination approach encompasses three sub-controllers: fuzzy controller, CPU controller, and memory controller. The fuzzy controller acts as a coordinator so that the control actions of the CPU and memory controllers complement each other in order to fulfill the application's performance objectives. The CPU controller and memory controller determine the right amount of CPU and memory, respectively taking the fuzzy controller's output as an indicator. We evaluate the proposed solution using three widely used cloud benchmark interactive applications in a virtualized environment. Different experiments are conducted under workload traces generated based on open and closed system models. The results show that the proposed coordination solution is able to maintain the desired performance with fewer control errors and more efficient resource usage, e.g., up-to 60% less memory usage in one scenario, and up-to 56% less CPU usage in another scenario compared to a non-fuzzy approach used as a baseline. Although the proposed fuzzy coordination approach is applied for coordinating CPU and memory controllers in our research, it is generic and can be used for any other cloud resources. Moreover, the proposed approach can be also support the coordination of more than two controllers.

9.2 Limitations of this Thesis

Although some significant results of our proposed solutions have been demonstrated, it is important to pinpoint the limitations of this thesis. In this section, we state a number of notable limitations, which highlight observations that are out of scope in our considerations and remain for future research.

- As for the proposed solution addressing the cloud infrastructure management, the current designed Bayesian network only supports the discrete values; therefore all the continuous input parameters need to be discretized. Such conversions may lead to losing the accuracy of the output, and consequently can cause making inefficient virtual machine placement decisions in some circumstance.

- The proposed multi-cloud service allocation framework includes both design-time and runtime activities, while in the scope of this thesis, we cover mainly the design-time part. While having the best set of selected services, activities such as monitoring the running services, detecting the SLA violation, and handling the interoperability issues are challenging in a multi-cloud environment, but they are out of the scope of this thesis. Moreover, although prospect theory has been widely accepted and used in economics, the evaluation of its effectiveness and accuracy in regarding to its influence on cloud service ranking still needs more studies and investigation.

- In the scope of this thesis, we argue that vertical elasticity, and in particular memory elasticity, is a topic that is not well explored by the state-of-the-art research work. Moreover, our research is among the first attempts in applying control theoretical approaches to realize vertical resource elasticity where the priority is about guaranteeing the application performance. However, relying only on vertical resource elasticity may be insufficient for accommodating large change of runtime workload. In such case, utilizing horizontal elasticity can be a more reliable solution, which is out of the scope of our research in this thesis.

- Although the achieved performance violation rate by using the proposed memory controllers as reactive approaches, are very low, even this amount might be unacceptable for the industrial usages, as the money which can be saved by using such controllers may not compensate the degradation in the satisfactions of users. Therefore, using a proactive approach that utilizes some estimation methods for the workloads, to address this concern would be superior. However, one can configure these controllers with a value lower than the desired output. This way, before the controlled output goes outside of its desired regime, corrective action can be triggered by the controller. i.e., explicitly, the controller can behave proactively. Moreover, the system model used in these controllers is not necessary has to capture the exact relationship between the controller's output and the measured output, and a rough estimation is enough to tune the controller [68]. However, the fact that the used model rebuilding mechanism in these controllers is linear regression may limit their abilities in highly dynamic runtime situations such as bursty workload.

- Apart from the challenges addressed throughout the controller designing process, there are still several technical constraints that should be carefully considered while focusing on the memory elasticity: (i) depending on the memory allocation strategy used in hypervisors, reducing memory size may not be beneficial for the host operating system. However, in the case of the used hypervisors and using

ballooning mechanism, the released virtual machine memory size can be used for other co-located virtual machines; (ii) even when the memory size can be changed at the operating system level, some applications cannot still support the dynamic memory allocation and eventually need to be restarted to take advantages of the new allocated memory, such as Java virtual machine (JVM) applications. However, in the scope of this thesis, we focus on the business logic tier that host the Web server, which can leverage the new allocated memory in a dynamic and live manner.

- The fuzzy coordination approach presented in this thesis for coordinating multiple controllers toward the same performance goal, uses a fuzzy knowledge-based, including the fuzzy rules and membership functions, which are extracted at design time and are not updated at runtime. Therefore, the proposed approach may not able to act as expected if the system enters a new circumstance, e.g., sudden and unexpected change in the nature of the application workload at runtime.

9.3 Future Work

As discussed in the previous section, it can be observed that some important issues are out of the scope of our proposed solutions in this thesis. These issues imply the following open research directions.

- To more efficiency conduct virtual machine placement across geographically distributed cloud infrastructures, utilizing a hybrid Bayesian networks can be considered. The hybrid Bayesian networks can support both continuous and discrete input parameters, so it can more realistically analysis the input parameters and thereby generate a more accurate output. This can eventually lead to make more precise virtual machine placement decisions.

- Following the runtime activities of the proposed multi-cloud service allocation framework, including SLA monitoring, and SLA violation detection, is a future research direction in our work. More specifically, possible monitoring strategies can be used in this phase, including developing APIs to provide a unified monitoring on multiple cloud providers, or enabling trusted third parties (TTP) to undertake the monitoring responsibilities [110]. SLA validation tracking can be done by utilizing abstract behavioral specification (ABS) language [98]. ABS is a high-level, executable programming languages, which is used to support full code generation and timed validation of models [17]. At runtime phase, ABS language can be used for the validation of SLAs in a multi-cloud environment. SLA violation detection or proactively detecting the future violations can be done by reasoning on the monitored information. Modeling the SLA detection problem as a root cause analysis problem in Bayesian networks [153] seems a promising solution. Strategies such as migrating to another cloud service provider, or defining a penalty model to decrease the provider's reputation can be applied as a compensation model.

- The proposed memory controllers can be enriched by these future directions: (i) using techniques such as Kalman Filtering to more precisely rebuild the system model and hereby be able to more robustly handle the workload fluctuating; (ii) taking into account the application workload prediction to proactively support the resource elasticity; (iii) applying the designed controllers on a multiple cloud environment.
- The proposed fuzzy coordination approach can be improved in various ways such as: (i) enhancing the fuzzy controller with features such as online learning, e.g., using reinforcement learning, for adaptation of the fuzzy rules and membership functions at runtime; (ii) extending the proposed solution to complement vertical elasticity with horizontal elasticity; (iii) extending the coordination approach to be able to dynamically support more distributed controllers in order to pave the way for complex, and large-scale self-adaptive systems.

Bibliography

[1] Apache Performance Tuning. Available online: http://www.devside.net/articles/apache-performance-tuning, Visited: Nov 2015.

[2] CloudFlow Project. Available online: http://eu-cloudflow.eu Visited: Nov 2015.

[3] Dynamic Scaling of CPU and RAM. Available online: https://cwiki.apache.org/confluence/display/CLOUDSTACK/Dynamic+scaling+of+CPU+and+RAM, Visited: Nov 2015.

[4] E-Pricing. Available online: http://en.wikipedia.org/wiki/Electricity_pricing Visited: Nov 2015.

[5] FIFA 1998 Web site Page View Statistics. Available online: http://ita.ee.lbl.gov/html/contrib/WorldCup.html, Visited: Nov 2015.

[6] forecast Weather Web Service. Available online: http://www.forecast.io Visited: Nov 2015.

[7] Fuzzylite: a Fuzzy Logic Control Library. Available online: http://www.fuzzylite.com, Visited: Nov 2015.

[8] Metric Choice Matters for Intelligent Auto-Scaling. Available online: http://elastisys.com/2015/08/24/metric-choice-matters-for-intelligent-auto-scaling/, Visited: Nov 2015.

[9] Olio. Available online: http://incubator.apache.org/projects/olio.html, Visited: Nov 2015.

[10] Page View Statistics for Wikimedia Projects. Available online: http://dumps.wikimedia.org/other/pagecounts-raw, Visited: Nov 2015.

[11] RUBBoS. Available online: http://jmob.ow2.org/rubbos.html, Visited: Nov 2015.

[12] RUBiS: Rice University Bidding System. Available online: http://rubis.ow2.org, Visited: Nov 2015.

[13] Standard Performance Evaluation Corporation: HP ML110. Available online: https://www.spec.org/power_ssj2008/results/res2011q1/power_ssj2008-20110127-00342.html Visited: Nov 2015.

[14] World Electrical Outages Statistics. Available online: http://www.nationmaster.com/country-info/stats/Energy/Electrical-outages/Days Visited: Nov 2015.

[15] Orna Agmon Ben-Yehuda, Muli Ben-Yehuda, Assaf Schuster, and Dan Tsafrir. The Resource-as-a-service (RaaS) Cloud. In *Proceedings of the 4th conference on Hot Topics in Cloud Ccomputing*, page 12, 2012.

[16] Sherif Akoush, Ripduman Sohan, Andrew Rice, Andrew W Moore, and Andy Hopper. Predicting the Performance of Virtual Machine Migration. In *International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 37–46. IEEE, 2010.

[17] Elvira Albert, Frank S de Boer, Reiner Hähnle, Einar Broch Johnsen, Rudolf Schlatte, S Lizeth Tapia Tarifa, and Peter YH Wong. Formal Modelling and Analysis of Resource Management for Cloud Architectures: An Industrial Case Study using Real-Time ABS. *Service Oriented Computing and Applications*, pages 1–17, 2014.

[18] Mohammed Alhamad, Tharam Dillon, and Elizabeth Chang. Conceptual SLA Framework for Cloud Computing. In *International Conference on Digital Ecosystems and Technologies (DEST)*, pages 606–610. IEEE, 2010.

[19] Ahmed Ali-Eldin, Johan Tordsson, Erik Elmroth, and Maria Kihl. Workload Classification for Efficient Auto-Scaling of Cloud Resources. Technical report, Department of Computer Science, Umea University, Umea, Sweden, 2013.

[20] Paul Allen. *Service Orientation: Winning Strategies and Best Practices*. Cambridge University Press, 2006.

[21] Jörn Altmann and Mohammad Mahdi Kashef. Cost Model Based Service Placement in Federated Hybrid Clouds. *Future Generation Computer Systems (FGCS)*, 41:79–90, 2014.

[22] D Amrhein and P Anderson. Cloud Computing Use Cases White Paper Version 4.0. Technical report, Cloud Computing Use Cases Group, 2010.

[23] Theodore Wilbur Anderson. *The Statistical Analysis of Time Series*, volume 19. John Wiley & Sons, 2011.

[24] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, et al. A View of Cloud Computing. *Communications of the ACM*, 53(4):50–58, 2010.

[25] John Asker and Estelle Cantillon. Properties of Scoring Auctions. *The RAND Journal of Economics*, 39(1):69–85, 2008.

[26] Karl Johan Astrom and Richard M Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. Princeton University Press, 2010.

[27] Takayuki Banzai, Hitoshi Koizumi, Ryo Kanbayashi, Takayuki Imada, Toshihiro Hanawa, and Mitsuhisa Sato. D-Cloud: Design of a Software Testing Environment for Reliable Distributed Systems using Cloud Computing Technology. In *International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, pages 631–636. IEEE, 2010.

[28] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and The Art of Virtualization. *SIGOPS Operating Systems Review*, 37(5):164–177, 2003.

[29] Artur Baruchi and Edson Toshimi Midorikawa. A Survey Analysis of Memory Elasticity Techniques. In *Euro-Par Parallel Processing Workshops*, pages 681–688. Springer, 2011.

[30] Debasish Basak, Srimanta Pal, and Dipak Chandra Patranabis. Support Vector Regression. *Neural Information Processing-Letters and Reviews*, 11(10):203–224, 2007.

[31] Ewnetu Bayuh Lakew, Cristian Klein, Francisco Hernandez, and Erik Elmroth. Performance-Based Service Differentiation in Clouds. In *International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, pages 505–514. IEEE, 2015.

[32] Anton Beloglazov, Jemal Abawajy, and Rajkumar Buyya. Energy-Aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing. *Future generation computer systems (FGCS)*, 28(5):755–768, 2012.

[33] Irad Ben-Gal. Bayesian Networks. *Encyclopedia of Statistics in Quality and Reliability*, 2007.

[34] Peter Bodık, Rean Griffith, Charles Sutton, Armando Fox, Michael Jordan, and David Patterson. Statistical Machine Learning Makes Automatic Control Practical for Internet Datacenters. In *Conference on Hot Topics in Cloud Computing*, pages 12–12, 2009.

[35] Ivan Breskovic, Jörn Altmann, and Ivona Brandic. Creating Standardized Products for Electronic Markets. *Future Generation Computer Systems (FGCS)*, 29(4):1000–1011, 2013.

[36] Ivan Breskovic, Michael Maurer, Vincent C Emeakaroha, Ivona Brandic, and Schahram Dustdar. Cost-Efficient Utilization of Public SLA Templates in Autonomic Cloud Markets. In *International Conference on Utility and Cloud Computing (UCC)*, pages 229–236. IEEE, 2011.

[37] Rajkumar Buyya, James Broberg, and Andrzej M Goscinski. *Cloud Computing: Principles and Paradigms*, volume 87. John Wiley and Sons, 2010.

[38] Nicolò Maria Calcavecchia, Ofer Biran, Erez Hadad, and Yosef Moatti. VM Placement Strategies for Cloud Scenarios. In *International Conference on Cloud Computing (CLOUD)*, pages 852–859. IEEE, 2012.

[39] Rodrigo N Calheiros, Rajiv Ranjan, and et. al. CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms. *Software: Practice and Experience*, 41(1):23–50, 2011.

[40] Roberto G Cascella, Lorenzo Blasi, Yvon Jegou, Massimo Coppola, and Christine Morin. Contrail: Distributed Application Deployment under SLA in Federated Heterogeneous Clouds. In *The Future Internet*, pages 91–103. Springer, 2013.

[41] Betty HC Cheng, Rogerio De Lemos, Holger Giese, Paola Inverardi, Jeff Magee, et al. Software Engineering for Self-Adaptive Systems: A Research Roadmap. In *Software Engineering for Self-adaptive Systems*, pages 1–26. Springer, 2009.

[42] Yu Cheng and Weihua Zhuang. Dynamic Inter-SLA Resource Sharing in Path-Oriented Differentiated Services Networks. *Transactions on Networking*, 14(3):657–670, 2006.

[43] Christopher Clark, Keir Fraser, Steven Hand, Jacob Gorm Hansen, Eric Jul, Christian Limpach, Ian Pratt, and Andrew Warfield. Live Migration of Virtual Machines. In *Conference on Symposium on Networked Systems Design and Implementation (NSDI)*, pages 273–286. USENIX Association, 2005.

[44] Kassidy Clark, Martijn Warnier, and Frances MT Brazier. Automated Non-repudiable Cloud Resource Allocation. In *Cloud Computing and Services Science*, pages 168–182. Springer, 2013.

[45] Contrail Consortium. Overview of the Contrail System, Components and Usage. Technical report, Contrail Consortium, 2014.

[46] Georgiana Copil, Daniel Moldovan, Hong-Linh Truong, and Schahram Dustdar. On Controlling Cloud Services Elasticity in Heterogeneous Clouds. pages 573–578, 2014.

[47] Adnan Darwiche. *Modelling and Reasoning with Bayesian Networks*. Cambridge University Press, 2009.

[48] Vahid Dastjerdi. *QoS-Aware and Semantic-Based Service Coordination for Multi-Cloud Environments*. PhD thesis, University of Melbourne, 2013.

[49] Wesam Dawoud, Ibrahim Takouna, and Christoph Meinel. Elastic Virtual Machine for Fine-grained Cloud Resource Provisioning. In *Global Trends in Computing and Communication Systems*, pages 11–25. Springer, 2012.

[50] Rogério De Lemos, Holger Giese, Hausi A Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M Villegas, Thomas Vogel, et al. Software Engineering for Self-Adaptive Systems: A Second Research Roadmap. In *Software Engineering for Self-Adaptive Systems II*, pages 1–32. Springer, 2013.

[51] Frederico Alvares de Oliveira, Thomas Ledoux, and Rémi Sharrock. A Framework for the Coordination of Multiple Autonomic Managers in Cloud Environments. In *International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 179–188, 2013.

[52] Frederico Alvares de Oliveira Jr and Thomas Ledoux. Self-management of Cloud Applications and Infrastructure for Energy Optimization. *SIGOPS Operating Systems Review*, 46(2):10, 2012.

[53] Yixin Diao, Neha Gandhi, Joseph L Hellerstein, Sujay Parekh, and Dawn M Tilbury. Using MIMO Feedback Control to Enforce Policies for Interrelated Metrics with Application to the Apache Web Server. In *Network Operations and Management Symposium*, pages 219–234, 2002.

[54] Schahram Dustdar, Yike Guo, Benjamin Satzger, and Hong-Linh Truong. Principles of Elastic Processes. *Internet Computing*, (5):66–71, 2011.

[55] Xavier Dutreilh, Nicolas Rivierre, Aurélien Moreau, Jacques Malenfant, and Isis Truck. From Data Center Resource Allocation to Control Theory and Back. In *International Conference on Cloud Computing (CLOUD)*, pages 410–417. IEEE, 2010.

[56] Erik Elmroth, Johan Tordsson, Francisco Hernández, Ahmed Ali-Eldin, Petter Svärd, Mina Sedaghat, and Wubin Li. Self-management challenges for multi-cloud architectures. In *Towards a Service-Based Internet*, pages 38–49. Springer, 2011.

[57] Vincent Emeakaroha. *Managing Cloud Service Provisioning and SLA Enforcement via Holistic Monitoring Techniques*. PhD thesis, Vienna University of Technology, 2012.

[58] Vincent C Emeakaroha, Ivona Brandic, Michael Maurer, and Ivan Breskovic. SLA-Aware Application Deployment and Resource Allocation in Clouds. In *Computer Software and Applications Conference Workshops (COMPSACW)*, pages 298–303. IEEE, 2011.

[59] Naeem Esfahani and Sam Malek. Uncertainty in Self-Adaptive Software Systems. In *Software Engineering for Self-Adaptive Systems II*, pages 214–238. Springer, 2013.

[60] Soodeh Farokhi. Towards an SLA-Based Service Allocation in Multi-cloud Environments. In *International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 591–594. IEEE/ACM, 2014.

[61] Soodeh Farokhi, Pooyan Jamshidi, Ivona Brandic, and Erik Elmroth. Self-Adaptation Challenges for Cloud-based Applications: A Control Theoretic Perspective. In *International Workshop on Feedback Computing*, 2015.

[62] Soodeh Farokhi, Pooyan Jamshidi, Ewnetu Bayuh Lakew, Ivona Brandic, and Erik Elmroth. A Hybrid Cloud Controller for Vertical Memory Elasticity: A Control-Theoretic Approach. *Future generation computer systems (FGCS) [submitted in Oct 2015, under review].*

[63] Soodeh Farokhi, Pooyan Jamshidi, Drazen Lucanin, and Ivona Brandic. Performance-based Vertical Memory Elasticity. In *International Conference on Autonomic Computing (ICAC)*, pages 151–152. IEEE, 2015.

[64] Soodeh Farokhi, Foued Jrad, Ivona Brandic, and Achim Streit. HS4MC: Hierarchical SLA-based Service Selection in Multi-Cloud Environments. In *International Conference on Cloud Computing and Services Science (CLOSER)*, pages 722–734, 2014.

[65] Soodeh Farokhi, Ewnetu Bayuh Lakew, Cristian Klein, Ivona Brandic, and Erik Elmroth. Coordinating CPU and Memory Elasticity Controllers to Meet Service Response Time Constraints. In *International Conference on Cloud and Autonomic Computing (ICCAC)*, pages 69–80. IEEE, 2015.

[66] Damián Fernández-Cerero, Alejandro Fernández-Montes, Luis González Abril, Juan Antonio Ortega, and Juan A Álvarez. Fear Assessment: Why Data Center Servers Should be Turned off. In *Computer and Communications Industry Association (CCIA)*, pages 253–256, 2014.

[67] Antonio Filieri, Carlo Ghezzi, Alberto Leva, and Martina Maggio. Self-Adaptive Software Meets Control Theory: A Preliminary Approach Supporting Reliability Requirements. In *International Conference on Automated Software Engineering (ASE)*, pages 283–292. IEEE, 2011.

[68] Antonio Filieri, Henry Hoffmann, and Martina Maggio. Automated Design of Self-Adaptive Software with Control-Theoretical formal Guarantees. In *International Conference on Software Engineering (ICSE)*, 2014.

[69] Antonio Filieri and Martina Maggio. Control Theory meets Software Engineering. Available online: `http://www.martinamaggio.com/dagstuhl/` Visited: Nov 2015.

[70] Antonio Filieri, Martina Maggio, and et. al. Software Engineering Meets Control Theory. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, 2015.

[71] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud Computing and Grid Computing 360-Degree Compared. In *Grid Computing Environments Workshop (GCE)*, pages 1–10. IEEE, 2008.

[72] Armando Fox, Rean Griffith, Anthony Joseph, Randy Katz, Andrew Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, and Ion Stoica. Above The Clouds: A Berkeley View of Cloud Computing. *University of California (UCB/EECS)*, 28:13, 2009.

[73] Guilherme Galante and Luis Carlos E de Bona. A Survey on Cloud Computing Elasticity. In *Conference on Utility and Cloud Computing (UCC)*, pages 263–270.

[74] Alessio Gambi, Giovanni Toffetti, Cesare Pautasso, and Mauro Pezze. Kriging Controllers for Cloud Applications. *Internet Computing*, 17(4):40–47, 2013.

[75] Alessio Gambi, Giovanni Toffetti, and Mauro Pezzè. Assurance of Self-adaptive Controllers for the Cloud. In *Assurances for Self-Adaptive Systems*, pages 311–339. Springer, 2013.

[76] Anshul Gandhi, Parijat Dube, Alexei Karve, Andrzej Kochut, and Li Zhang. Adaptive, Model-driven Autoscaling for Cloud Applications. In *International Conference on Autonomic Computing (ICAC)*, pages 57–64. USENIX, 2014.

[77] David Garlan. Software Engineering in an Uncertain World. In *Workshop on Future of Software Engineering Research*, pages 125–128. ACM, 2010.

[78] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. PRESS: Predictive Elastic ReSource Scaling for Cloud Systems. In *International Conference on Network and Service Management (CNSM)*, pages 9–16, 2010.

[79] Jens Grossklags and Alessandro Acquisti. When 25 Cents is too Much: An Experiment on Willingness-to-Sell and Willingness-to-Protect Personal Information. In *Workshop on the Economics of Information Security (WEIS)*, 2007.

[80] Nikolay Grozev and Rajkumar Buyya. Multi-Cloud Provisioning and Load Distribution for Three-Tier Applications. *Transactions on Autonomous and Adaptive Systems (TAAS)*, 9(3):13, 2014.

[81] Dmytro Grygorenko. Cost-based Decision Making in Cloud Environments using Bayesian Networks. Master thesis, Faculty of Informatics, Vienna University of Technology, Austria, 2014.

[82] Dmytro Grygorenko, Soodeh Farokhi, and Ivona Brandic. Cost-Aware VM Placement Across Distributed DCs using Bayesian Networks. In *International Conference on Economics of Grids, Clouds, Systems and Services (GECON)*, pages 69–80. Springer LNCS, 2015.

[83] Soguy Gueye, Noel De Palma, and Eric Rutten. Component-based Autonomic Managers for Coordination Control. In *Coordination Models and Languages*, pages 75–89, 2013.

[84] Soguy Mak-Karé Gueye, Noel De Palma, Éric Rutten, Alain Tchana, and Nicolas Berthier. Coordinating Self-Sizing and Self-Repair Managers for Multi-Tier Systems. *Future Generation Computer Systems (FGCS)*, 35:14–26, 2014.

[85] Amit Gupta, Ehab Ababneh, Richard Han, and Eric Keller. Towards Elastic Operating Systems. In *Conference on Hot Topics in Operating Systems*, page 16. USENIX, 2013.

[86] Irfan Habib. Virtualization with KVM. *Linux Journal*, 2008(166):8, 2008.

[87] John den Hann. Model Driven Architecture Basic Concepts, 2008. Available online: `http://www.omg.org/mda/specs.htm`, Visited Nov 2015.

[88] David Heckerman. *A Tutorial on Learning with Bayesian Networks*. Springer, 2008.

[89] Joseph L Hellerstein, Yixin Diao, Sujay Parekh, and Dawn M Tilbury. *Feedback Control of Computing Systems*. John Wiley and Sons, 2004.

[90] Nikolas Roman Herbst, Samuel Kounev, and Ralf Reussner. Elasticity in Cloud Computing: What It Is, and What It Is Not. In *International Conference on Autonomic Computing (ICAC)*, pages 23–27, 2013.

[91] Paul Horn. Autonomic Computing: IBM's Perspective on the State of Information Technology. 2001.

[92] Markus C Huebscher and Julie A McCann. A Survey of Autonomic Computing-Degrees, Models, and Applications. *Computing Surveys (CSUR)*, 40(3):7, 2008.

[93] Olumuyiwa Ibidunmoye, Francisco Hernández-Rodriguez, and Erik Elmroth. Performance Anomaly Detection and Bottleneck Identification. *Computing Surveys (CSUR)*, 48(1):4, 2015.

[94] Wassim Itani, Cesar Ghali, Ayman I Kayssi, and Ali Chehab. Accountable Reputation Ranking Schemes for Service Providers in Cloud Computing. In *International Conference on Cloud Computing and Services Science (CLOSER)*, pages 49–55, 2011.

[95] Pooyan Jamshidi. *A Framework for Robust Control of Uncertainty in Self-adaptive Software Connectors*. PhD thesis, Dublin City University, 2014.

[96] Pooyan Jamshidi, Aakash Ahmad, and Claus Pahl. Autonomic Resource Provisioning for Cloud-based Software. In *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 95–104, 2014.

[97] Hua Jin, Hua Zou, Fangchun Yang, Rongheng Lin, and Xinchao Zhao. QoS-Based Service Selection for Multiple Requests with Different Classes. In *International Conference on Service Systems and Service Management (ICSSSM)*, pages 604–609, 2012.

[98] Einar Broch Johnsen, Reiner Hähnle, Jan Schäfer, Rudolf Schlatte, and Martin Steffen. ABS: A Core Language for Abstract Behavioural Specification. In *formal Methods for Components and Objects*, pages 142–164. Springer, 2012.

[99] Pallavi Joshi, Haryadi S Gunawi, and Koushik Sen. PreFail: A Programmable Tool for Multiple-Failure Injection. In *SIGPLAN Notices*, volume 46, pages 171–188. ACM, 2011.

[100] James Joyce. Bayes' Theorem. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. 2008.

[101] Foued Jrad, Jie Tao, Rico Knapper, Christoph M. Flath, and Achim Streit. A Utility-Based Approach for Customised Cloud Service Selection. *International Journal of Computational Science and Engineering*, 2013.

[102] Daniel Kahneman and Amos Tversky. Prospect Theory: An Analysis of Decision Under Risk. *Journal of the Econometric Society (Econometrica)*, pages 263–291, 1979.

[103] Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. Self-Adaptive and Self-Configured CPU Resource Provisioning for Virtualized Servers using Kalman Filters. In *International conference on Autonomic computing (ICAC)*, pages 117–126. IEEE, 2009.

[104] Christos T Karamanolis, Magnus Karlsson, and Xiaoyun Zhu. Designing Controllable Computer Systems. In *Hot Topics in Operating Systems (HotOS)*, 2005.

[105] Jeffrey O Kephart and David M Chess. The Vision of Autonomic Computing. *Computer*, 36(1):41–50, 2003.

[106] Maria Kihl, Erik Elmroth, Johan Tordsson, Karl-Erik Årzén, and Anders Robertsson. The Challenge of Cloud Control. In *Feedback Computing*, 2013.

[107] Wonyoung Kim, Meeta S Gupta, Gu-Yeon Wei, and David Brooks. System Level Analysis of Fast, Per-Core DVFS using On-Chip Switching Regulators. In *International Symposium on High Performance Computer Architecture (HPCA)*, pages 123–134. IEEE, 2008.

[108] Cristian Klein, Martina Maggio, Karl-Erik Årzén, and Francisco Hernández-Rodriguez. Brownout: Building more Robust Cloud Applications. In *International Conference on Software Engineering (ICSE)*, pages 700–711, 2014.

[109] Petri Kontkanen, Petri Myllymäki, Tomi Silander, Henry Tirri, and Peter Grunwald. Comparing Predictive Inference Methods for Discrete Domains. In *International Workshop on Artificial Intelligence and Statistics*. Citeseer, 1997.

[110] Dimosthenis Kyriazis. Cloud Computing Service Level Agreements, Exploitation of Research Results. Technical report, European Commission Directorate General Communications Networks Content and Technology Unit, 2013.

[111] Ewnetu Bayuh Lakew. *Autonomous Cloud Resource Provisioning: Accounting, Allocation, and Performance Control*. PhD thesis, Umea University, Department of Computing Science, 2015.

[112] Ewnetu Bayuh Lakew, Cristian Klein, Francisco Hernandez, and Erik Elmroth. Towards Faster Response Time Models for Vertical Elasticity. In *IEEE Conference on Utility and Cloud Computing (UCC)*, pages 560–565, 2014.

[113] Palden Lama and Xiaobo Zhou. Autonomic Provisioning with Self-Adaptive Neural Fuzzy Control for Percentile-based Delay Guarantee. *Transactions on Autonomous and Adaptive Systems (TAAS)*, 8(2):9, 2013.

[114] Chuen Chien Lee. Fuzzy Logic in Control Systems: Fuzzy Logic Controller II. *Systems, Man and Cybernetics, IEEE Transactions on*, 20(2):419–435, 1990.

[115] Young Choon Lee, Chen Wang, Albert Y Zomaya, and Bing Bing Zhou. Profit-Driven Service Request Scheduling in Clouds. In *International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 15–24. IEEE/ACM, 2010.

[116] Jian Li, Kai Shuang, Sen Su, Qingjia Huang, Peng Xu, Xiang Cheng, and Jie Wang. Reducing Operational Costs through Consolidation with Resource Prediction in the Cloud. In *International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 793–798. IEEE/ACM, 2012.

[117] Kangkang Li, Huanyang Zheng, and Jie Wu. Migration-Based Virtual Machine Placement in Cloud Systems. In *International Conference on Cloud Networking (CloudNet)*, pages 83–90. IEEE, 2013.

[118] Harold C Lim, Shivnath Babu, and Jeffrey S Chase. Automated Control for Elastic Storage. In *International Conference on Autonomic Computing (ICAC)*, pages 1–10. IEEE, 2010.

[119] Greg Linden. Make Data Useful, 2006. Available online: `http://glinden.blogspot.com` Visited: Nov 2015.

[120] Weifeng Liu, Jose C. Principe, and Simon Haykin. *Kernel Adaptive Filtering: A Comprehensive Introduction*, volume 57. John Wiley & Sons, 2011.

[121] Lei Lu, Xiaoyun Zhu, Rean Griffith, Pradeep Padala, Aashish Parikh, Parth Shah, and Evgenia Smirni. Application-driven Dynamic Vertical Scaling of Virtual Machines in Resource Pools. In *Network Operations and Management Symposium (NOMS)*, pages 1–9, 2014.

[122] Qinghua Lu, Xiwei Xu, Liming Zhu, Len Bass, Zhanwen Li, Sherif Sakr, Paul L Bannerman, and Anna Liu. Incorporating Uncertainty into in-Cloud Application Deployment Decisions for Availability. In *International Conference on Cloud Computing (CLOUD)*, pages 454–461. IEEE, 2013.

[123] Dražen Lučanin and Ivona Brandic. Pervasive Cloud Controller for Geotemporal Inputs. *Transactions on Cloud Computing (TCC)*, 2015.

[124] Dražen Lučanin, Foued Jrad, Ivona Brandic, and Achim Streit. Energy-Aware Cloud Management through Progressive SLA Specification. In *Economics of Grids, Clouds, Systems, and Services (GECON)*, pages 83–98. Springer, 2014.

[125] Martina Maggio, Henry Hoffmann, Alessandro V Papadopoulos, Jacopo Panerati, Marco D Santambrogio, Anant Agarwal, and Alberto Leva. Comparison of Decision-making Strategies for Self-optimization in Autonomic Computing Systems. *Transactions on Autonomous and Adaptive Systems (TAAS)*, 7(4):36, 2012.

[126] Martina Maggio, Henry Hoffmann, Marco D Santambrogio, Anant Agarwal, and Alberto Leva. Decision Making in Atonomic Computing Systems: Comparison of Approaches and Techniques. In *International Conference on Autonomic Computing (ICAC)*, pages 201–204. IEEE, 2011.

[127] Martina Maggio, Cristian Klein, and Karl Arzen. Control Strategies for Predictable Brownouts in Cloud Computing. In *International Federation of Automatic Control (IFAC)*, 2014.

[128] Zaigham Mahmood. Cloud Computing: Characteristics and Deployment Approaches. In *International Conference on Computer and Information Technology (CIT)*, pages 121–126. IEEE, 2011.

[129] Seyed Saeid Masoumzadeh and Helmut Hlavacs. Integrating VM Selection Criteria in Distributed Dynamic VM Consolidation using Fuzzy Q-Learning. In *International Conference on Network and Service Management (CNSM)*, pages 332–338. IEEE, 2013.

[130] Joe McKendrick. Cloud Computing's Hidden Green Benefits, 2011. Available online: `http://www.forbes.com/sites/joemckendrick/2011/10/03/cloud-computings-hidden-green-benefits` Visited: Nov 2015.

[131] Peter Mell and Timothy Grance. The NIST Definition of Cloud Computing. *National Institute of Standards and Technology Special Publication*, pages 800–145, 2014.

[132] Aleksandar Milenkoski, Alexandru Iosup, Samuel Kounev, Kai Sachs, Piotr Rygielski, Jason Ding, Walfredo Cirne, and Florian Rosenberg. Cloud Usage Patterns: A formalism for Description of Cloud Usage Scenarios. *arXiv preprint arXiv:1410.1159*, 2014.

[133] Mahboobeh Moghaddam and Joseph G Davis. Service Selection in Web Service Composition: A Comparative Review of Existing Approaches. In *Web Services Foundations*, pages 321–346. Springer, 2014.

[134] Germán Moltó, Miguel Caballer, Eloy Romero, and Carlos de Alfonso. Elastic Memory Management of Virtualized Infrastructures for Applications with Dynamic Memory Requirements. *Procedia Computer Science*, 18:159–168, 2013.

[135] Kevin Murphy. A Brief Introduction to Graphical Models and Bayesian Networks, 1998. Available online: `http://www.cs.ubc.ca/~murphyk/Bayes/bnintro.html` Visited: Nov 2015.

[136] Fiona Nah. A Study on Tolerable Waiting Time: How Long are Web Users Willing to Wait? *Behaviour and Information Technology*, 23(3):153–163, 2004.

[137] Richard Neapolitan. *Learning Bayesian Networks*, volume 38. Prentice Hall Upper Saddle River, 2004.

[138] Neovise. Second-Generation Cloud Computing IaaS Services, What It Means, And Why We Need It Now. Technical report, ProfitBricks IaaS, 2013.

[139] Thomas Damgaard Nielsen, Christian Iversen, and Philippe Bonnet. Private Cloud Configuration with Metaconfig. In *International Conference on Cloud Computing (CLOUD)*, pages 508–515. IEEE, 2011.

[140] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The Eucalyptus Open-Source Cloud Computing System. In *International Symposium on Cluster Computing and the Grid (CCGrid)*, pages 124–131. IEEE/ACM, 2009.

[141] Object Management Group. MDA, the Architecture of Choice for a Changing World, 2015. Available online: `http://www.omg.org/mda/`, Visited Nov 2015.

[142] Academy of Behavioral Finance and Economics. Prospect Theory. Available online: `http://prospect-theory.behaviouralfinance.net/` Visited: Nov 2015.

[143] Djamila Ouelhadj, J Garibaldi, Jon MacLaren, Rizos Sakellariou, and K Krishnakumar. A Multi-Agent Infrastructure and a Service Level Agreement Negotiation Protocol for Robust Scheduling in Grid Computing. In *Advances in Grid Computing*, pages 651–660. Springer, 2005.

[144] Pradeep Padala, Kang G Shin, Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, and Kenneth Salem. Adaptive Control of Virtualized Resources in Utility Computing Environments. In *SIGOPS Operating Systems Review*, volume 41, pages 289–302. ACM, 2007.

[145] Sujay Parekh, Neha Gandhi, Joseph Hellerstein, Dawn Tilbury, T Jayram, and Joe Bigus. Using Control Theory to Achieve Service Level Objectives in Performance Management. *Real-Time Systems*, 23(1-2):127–141, 2002.

[146] Tharindu Patikirikorala and Alan Colman. Feedback Controllers in the Cloud. In *Asia Pacific Software Engineering Conference (APSEC)*, 2010.

[147] Tharindu Patikirikorala, Alan Colman, Jun Han, and Liuping Wang. A Multi-model Framework to Implement Self-managing Control Systems for QoS Management. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 218–227. ACM, 2011.

[148] Dana Petcu. Multi-Cloud: Expectations and Current Approaches. In *International Workshop on Multi-cloud Applications and Federated Clouds*, pages 1–6. ACM, 2013.

[149] Dana Petcu. Consuming Resources and Services From Multiple Clouds. *Journal of Grid Computing*, pages 1–25, 2014.

[150] Daryl C Plummer, David Mitchell Smith, Thomas J Bittman, David W Cearley, David J Cappuccio, Donna Scott, Rakesh Kumar, and Bruce Robertson. Five Refining Attributes of Public and Private Cloud Computing. *Gartner Research*, 2009.

[151] CA Pollino and C Henderson. Bayesian Networks: A Guide for their Application in Natural Resource Management and Policy. Technical report, 2010.

[152] Olivier Pourret, Patrick Naim, and Bruce Marcot. *Bayesian Networks: a Practical Guide to Applications*, volume 73. Wiley Online Library, 2008.

[153] Satyabrata Pradhan, Rajveer Singh, Komal Kachru, and Srinivas Narasimhamurthy. A Bayesian Network Based Approach for Root-Cause-Analysis in Manufacturing Process. In *International Conference on Computational Intelligence and Security (CIS)*, pages 10–14. IEEE, 2007.

[154] Jia Rao, Yudi Wei, Jiayu Gong, and Cheng-Zhong Xu. DynaQoS: Model-free Self-tuning Fuzzy Control of Virtualized Resources for QoS Provisioning. In *International Workshop on Quality of Service (IWQoS)*, pages 1–9. IEEE, 2011.

[155] Christoph Redl, Ivan Breskovic, Ivona Brandic, and Schahram Dustdar. Automatic SLA Matching and Provider Selection in Grid and Cloud Computing Markets. In *International Conference on Grid Computing (GCA)*, pages 85–94. ACM/IEEE, 2012.

[156] Mendel Rosenblum. VMware's Virtual Platform. In *Hot Chips*, volume 1999, pages 185–196, 1999.

[157] Tudor-Ioan Salomie, Gustavo Alonso, Timothy Roscoe, and Kevin Elphinstone. Application Level Ballooning for Efficient Server Consolidation. In *European Conference on Computer Systems (EuroSys)*, pages 337–350. ACM, 2013.

[158] Laura Savu. Cloud Computing: Deployment Models, Delivery Models, Risks and Research Challenges. In *International Conference on Computer and Management (CAMAN)*, 2011.

[159] Bianca Schroeder, Adam Wierman, and Mor Harchol-Balter. Open Versus Closed: A Cautionary Tale. In *Networked Systems Design and Implementation (NSDI)*, volume 6, pages 18–32, 2006.

[160] Lutz Schubert, Keith G Jeffery, and Burkard Neidecker-Lutz. *The Future of Cloud Computing: Opportunities for European Cloud Computing Beyond 2010*. European Commission, 2010.

[161] George AF Seber and Alan J Lee. *Linear Regression Analysis*, volume 936. John Wiley & Sons, 2012.

[162] Mina Sedaghat, Francisco Hernandez, and Erik Elmroth. Unifying Cloud Management: Towards Overall Governance of Business Level Objectives. In *International Conference on Cluster, Cloud and Grid Computing (CCGrid)*, pages 591–597, 2011.

[163] Mina Sedaghat, Francisco Hernandez-Rodriguez, and Erik Elmroth. A Virtual Machine Re-Packing Approach to the Horizontal vs. Vertical Elasticity Trade-off for Cloud Autoscaling. In *Cloud and Autonomic Computing Conference (CAC)*, page 6. ACM, 2013.

[164] Jyothi Sekhar, Getzi Jeba, and S Durga. A Survey on Energy Efficient Server Consolidation through VM Live Migration. *International Journal of Advances in Engineering and Technology*, 5(1):515–525, 2012.

[165] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. CloudScale: Elastic Resource Scaling for Multi-tenant Cloud Systems. In *Symposium on Cloud Computing*, page 5. ACM, 2011.

[166] Seokho Son, Gihun Jung, and Sung Chan Jun. An SLA-Based Cloud Computing that Facilitates Resource Allocation in the Distributed Data Centers of a Cloud Provider. *The Journal of Supercomputing*, pages 1–32, 2013.

[167] Fei Song, Daochao Huang, Huachun Zhou, and Ilsun You. Application-Aware Virtual Machine Placement in Data Centers. In *International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS)*, pages 191–196. IEEE, 2012.

[168] Ying Song, Yuzhong Sun, and Weisong Shi. A Two-Tiered On-Demand Resource Allocation Mechanism for VM-Based Data Centers. *Transactions on Services Computing (TSC)*, 6(1):116–129, 2013.

[169] Simon Spinner, Nikolas Herbst, Samuel Kounev, Xiaoyun Zhu, Lei Lu, Mustafa Uysal, and Rean Griffith. Proactive Memory Scaling of Virtualized Applications. In *International Conference on Cloud Computing (CLOUD)*, pages 277–284. IEEE, 2015.

[170] Simon Spinner, Samuel Kounev, Xiaoyun Zhu, Lei Lu, Mustafa Uysal, Anne Holler, and Rean Griffith. Runtime Vertical Scaling of Virtualized Applications via Online Model Estimation. In *International Conference on Self-Adaptive and Self-Organizing Systems (SASO)*, pages 157–166, 2014.

[171] Christopher Stewart and Kai Shen. Performance Modeling and System Management for Multi-Component Online Services. In *Conference on Symposium on Networked Systems Design and Implementation (NSDI)*, pages 71–84. USENIX Association, 2005.

[172] Giovanni Toffetti, Alessio Gambi, Mauro Pezzé, and Cesare Pautasso. *Engineering Autonomic Controllers for Virtualized Web Applications*. Springer, 2010.

[173] Luis Tomas, Carmen Carrion, Blanca Caminero, and Agustin Caminero. Exponential Smoothing for Network-Aware Meta-Scheduler in Advance in Grids. In *International Conference on Parallel Processing Workshops (ICPPW)*, pages 323–330. IEEE, 2010.

[174] Amos Tversky and Daniel Kahneman. Advances in Prospect Theory: Cumulative Representation of Uncertainty. *Journal of Risk and Uncertainty*, 5(4):297–323, 1992.

[175] Olli Varis. Bayesian Decision Analysis for Environmental and Resource Management. *Environmental Modelling and Software*, 12(2):177–185, 1997.

[176] Nedeljko Vasić, Dejan Novaković, Svetozar Miučin, Dejan Kostić, and Ricardo Bianchini. DejaVu: Accelerating Resource Allocation in Virtualized Environments. *SIGARCH Computer Architecture News*, 40(1):423–436, 2012.

[177] Philippe Vincke. *Multicriteria Decision-Aid*. John Wiley and Sons, 1992.

[178] R Hevner von Alan, Salvatore T March, Jinsoo Park, and Sudha Ram. Design Science in Information Systems Research. *MIS Quarterly*, 28(1):75–105, 2004.

[179] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Cost of Virtual Machine Live Migration in Clouds: A Performance Evaluation. In *Cloud Computing*, pages 254–265. Springer, 2009.

[180] Wenting Wang, Haopeng Chen, and Xi Chen. An Availability-Aware Virtual Machine Placement Approach for Dynamic Scaling of Cloud Applications. In *International Conference on Autonomic and Trusted Computing (UIC/ATC)*, pages 509–516. IEEE, 2012.

[181] Yufeng Wang, Chiu C Tan, and Ningfang Mi. Using Elasticity to Improve Inline Data Deduplication Storage Systems. In *International Conference on Cloud Computing (CLOUD)*, pages 785–792, 2014.

[182] G Weidl, AL Madsen, and S Israelson. Applications of Object-Oriented Bayesian Networks for Condition Monitoring, Root Cause Analysis and Decision Support on Operation of Complex Continuous Processes. *Computers and Chemical Engineering*, 29(9):1996–2009, 2005.

[183] Haishan Wu, Asser N Tantawi, and Tao Yu. A Self-Optimizing Workload Management Solution for Cloud Applications. In *International Conference on Web Services (ICWS)*, pages 483–490. IEEE, 2013.

[184] Linlin Wu, Saurabh Kumar Garg, and Rajkumar Buyya. SLA-Based Resource Allocation for Software as a Service provider (SaaS) in Cloud Computing Environments. In *International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 195–204. IEEE/ACM, 2011.

[185] Linlin Wu, S Kumar Garg, Steve Versteeg, and Rajkumar Buyya. SLA-based Resource Provisioning for Hosted Software as a Service Applications in Cloud Computing Environments. *Transactions on Services Computing*, 2013.

[186] Ming Xia, Marwan Batayneh, Lei Song, Charles U Martel, and Biswanath Mukherjee. SLA-Aware Provisioning for Revenue Maximization in Telecom Mesh Networks. In *Global Telecommunications Conference (GLOBECOM)*, pages 1–5. IEEE, 2008.

[187] Hong Xu, Chen Feng, and Baochun Li. Temperature Aware Workload Management in Geo-Distributed Datacenters. In *SIGMETRICS Performance Evaluation Review*, volume 41, pages 373–374. ACM, 2013.

[188] Hong Xu, Chen Feng, and Baochun Li. Temperature Aware Workload Management in Geo-distributed Datacenters. In *SIGMETRICS Performance Evaluation Review (PER)*, volume 41, pages 373–374. ACM, 2013.

[189] Jing Xu, Ming Zhao, Jose fortes, Robert Carpenter, and Mazin Yousif. On the Use of Fuzzy Modeling in Virtualized Data Center Management. In *International Conference on Autonomic Computing (ICAC)*, pages 25–25. IEEE, 2007.

[190] Stephen Yau and Yin Yin. Qos-Based Service Ranking and Selection for Service-Based Systems. In *International Conference on Services Computing (SCC)*, pages 56–63. IEEE, 2011.

[191] Lenar Yazdanov and Christof Fetzer. Vertical Scaling for Prioritized VMs Provisioning. In *Cloud and Green Computing*, pages 118–125, 2012.

[192] Liangzhao Zeng, Boualem Benatallah, Marlon Dumas, Jayant Kalagnanam, and Quan Z Sheng. Quality Driven Web Services Composition. In *International Conference on World Wide Web (WWW)*, pages 411–421. ACM, 2003.

[193] Yue Zhang, Kwei-Jay Lin, and Jane YJ Hsu. Accountability Monitoring and Reasoning in Service-Oriented Architectures. *Service Oriented Computing and Applications (SOCA)*, 1(1):35–50, 2007.

[194] Xiaoyun Zhu, Mustafa Uysal, Zhikui Wang, Sharad Singhal, Arif Merchant, and Pradeep Padala. What Does Control Theory Bring to Systems Research? *SIGOPS Operating Systems*, 43(1):62–69, 2009.

APPENDIX A

Curriculum Vitae

Soodeh Farokhi

Personal Information

Date of birth	Sep 1985
Place of birth	Abadeh, Iran
Citizenship	Iranian
E-mail	soodeh.farokhi@tuwien.ac.at
Web	`www.ec.tuwien.ac.at/soodeh.farokhi`
Affiliation	Institute of Software Technology and Interactive Systems, Electronic Commerce Group, Faculty of Informatics, Vienna University of Technology
Address	Favoritenstraße 9-11/188, 1040 Wien, Austria

Education

2013 - 2016	PhD in Computer Science, Vienna University of Technology, Austria
2008 - 2011	MSc in Software Engineering, Shahid Beheshti University, Iran
2003 - 2008	BSc in Software Engineering, Shahid Beheshti University, Iran

Employment

2013 - 2016	Research assistant, Vienna University of Technology, Austria
2008 - 2012	Project manager and system analyst, PeykAsa software co., Iran.
2011 - 2012	Co-founder and research assistant, SOEA research-industrial laboratory, Iran
2008 - 2012	Research assistant, ASER research group, Iran
2009 - 2010	Teaching assistant, Shahid Beheshti University, Iran

Scientific activities

Research projects

- **HALEY**: Holistic Energy Efficient Approach for the Management of Hybrid Clouds, 2013 - 2016
- **ARiSE**: Austrian Society for Rigorous Systems Engineering, 2013 - 2016

Scientific talks

- Presenting the research paper, "Coordinating CPU and Memory Elasticity Controllers to Meet Response Time Constraints", at International Conference on Cloud and Autonomic Computing (ICCAC 2015), Cambridge, USA, 24 Sep 2015.
- Presenting the research poster, "Performance-based Vertical Memory Elasticity", at 12th IEEE International Conference on Autonomic Computing (ICAC 2015), Gronoble, France, 9 July 2015.
- Remotely present the research paper, "Self-Adaptation Challenges for Cloud-Based Applications: A Control Theoretic Perspective" , at 10th International Workshop on Feedback Computing (Feedback Computing 2015), Seattle, USA, 13 April 2015.
- Having scientific the talk, "Vertical Elasticity for Cloud-based Applications", at the distributed systems group, Department of Computing Science, Umeå Univerity, Umeå Sweden, 5 March 2015.
- Presenting the research paper, "HS4MC: Hierarchical SLA-based Service Selection for Multi-Cloud Environments", at 4th International Conference on Cloud Computing and Services Science, Multi-Clouds Special Session (CLOSER 2014), Barcelona, Spain, 5 April 2014.

Research visits

- Visiting research Umeå University, as a Short Term Scientific Mission (STSM) granted by IC1304 COST-ACROSS, Umeå Sweden, 27 Feb - 17 March, 2015.

Research events

- Participating at 12th IEEE International Conference on Autonomic Computing (ICAC'15), attending the workshop on Distributed Adaptive Systems (DAS'15), and workshop on Self-Improving System Integration (SISSY'15), 7-10 July 2015, Grenoble, France.
- Participating at 7th Cloud Control Workshop, 9-11 June 2015, Nasslingen, Sweden.
- Participating at 7th International Conference on Utility and Cloud Computing (UCC'14), attending 6th Cloud Control Workshop, Intercloud Architecture and Project tutorial, and Cloud Plugfest workshop, 8-11 Dec 2014, London, UK.

Research collaborations

- Dr. Pooyan Jamshidi, Imperial College London, UK, May 2014 - present
- Prof. Erik Elmroth, Dr. Ewnetu Bayuh, and Dr. Cristian Klein, Umeå University, Sweden, Dec 2014 - present
- Dr. Foued Jrad, and Prof. Achim Streit, Karlsruhe Institute of Technology (KIT), Germany, March-April 2014.

Review service

- (Sub) reviewer for the 12 international conferences and 4 journals in the computer science domain, 2013 - 2016.
  - Journals: The Computer Journal (Oxford), Computing (Springer), Software: Practice and Experience (Wiley), Computer Networks (Elsevier).
  - Conferences: ISCC 2013, Euro-Par 2013, Runtime Verification (RV) 2014, ICSOC 2014 and 2015, UCC 2014, BDC 2014, CCGrid 2014 and 2015, IEEECloud 2015, GECON 2015, WORKS 2015.

Students advising

- Co-advising a master thesis, "Cost-Based Decision Making in Cloud Environments using Bayesian Networks", Vienna University of Technology, Aug 2014.

Publications

- **Soodeh Farokhi**, Pooyan Jamshidi, Ewnetu Bayuh Lakew, Ivona Brandic, and Erik Elmroth. A Hybrid Cloud Controller for Vertical Memory Elasticity: A Control-theoretic Approach. Future Generation Computer Systems (FGCS), The International Journal of Grid Computing and eScience, Elsevier. (*under review*)

- **Soodeh Farokhi***, Ewnetu Bayuh Lakew*, Cristian Klein, Ivona Brandic, Erik Elmroth. Coordinating CPU and Memory Elasticity Controllers to Meet Response Time Constraints. International Conference on Cloud and Autonomic Computing (ICCAC 2015). pp. 69-80, IEEE. Cambridge, MA, USA. 21-24 Sep, 2015. (*contributed equally).

- **Soodeh Farokhi***, Dmytro Grygorenko*, and Ivona Brandic. Cost-Aware VM Placement across Distributed DCs using Bayesian Networks. 12th International Conference on Economics of Grids, Clouds, Systems and Services (GECON 2015). Springer LNCS. Cluj-Napoca, Romania. 15-17 Sep, 2015. (*contributed equally).

- **Soodeh Farokhi**, Pooyan Jamshidi, Drazen Lucanin, and Ivona Brandic. Performance based Vertical Memory Elasticity. 12th International Conference on Autonomic Computing (ICAC 2015). pp. 151-152, IEEE. Gronoble, France. 7-10 July, 2015.
- **Soodeh Farokhi**, Pooyan Jamshidi, Ivona Brandic, and Erik Elmroth. Self-adaptation Challenges for Cloud-based Applications: A Control Theoretic Perspective. 10th International Workshop on Feedback Computing (Feedback Computing 2015). Seattle, USA. 13 April, 2015.
- **Soodeh Farokhi**. Towards an SLA-based Service Allocation in Multi-Cloud Environments. 14th International Symposium on Cluster, Cloud and Grid Computing (CCGrid 2014). pp. 591-594. IEEE/ACM. Chicago, USA. 26-29 May, 2014.
- **Soodeh Farokhi**, Foued Jrad, Ivona Brandic, and Achim Streit. HS4MC: Hierarchical SLA-based Service Selection for Multi-Cloud Environments. 4th International Conference on Cloud Computing and Services Science (CLOSER 2014). pp. 722-734, SciTe Press. Barcelona, Spain. 3-5 April, 2014.
- **Soodeh Farokhi**, Amirreza Ghaffari, Hassan Haghighi, and Fereidoon Shams. MDCHeS: Model-driven Dynamic Composition of Heterogeneous Service. Special Issue of International Journal of Communications, Network and System Sciences (IJCNS), Volume 05, Number 29A, USA, California, Sep 2012.
- **Soodeh Farokhi**, Amir Ghaffari, Ali Nikravesh, and Fereidoon Shams. A Model Driven Framework to Compose Heterogeneous Services. International Conference on Information Science and Applications (ICISA 2011), pp. 552-561, 2011.
- Ali Nikravesh, Fereidoon Shams, **Soodeh Farokhi**, Amir Ghaffari. 2PSIM: Two Phase Service Identifying Method. On the Move to Meaningful Internet Systems Conferences (OTM 2011). Vol. 7045, pp. 625-634, Springer, 2011.

Thesis

- **Doctoral Thesis**. Quality of Service Control Mechanisms in Cloud Computing Environments, Faculty of Informatics, Vienna University of Technology, Austria, Jan 2016.
  Master Thesis: Semi-automated Model Driven Web Service Composition, Electrical and Computer Engineering Department, Shahid Beheshti University, Iran, June 2011.

Technical report

- **Soodeh Farokhi**, A Survey on Web Service Composition Methods. Electrical and Computer Engineering Department, Shahid Beheshti University, Iran, May 2010.

Key Courses

- Fundamental Underpinnings of the Cloud, by Prof. Frank Leymann (PhD course).
- Advanced Internet Computing, by Prof. Schahram Dustdar (PhD course).
- Large-scale Distributed Computing, and Service Level Agreements, by Dr. Ivona Brandic (PhD courses).
- Advanced Software Engineering and Software Architecture (Master courses).
- Software Testing, and Formal Specification-Verification of Software (Master courses).
- Ultra Large Scale Systems and Enterprise Resource Planning (Master courses).

Honors and awards

- Awarded an STSM grant by IC1304 COST-ACROSS for a 3-week visit at Umeå university, Sweden, March 2015.
- Awarded a travel grant by IEEE for attending the IEEE Woman in Engineering International Leadership conference, San Jose, USA, April 2015.
- Admission for a 4-day training entrepreneurship workshop (14-17 Feb), i^2c StartAcademy, Vienna University of Technology, Jan 2015.
- Full PhD scholarship by Vienna University of Technology, Austria, Sep 2013.
- Awarded as the best project manager among 6 candidates at PeykAsa Co., 2011.
- Iran telecommunication research center grant for M.Sc. thesis, 2009.
- Ranked 57^{th} among 9000 participants in the nation-wide MSc entrance exam, Iran, 2008.
- Ranked among top 1% of students participating in nation-wide BSc entrance exam, Iran, 2003.

Memberships

2013 - 2015	DSG (Distributed Systems Group)
2015 - 2016	EC (Electronic Commerce Group)
2013 - 2016	ForSyte (Formal Methods in Systems Engineering)
2013 - Present	WIE (IEEE Women in Engineering)
2013 - Present	ICWG/2302 (IEEE InterCloud Working Group) (as a volunteer)
2010 - Present	IEEE (Institute of Electrical and Electronics Engineers)