FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Governance of Cloud Computing Infrastructures using Knowledge Management

DISSERTATION

zur Erlangung des akademischen Grades

Doktor der technischen Wissenschaften

eingereicht von

DI DI Michael Maurer
Matrikelnummer 0125473

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Univ.Prof. Dr. Schahram Dustdar

Diese Dissertation haben begutachtet:

(Univ.Prof. Dr. Schahram Dustdar)	(Prof. Dr. Rizos Sakellariou)

Wien, 26.04.2012

(DI DI Michael Maurer)

Erklärung zur Verfassung der Arbeit

DI DI Michael Maurer
Guglgasse 6/2/608, 1110 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.

(Ort, Datum) (Unterschrift Verfasser)

Acknowledgements

Official words

The work in this thesis has been funded by the Vienna Science and Technology Fund (WWTF) through project ICT08-018 and by COST-Action IC0804 on Energy Efficiency in Large Scale Distributed Systems.

Motivation

After my second master studies I knew that I was able to learn about and understand the relevant fields in mathematics and computer science. However, only learning about these fields felt no longer enough. I wanted to go to the edge of knowledge and extend it – at least a tiny bit. That is why I went for PhD. This "bit" that on my way I pushed the edge a little further is subsumed in this thesis.

"Methodology"

After such a long journey, there are a lot of people I am thankful for. Among them are my advisor Schahram Dustdar, who gave me the freedom, advice, and encouragement I needed to pursue my thesis, and my project leader Ivona Brandic, who was always there to support me and helped me to develop ideas and put me in contact with many interesting people. One of them was Rizos Sakellariou, my second reviewer, who helped me through his constructive comments to realize several research ideas. He made my stay at the University of Manchester a very pleasant and fruitful research experience. Others were Jean-Marc Pierson, Georges Da Costa, and Damien Borgetto, who I visited at the Institut de Recherche en Informatique de Toulouse (IRIT). During this collaboration I could not only brush up my French and enjoy the French "art de vivre", but also learn how exciting, intriguing and fun research can be.

For the time I spent in Vienna I want to thank my colleagues at the Distributed Systems Group, and foremost those working with me on my project: Vincent Chimaobi Emeakaroha, who with me was the first to start on this project; Ivan Brešković, who made this group much more interactive and lively; and Toni Mastelić and Dražen Lučanin, who brought new ideas and a fresh breeze to the group.

Yet a different thank goes to my Scout group and the kids in my group, who give me a different perspective on life apart from science and work, and help to balance it out.

Last but not least I am thankful for the steady support of my family and friends, all foremost my mother Elisabeth Riegler, who has always encouraged me to pursue my interests, but also

my sister Valerie Riegler and my brother Christian Löw for the endless chats we had and gaming nights we spent together.

Michael Maurer

Abstract

Cloud computing has gained a lot of momentum in recent years. Its vision is to offer computing power as a utility implying sheerly unlimited and instantly provisioned resources for customers. However, there are a lot of obstacles towards these goals. This thesis tackles two of them related to Service Level Agreements (SLAs): adaptive SLA mapping, and resource- and energy-efficient SLA enactment.

A Cloud provider who wants to offer computing resources signs an SLA with the customer. In this SLA the provider states the Quality of Service (QoS) she will guarantee, the price the customer will have to pay, and the penalty the provider will have to pay in case she breaches the QoS guarantees. The customer has his own SLA with the QoS guarantees he wants, and the prices he is willing to pay. Matching these bids and asks is especially hard as non-standardized varying definitions of computing resources in electronic markets cause a large variety of different SLAs. Moreover, these SLAs are typically bound to internal business processes and, therefore, cannot be altered easily.

In this thesis we use SLA templates and SLA mappings that allow providers and customers to map parameters of their SLAs to each other without changing the original ones. From these mappings we learn user preferences and are able to generate and adapt public SLA templates that reflect the users' needs and help to standardize SLAs. We present a cost-benefit analysis of this approach and evaluate various learning and adaptation strategies.

Furthermore, after bids and asks have been matched and the SLA has been signed, the Cloud provider has to keep up to her promises to avoid SLA penalties – despite all the dynamism of workload changes. On the other hand, also under-utilization of resources and high energy wastage are big cost factors in large-scale distributed systems. Consequently, a Cloud provider aims at minimizing SLA violations, maximizing resource utilization, and minimizing energy wastage. However, this is not straightforward as Cloud computing infrastructures consist of many differently configurable elements as applications, virtual machines (VMs), physical machines (PMs), and also other Cloud providers, to which applications can be outsourced. This leads to a plethora of possible reconfiguration and reallocation actions of these elements and the resources they are assigned to. Many of the resulting problems are typically NP-hard.

This thesis uses autonomic computing and knowledge management to govern cloud computing infrastructures. We find and structure possible reactive and proactive actions that prevent SLA violations, increase resource utilization and lower energy usage. First, we focus on VM resource (re-)configuration and investigate several knowledge management (KM) techniques such as case based reasoning, default logic, situation calculus, or a rule-based approach. We design and implement a KM-technique agnostic simulation engine to evaluate the suitability of these

approaches. The rule-based approach is found most profitable in terms of the quality of the recommended actions, as well as its scalability. However, parameters were identified, on which the performance of the rule-based approach largely depends. Therefore, a self-adapting rule-based approach is introduced that autonomically adapts to changing workload volatility. Furthermore, we tackle VM migrations and PM power management. We introduce migration models for VMs and power management models for PMs, show that this management problem is an instance of the NP-hard binary integer programming problem, and apply and evaluate several heuristics reducing energy consumption. Doing this, it also proven that the VM (re-)configurations do not only increase resource, but also energy efficiency. Finally, we show a possible extension of the KM approach for Cloud federations.

Kurzfassung

Cloud Computing hat in den letzten Jahren reges Interesse erfahren. Die Vision von Cloud Computing, Kunden Rechenleistung als Dienstleistung wie Wasser, Strom oder Gas anzubieten, impliziert das Vorhandensein von schier unlimitierten und augenblicklich verfügbaren Rechenresourcen. Jedoch gibt es noch viele Hindernisse auf dem Weg dieses Ziel zu erreichen. Diese Dissertation nimmt zwei davon in Angriff, die im Bezug zu *Service Level Agreements* (deutsch etwa Dienstgütevereinbarung, Abk. SLA) stehen: adaptive *SLA Mappings* (deutsch etwa SLA Zuordnungen) und resource- und energieeffizientes *SLA enactement* (deutsch etwa das Einhalten von SLAs).

Ein Anbieter von Cloud Computing, der seine Rechenresourcen anbieten möchte, unterzeichnet einen Vertrag, ein sogenanntes SLA, mit seinem Kunden. In diesem SLA spezifiziert der Anbieter die Gütekriterien (englisch *Quality of Service*, Abk. QoS), die er einhalten möchte, den Preis, den der Kunde bezahlen muss, sowie die Strafe, die der Anbieter bezahlen muss, falls er die QoS-Garantien bricht. Der Kunde hat sein eigenes SLA mit QoS-Garantien, die er sucht, und mit den Preisen, die er zu zahlen bereit ist. Das Zusammenbringen von Angebot und Nachfrage ist speziell deshalb schwierig, weil nicht standardisierte, sich häufig ändernde Definitionen von Rechenresourcen in elektronischen Märkten eine große Vielfalt an verschiedenen SLAs verursachen. Des Weiteren sind diese SLAs meistens in internen Geschäftsprozessen verankert und können daher nicht leicht verändert werden.

Wir verwenden in dieser Dissertation SLA-Vorlagen (englisch *SLA templates*) und SLA Mappings, die es Anbietern und Kunden erlauben sich Parameter ihrer SLAs gegenseitig zuzuordnen ohne die originalen SLAs verändern zu müssen. Von diesen Mappings lernen wir die Präferenzen der Anwender. Dadurch sind wir in der Lage öffentliche SLA-Vorlagen zu generieren und zu adaptieren, die die Anwenderpräferenzen widerspiegeln und die uns helfen SLAs zu standardisieren. Wir präsentieren eine Kosten-Nutzenrechnung von diesem Ansatz und evaluieren verschiedene Lern- und Adaptionsstrategien.

Nachdem sich Angebot und Nachfrage gefunden haben und das SLA unterzeichnet worden ist, muss der Cloud Computing-Anbieter seine Versprechen trotz dynamischer Auslastungsveränderungen einhalten um SLA-Strafzahlungen zu vermeiden. Andererseits sind auch Resourceunterauslastung und die dadurch entstehende Energieverschwendung große Kostenfaktoren in großen verteilen Systemen (englisch *large-scale distributed systems*). Daher haben Cloud Computing-Anbieter das Ziel SLA-Verletzungen zu minimieren, die Resourceauslatung zu maximieren und Energieverschwendung zu minimieren. Allerdings ist dies nicht einfach, da Cloud Computing-Infrastrukturen aus vielen verschieden konfigurierbaren Elementen bestehen. Diese Elemente sind Applikationen, virtuelle Maschinen (VM), physische Maschinen (Server, Abk.

PM) und andere Cloud-Anbieter, zu denen man Applikationen outsourcen kann. Dies führt zu einer unüberschaubaren Anzahl an möglichen Rekonfigurations- und Reallokationssaktionen von diesen Elementen und den Resourcen, denen sie zugeordnet sind. Viele der entstehenden Probleme sind typischerweise NP-schwer.

Diese Dissertation benutzt *Autonomic Computing* und Wissensmanagement (englisch *knowledge management*, Abk. KM) um Cloud Computing-Infrastrukturen zu verwalten und zu steuern. Wir finden und strukturieren mögliche reaktive und proaktive Aktionen, die SLA-Verletzungen vermeiden, die Resourcenauslastung erhöhen und den Energieverbrauch reduzieren. Zuerst konzentrieren wir uns auf die (Re-)konfiguration von VM-Resourcen und betrachten einige Wissensmanagementmethoden wie *Case Based Reasoning* (deutsch Fallbasiertes Schließen), *Default Logic*, Situationskalkül (englisch *situation calculus*) und einen regelbasierten Ansatz. Wir entwerfen und implementieren eine KM-agnostische Simulationsumgebung, mit der wir die Eignung dieser Ansätze evaluieren. Der regelbasierte Ansatz schneidet am besten ab, sowohl was die Qualität der empfohlenen Aktionen, als auch seine Skalierbarkeit (englisch *scalability*) betrifft. Allerdings wurden auch Parameter identifiziert, die die Leistung des regelbasierten Ansatzes stark beeinflussen. Deswegen wurde ein selbstadaptiver regelbasierter Ansatz entwickelt, der sich schwankender Auslastungsvolatilität (englisch *workload volatility*) autonom anpasst. Weiters beschreiben wir Lösungen und Modelle für das Migrieren von VMs und das PM-Power-Management. Wir zeigen, dass dieses Managementproblem eine Instanz des NP-schweren *binary integer programming*-Problems ist und evaluieren mehrere Heuristiken, die den Energieverbrauch reduzieren. Dabei zeigen wir auch, dass die (Re-)konfigurationen der VMs nicht nur die Resource-, sondern auch die Energieeffizienz steigern. Schließlich präsentieren wir eine mögliche Erweiterung der KM-Lösung für Cloud-Föderationen (englisch *Cloud federations*).

Contents

Earlier Publications

Most work in this thesis has been published at conferences, in journals or as book chapters. These core papers build the foundation of this thesis. They are listed here, but will generally not be explicitly referenced again. Parts of these papers are contained in verbatim. Please refer to Appendix A for a full publication list of the author of this thesis.

Refereed Publications in Conference Proceedings

1. Michael Maurer, Ivona Brandic, Rizos Sakellariou. Enacting SLAs in Clouds Using Rules. Euro-Par 2011, Bordeaux, France, August 29 - September 2, 2011. [151]

2. Michael Maurer, Ivona Brandic and Rizos Sakellariou. Simulating Autonomic SLA Enactment in Clouds using Case Based Reasoning. ServiceWave 2010, Ghent, Belgium, December 13-15 2010. [150]

3. Michael Maurer, Vincent C. Emeakaroha, Ivona Brandic, Jörn Altmann. Cost and Benefit of the SLA Mapping Approach for Defining Standardized Goods in Cloud Computing Markets. International Conference on Utility and Cloud Computing (UCC 2010) in conjunction with the International Conference on Advanced Computing (ICoAC 2010), December 14-16, 2010, Chennai, India. [155]

4. Michael Maurer, Ivan Breskovic, Vincent C. Emeakaroha, Ivona Brandic. Revealing the MAPE Loop for the Autonomic Management of Cloud Infrastructures. Workshop on Management of Cloud Systems (MoCS 2011), in association with the IEEE Symposium on Computers and Communications (ISCC 2011), 28 June 2011, Kerkyra (Corfu) Greece. [148]

5. Vincent Chimaobi Emeakaroha*, Pawel Labaj*, Michael Maurer*, Ivona Brandic and David P. Kreil. Optimizing Bioinformatics Workflows for Data Analysis Using Cloud Management Techniques. The 6th Workshop on Workflows in Support of Large-Scale Science (WORKS11), in conjunction with Supercomputing 2011, Seattle, November 12-18, 2011. (* contributed equally) [80]

6. Michael Maurer, Ivona Brandic, Vincent C. Emeakaroha, Schahram Dustdar. Towards Knowledge Management in Self-adaptable Clouds. IEEE 2010 Fourth International Workshop of Software Engineering for Adaptive Service-Oriented Systems (SEASS '10), in conjunction with ICWS 2010 and SCC 2010, Miami, Florida, USA, July 5-10, 2010. [149]

7. Gabor Kecskemeti, Michael Maurer, Ivona Brandic, Attila Kertesz, Zsolt Nemeth and Schahram Dustdar. Facilitating self-adaptable Inter-Cloud management. 20th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing PDP 2012. Munich, Germany, 15-17 February, 2012. [106]

8. Damien Borgetto*, Michael Maurer*, Georges Da Costa, Jean-Marc Pierson, and Ivona Brandic. Energy-efficient and SLA-aware managament of iaas clouds. In Third international conference on future energy systems (e-Energy 2012), Madrid, Spain, May 2012. (accepted). (* contributed equally) [49]

9. Michael Maurer, Ivona Brandic, and Rizos Sakellariou. Self-adaptive and resource efficient SLA enactment for cloud computing infrastructures. In 5th International Conference on Cloud Computing (IEEE Cloud 2012) (submitted), Honolulu, HI, USA, June 2012. [153]

Refereed Publications in Journals

1. Michael Maurer, Vincent C. Emeakaroha, Ivona Brandic, Joern Altmann. Cost-Benefit Analysis of an SLA Mapping Approach for Defining Standardized Cloud Computing Goods. Future Generation Computer Systems, 2011, doi:10.1016/j.future.2011.05.023. [156]

2. Michael Maurer, Ivona Brandic, and Rizos Sakellariou. Enacting SLAs in Clouds using Knowledge Management. Future Generation Computer Systems (submitted), 2012. [152]

Book Chapters

1. Michael Maurer, Vincent C. Emeakaroha, and Ivona Brandic. Economic analysis of the SLA mapping approach for cloud computing goods. In Achieving Federated and Self-Manageable Cloud Infrastructures: Theory and Practice. IGI Global, 2012. [154]

CHAPTER 1

Introduction

Cloud computing is an emerging IT paradigm for large-scale distributed systems. Its vision is to provide computing power as a utility, like gas, electricity or water [62]. According to the U.S. National Institute of Standards and Technology (NIST), Cloud computing is "a model for enabling 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 service provider interaction" [159]. This implies that computing power should be available to users at any time and at the right amount they desire. It also implies that Cloud computing providers need to guarantee specific non-functional requirements to the user, as response time, throughput, or storage. These Quality of Service (QoS) goals are subsumed and specified in so-called *Service Level Agreements* (SLAs), which also contain penalties the provider has to pay in case of violations of these guarantees.

Providing computing power as a utility can be achieved by offering differing entities as a service, such as software, platforms, or infrastructure. These different delivery models of Cloud computing are therefore called *software as a service* (SaaS), *platform as a service* (PaaS), and *infrastructure as a service* (IaaS). Prominent Cloud providers are Google Mail [19], Google Docs [18], or salesforce.com [10] for SaaS; Google App Engine [7], Windows Azure [9] for Paas; and Amazon EC2 [3] for IaaS.

Cloud computing can also be classified into several deployment models, which are public, private, hybrid and community Clouds. According to [159] *public Clouds* are "provisioned for open use by the general public", which may be enterprises, academic or governmental institutions, or individuals. Data is stored in the datacenter of Cloud providers. *Private Clouds* are run inside organizations. This deployment model is especially attractive for institutions which already have computing infrastructures, but do not want to store their data in data centers, which they cannot fully control. *Community Clouds* are "provisioned for exclusive use by a specific community of consumers from organizations that have shared concerns". Finally, *hybrid Clouds* represent a combination of at least two distinct Cloud infrastructures that "remain unique entities, but are bound together", and can exchange data and applications among them. *Cloud*

federations [61] are a typical scenario for hybrid Clouds.

Cloud computing relies upon several state-of-the-art technologies. Among them are virtualization [39], monitoring [37, 158], scheduling [132, 87], work on clusters and grids [59, 42], theory of electronic markets [29], and optimization theory [129]. Despite the existence of these technologies Armbrust et al. [35] list in their famous technical report "Above the Clouds: A Berkeley View of Cloud Computing" top 10 obstacles hindering the adoption of Cloud computing making it the state-of-the art computing paradigm. These obstacles include the availability of the service, data lock-in, data confidentiality and auditability, data transfer bottlenecks, performance unpredictability, scalable storage, bugs in large distributed systems, and quick scaling.

In this thesis we will tackle quick scaling, and performance unpredictability by enacting SLAs. Our work on SLA generation and adaptive SLA mapping also helps to mitigate data lock-in, as this technique helps customers to change to Cloud providers using different SLAs. Furthermore, we want to add two problems that have not been dealt with in [35]: resource and energy efficiency. This is especially crucial, because ICT currently makes up for 2% of the worldwide CO_2 production [2,83]. Even in the state-of-the-art data centers, the massive amount of physical machines, i.e., servers, leads to high power consumption and carbon footprint of the data center, as well as high operation costs [103].

1.1 Problem Statement

Before a user can benefit from using a service offered by a Cloud provider, an SLA has to be signed. An SLA comprises elements such as names of trading parties, names of SLA attributes, measurement metrics, and attribute values [184]. Despite the existence of SLAs, buyers and sellers of computing resources face the problem of varying definitions of computing resources in Cloud computing markets. Computing resources are described through different non-standardized attributes, e.g., CPU cores, execution time, inbound bandwidth, outbound bandwidth, and processor type [182]. Sellers use them to describe their supply of resources. Buyers use them to describe their demand for resources. As a consequence, a large variety of different SLAs emerges. Even though buyers and sellers might have similar needs, a different description of these needs impedes successful matchmaking between offers from sellers and requirements from buyers. The probability to find such a match becomes even lower the more different resource types there are [184], and thus due to the non-standardization of Cloud computing goods successful matchmaking becomes very unlikely. The probability of matching offers and requirements is called *market liquidity*. Both Cloud computing providers and users have a high incentive to increase this liquidity.

Eventually, after the SLA has been signed, the Cloud provider commits herself to enact it. For the underlying infrastructure this means that it has to react to dynamic load changes, ranging from peak performance to utilization gaps. This brings up two issues: on the one hand, the management of a Cloud computing infrastructure has to guarantee the pre-established SLAs despite all the dynamism of workload changes. The QoS goals contained in the SLAs are called *Service Level Objectives* (SLOs). Informally, they can be stated as "storage should be at least 1000 GB", "bandwidth should be at least 10 Mbit/s" or "response time should be less than 2 s". On the other hand, a Cloud provider aims at efficiently utilizing resources and reducing resource

wastage. Only allocating what is really needed is a crucial question to reducing wasted energy consumption. As stated above, energy consumption of a data center is a non negligible part of expenses for data centers [103]. Thus, Cloud providers aim at concurrently achieving the following conflicting goals: minimizing SLA violations, maximizing resource utilization, and minimizing energy consumption. However, this is not straightforward as Cloud computing infrastructures consist of many differently configurable elements such as applications, virtual machines (VMs), physical machines (PMs), and also other Cloud providers, to which applications can be outsourced. This leads to a plethora of possible reconfiguration and reallocation actions of these elements and the resources they are assigned to. Many of the resulting problems are typically NP-hard, and scalable heuristics have to be found that solve these problems in an acceptably small amount of time. In more detail, dynamic resource allocation and energy consumption has to be seen in the context of (i) the configuration of virtual machines, (ii) the deployment of applications on virtual machines, (iii) the deployment of virtual machines on physical machines, (iv) the power state (on/off/sleep) of physical machines, and (v) the possibility to outsource computation to other Cloud providers. The interplay of these different levels is a crucial aspect of this research problem.

1.2 Methodology

This work is embedded in the *Foundations of Self-governing ICT infrastructures* (FoSII) project [17]. The FoSII project aims at developing an infrastructure for autonomic SLA management and enforcement. For managing SLAs and bridging the gap between differently defined SLAs, we apply the method of *SLA mappings* in this thesis. SLA mapping has been first proposed and implemented in [51]. It allows to define mappings, i.e., translations from one SLA document to another one that uses different SLA parameter names. However, issuing these SLA mappings is quite costly. Thus, we apply several learning strategies to generate public SLA templates that reflect the users' needs and help to reduce costs of generating SLA mappings.

Besides the already implemented monitoring framework [79] that takes care of monitoring the state of the Cloud infrastructure and its applications, the knowledge management (KM) system presented in this thesis represents another essential building block of the FoSII infrastructure. [52] proposes an approach to manage Cloud infrastructures by means of Autonomic Computing, which in a control loop monitors (M) Cloud parameters, analyzes (A) them, plans (P) actions and executes (E) them; the full cycle is known as MAPE [102]. According to [100] a MAPE-K loop stores knowledge (K) required for decision-making in a knowledge base (KB) that is accessed by the individual phases. We investigate several KM techniques and apply different heuristics to solve the resource allocation and resource management problems for energy-efficient and resource-efficient SLA enactment.

1.3 Research Questions

The discussion in the previous sections raise the following five research questions.

Research question 1:
How can one define standardized Cloud computing SLAs?

Cloud providers and customers meet in Cloud markets, which are electronic markets that trade Cloud computing goods. Since each of them uses their own definition of Cloud computing resources, e.g., "CPU cores", "Cores of one CPU", or "Computing power", there exist no standardized Cloud computing SLAs. This prevents customers from finding relevant providers, and vice versa, and aids vendor or data lock-in.

Research question 2:
What are the possibilities and different levels of allocating resources and reactive actions to prevent SLA violations in Cloud computing infrastructures?

There exist many configurable, tunable and manageable entities in Cloud computing infrastructures, which are heterogeneous applications, VMs, PMs, and other Cloud providers. The first task is to identify means of adjusting these elements. The second task consists of structuring the found adaptation actions into subproblems that form a consistent model of Cloud computing infrastructures. Ideally, using a "divide and conquer" strategy, one should be able to solve these subproblems sequentially without a high inter-dependability of found solutions.

Research question 3:
How can one autonomically and efficiently (in quality, energy and time) allocate and reallocate resources for VMs and PMs in order to proactively prevent SLA violations? Consequently, how can one increase energy efficiency in Cloud computing infrastructures?

The adoption of Cloud computing as a state-of-the-art computing paradigm hinges upon perceiving it as a reliable means of computing. This and fluctuating customers and workloads require high-quality resource allocation and re-allocation solutions in a short amount of time for a large and very large number of applications, VMs and PMs. As expenses on energy form a large part of expenses of Cloud computing providers, an energy efficient implementation of the SLA enactment is crucial for a future generation technology.

Research question 4:
What knowledge management technique, i.e., a technique of how stored information should be used, is most suitable to be used in an autonomic control loop governing Cloud computing infrastructures? How does it interact with the other phases of the autonomic control loop?

When extending the autonomic loop with knowledge management, it is important to determine which technique is most appropriate to govern Cloud computing infrastructures. Related to research question 3, a proposed technique should of course be highly scalable. It is judged by the quality of the decisions it recommends. Furthermore, it has to be determined in what part of the MAPE cycle the knowledge management technique should be employed, and when to interact with it.

Research question 5:
How can the found approach be extended for the use of Cloud federations and hybrid Clouds?

Knowledge management will not only be of concern within a Cloud computing infrastructure, but also among several collaborating Clouds. It should be analyzed, in which parts of a Cloud federation knowledge management should be deployed, and how it could be utilized to achieve similar goals for the whole federation as well as for individual Clouds.

1.4 Scientific Contributions

Following the research questions posed in Section 1.3, the following contributions to the state of the art are shown in this thesis.

Contribution 1:
An adaptive SLA mapping approach has been introduced to bridge the gap between differently defined SLAs, which, however, describe the same or similar Cloud computing goods.

A cost and benefit analysis of different methods for standardizing SLAs has been carried out. The standardization is based on previous SLA mappings carried out by consumers on a Cloud computing market. This contribution has been originally published in [155,156] and is presented in Chapter 3.

Contribution 2:
Possible reallocation actions for applications running on a Cloud computing infrastructure have been determined. They have been structured into so called escalation levels*, into which they have been placed according to their locality and complexity.*

The hierarchical model allows for a sequential solution of the allocations and reallocation problems. However, some of the resulting subproblems are still NP-hard (cf. Contribution 3). This contribution has been originally published in [151] and is presented in Section 2.3.

Contribution 3:
A self-adaptive rule-based KM approach for VM reconfiguration, VM-PM allocation and reallocation, and PM power management has been designed, implemented, and evaluated.

The approach prevents almost all SLA violations, increases the utilization of all resources, and attains both goals by a low number of VM reallocation actions. The approach is able to self-adapt its most important parameters based on the workload of an application and on utility, and is highly scalable. Using several heuristics for VM-PM allocation and PM power management, and introducing energy models for VM migration, and PM power management, it is shown via simulations that this approach can heavily reduce consumed energy by Cloud computing infrastructures. This contribution has been originally published in [151,49,153] and is presented in Sections 4.5, Chapter 5 and Sections 6.6, 6.7.

Contribution 4:
Several possible KM techniques to be used for Cloud computing infrastructures have been studied. Two candidates have been designed, implemented and evaluated with the help of a KM technique agnostic simulation engine developed and designed for this purpose.

In more detail, a preliminary analysis of the following KM techniques has been conducted: rules, default logic, situation calculus, and case based reasoning. From this analysis, we have found case based reasoning and a rule-based approach to be most promising. We designed, implemented and evaluated both approaches. Especially the rule-based approach achieves low SLA violations rates, high resource utilization, and achieves this by few time- and energy-consuming reallocation actions. This contribution has been originally published in [149, 150, 151] and is presented in Sections 4.1, 4.3, 4.4, and 6.3, 6.4, 6.5.

Contribution 5:
An extension of the rule-base approach for Cloud federations has been presented. It has been shown how rules can be formulated to govern Cloud federations for meeting SLAs resource-efficiently.

Three architectures have been proposed and analyzed for placing KM systems. A rule-based system has been chosen as KM technique based on experience in Contributions 3-4. Additional elements of the Cloud federation architecture have been formalized, and the feasibility of this approach has been shown by pointing out possible sample rules. This contribution has been originally published in [106] and is presented in Chapter 7.

1.5 Organization of the Thesis

The rest of this thesis is structured as follows.

- Chapter 2 gives necessary background information about the FoSII project, the adaptive SLA mapping approach, autonomic computing and the different phases of its control loop. Furthermore, it enumerates possible adaptation actions and structures them into so-called *escalation levels*. This chapter is mostly based on work from [148].
- Chapter 3 presents the methodology of the adaptive SLA mapping approach and its evaluation. This chapter is based on [156].
- Chapter 4 deals with resource-efficient SLA enactment and knowledge management. It compares different methods of knowledge management (based on [149]), and presents the design and implementation of an approach using case based reasoning (based on [150]) and rules (based on [151]). Finally, it exposes an approach that self-adapts the rule-based one (based on [153]).
- Chapter 5 tackles the energy efficiency aspect of the SLA enactment approaches. Section 5.4 is based on [49].
- Chapter 6 presents evaluation results for Chapters 4 and 5, and describes the developed KM-technique agnostic simulation engine and the workload generation mechanisms. This chapter is based on the evaluation results of [150, 151, 80, 153].
- Chapter 7 extends the knowledge management approach for the use of Cloud federations. This chapter is based on [106].

- Chapter 8 describes the state of the art and presents the enhancements this thesis has made to it.
- Chapter 9 concludes this thesis, talks about its limitations, gives a critical reflection and an outlook into possible future work.

CHAPTER 2

Conceptual Model of Adaptive SLA Mapping and Autonomic SLA Enactment

This chapter highlights the foundations of this thesis. We explain autonomic computing, the project architecture, and enumerate and structure reactive and proactive actions for SLA enactment.

2.1 Outline of the FoSII Project Architecture

Cloud computing represents a novel paradigm for on-demand provisioning of ICT infrastructures, services, and applications. Thereby, resources are provisioned in predefined quality considering various functional and non-functional guarantees. Key concepts distinguishing Cloud Computing from other paradigms for the realization of large-scale distributed systems include (i) unlimited scalability of resources, (ii) sophisticated Service Level Agreement (SLA) management and generation, giving the customer guarantees on various non-functional aspects, and (iii) extensive use of virtualization technologies [62]. Many of the key concepts cope with contradicting goals, as, for example, unlimited scalability vs. energy efficiency. Scalability, i.e., providing the desired amount of resources at the right time, usually causes wastage of energy due to idle states or standby modes of devices and infrastructures; they still consume energy, although being unused. Autonomic Computing seems to be one of the promising solutions for the management of Cloud infrastructures by optimizing various (and maybe contradicting) goals as, for example, efficient resource usage, SLA management, virtualization and at the same time minimizing human interaction with the system and energy consumption.

Autonomic systems require high-level guidance from humans, but autonomically decide which steps need to be done to keep the system stable [109]. Such systems constantly adapt themselves to changing environmental conditions. Similar to biological systems, e.g., human

body, autonomic systems maintain their state and adjust operations considering changing components, workload, external conditions, hardware, and software failures. Autonomic computing has served as a promising concept for the infrastructure management in various areas, e.g., services, Grids, and SLA management [100, 174]. The autonomic control loop is known as MAPE [102], where (M) stands for monitoring the managed elements, (A) for their analysis, (P) for planning actions, and (E) for their execution. The MAPE-K loop stores knowledge (K) required for decision-making in a knowledge base (KB).

However, existing autonomic frameworks, e.g., for Grids or SLA management, cannot easily be applied to Cloud computing infrastructures due to various reasons. For example, due to the virtualization layer, monitoring tools usually have to be configured on demand, distinguishing application based monitoring and resource based monitoring [81]. Energy efficiency requires novel techniques for the management of resources [150], while SLA generation requires advanced concepts for the management of the heterogeneous user base [155]. Thus, the traditional MAPE loop has to be revealed and tailored to Cloud specific solutions.

In the *Foundations of Self-Governing ICT Infrastructures* (FoSII) project, we develop novel techniques and methods for self-governing ICT infrastructures, and consequently apply the developed infrastructures for self-managed Clouds [17]. One of the core research issues of the FoSII project is the development of an appropriate autonomic loop suitable for the self-management of Clouds. Thus, this thesis proposes an extended MAPE-K loop, called A-MAPE-K, including an *Adaptation* phase added to the traditional MAPE-K phases. The adaptation phase is necessary as a balance to the virtualization layer. During the *Adaptation* (A) phase, Cloud infrastructures, as well as applications to be deployed on the Clouds, are tailored and adapted. Moreover, we present novel concepts for the implementation of the *Monitoring* and *Knowledge Management* phases considering virtualization overhead.

2.2 Autonomic Loop and Cloud Computing

This section explains the foundations of autonomic computing, and discusses a motivating scenario for the development of the A-MAPE-K loop. Furthermore, it presents the SLA lifecycle, which should be supported by the autonomic loop, and finally it discusses the conceptional design of the A-MAPE-K loop.

Autonomic Computing

The vision of Autonomic Computing was described in [111]. It presents the idea of a managed element that is controlled by an autonomic manager in a MAPE loop. Furthermore, the authors describe properties of self-management using autonomic computing, which are referred to as *self-* properties*. These properties are:

self-configuration The configuration of managed elements follows high-level polices. Installing, configuring and integrating systems should no longer be done manually.

Table 2.1: Sample SLA parameter objectives.

SLA Parameter	Value
Incoming Bandwidth (IB)	$>$ 10 Mbit/s
Outgoing Bandwidth (OB)	$>$ 12 Mbit/s
Storage (St)	$>$ 1024 GB
Availability (Av)	$\geq$ 99%
Clock speed (Cs)	$\geq$ 1000 MHz

self-optimization The autonomic manager tries to ameliorate the managed elements constantly. The autonomic manager tunes parameters automatically and adapts the managed elements to current circumstances.

self-healing The autonomic manager automatically detects, diagnoses, and repairs problems of their managed elements.

self-protection The autonomic manager defends against attacks, e.g., DDoS attacks, automatically.

In [52] Brandic proposes to use principles of autonomic computing to manage Cloud services. We will discuss an extension of this proposal to manage Cloud computing infrastructures in the following.

Motivating Scenario

Table 2.1 depicts an SLA used to exemplify A-MAPE-K phases. We assume an IaaS scenario, where SLAs specify guaranteed resources suitable for the application execution in a VM. The column *SLA Parameter* defines typical Service Level Objectives (SLOs) including *incoming bandwidth (IB)*, *outgoing bandwidth (OB)*, *storage (St)*, *availability (Av)* and *clock speed (Cs)*. The column *Value* specifies a concrete value with the according relational operator. SLAs are generated between the Cloud provider and user before the deployment of the application. The following section will discuss the lifecycle necessary for the establishment and management of SLAs between the user and the provider.

SLA Lifecycle

We assume a typical Cloud use case where potential Cloud users deploy applications in an IaaS manner, as explained next. The service provider registers resources (i.e., VMs) to particular databases containing public SLA templates. Thereafter, Cloud users can look up Cloud services that they want to use for the deployment of applications. Similar to the provider, the Cloud user also has an SLA template utilized for his private business processes. We assume that the private SLA template cannot be changed, since it could also be part of some other local business processes and has usually to comply with different legal and security guidelines. If matching

SLA templates are found, the SLA contract can be negotiated and established and the application can be deployed and executed.

Once the applications are deployed, the execution should be done in an autonomic way, minimizing user interactions with the system, optimizing energy consumption, and preventing violations of established SLAs. Resource management requires adequate monitoring techniques, which are used for application based SLA monitoring and deciding whether an SLA is violated or not. This is, however, far from trivial. Furthermore, in order to prevent SLA violations, knowledge management techniques are necessary. They are used to determine if applications can be migrated and virtual machines (VMs) and physical machines (PMs) (re-)configured, migrated or switched off/on on demand in order to prevent SLA violations.

A-MAPE-K Loop Design

This section presents how the aforementioned SLA lifecycle can be realized using the autonomic loop. We distinguish between system set up time and application runtime. During system set up time the applications and the infrastructure are tailored and *adapted*. Once the applications are deployed, we consider *monitoring*, *knowledge management* and *execution* phases during the application runtime. In this section, in particular, we focus on the adaptation, monitoring, and knowledge management phases, as shown in Figure 2.1.

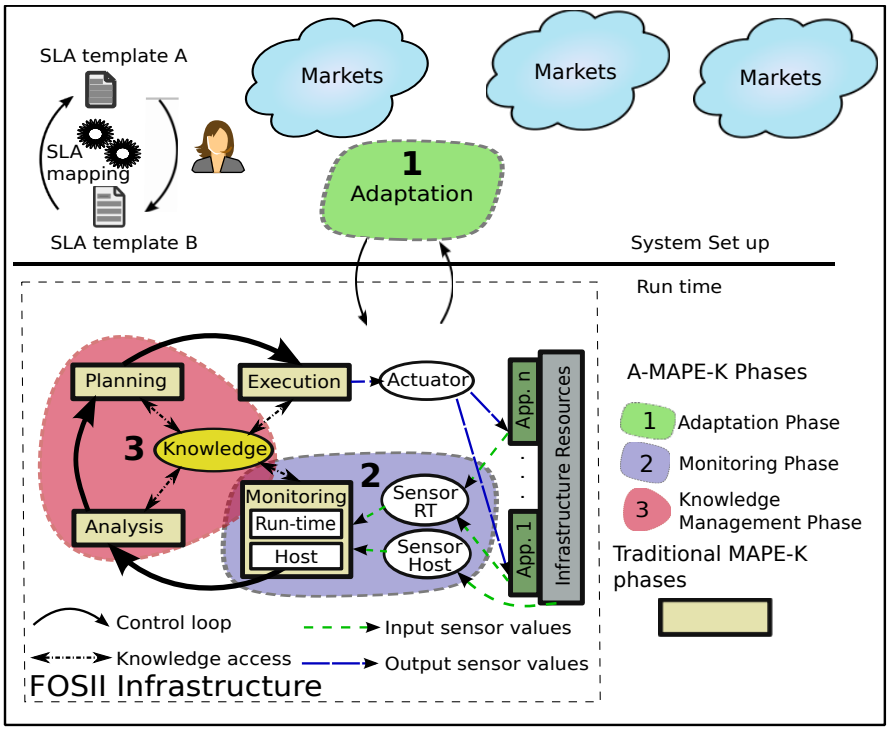


Figure 2.1: FoSII Infrastructure Overview

Adaptation As shown in Figure 2.1, part 1, the *adaptation* phase comprises all necessary steps to be done before successful deployment and start of the application. This includes SLA contract establishment and tailoring of the monitoring systems for the particular application. During this phase it has to be ensured that private templates of the provider and consumers match publicly available templates. However, public and private templates may differ. A typical mismatch between templates would be between different **measurement units** of attributes, as, for example, for the SLO clock speed (see sample SLA parameter objectives, Table 2.1), or **missing attributes**. Therefore, a mechanism is required for the automatic adaptation between different templates. Adaptation can include handling of missing SLA parameters, inconsistencies between attributes and translation between different attributes. More complex adaptations would include automatic service aggregation, including third party services, if, for example, the *clock speed* attribute is completely missing in the public template, but required in the private template. A third party provider (e.g., a computer hardware reseller) could be integrated to deliver information about the *clock speed* attribute. Possible machine-readable formulations of SLAs (expressed in XML) are the WSLA [108] and the WS-Agreement [32] format.

Monitoring Clouds face the problem of SLA parameters required by an application usually differing from the parameters measured by the monitoring tools. A typical application based SLA parameter is *system availability*, as depicted in Table 2.1. Current monitoring systems (e.g., ganglia [146]) facilitate monitoring only of low-level system resources, such as *system up time* and *down time*. Thus, availability has to be calculated based on those low-level metrics. To achieve that, the monitoring phase should comprise two core components, namely a *host monitor* and a *run-time monitor* (see Figure 2.1, part 2). The former is responsible for monitoring low-level resource metrics, e.g., system up time and down time directly delivered by the measurement tools (e.g., ganglia), whereas the latter is responsible for metric mapping, e.g., mapping of *system up time* and *down time* to *system availability* and consequently for the monitoring of SLA agreements. Another example for VM parameters retrieved by direct measurements would be `free_disk` or `packets_sent` in comparison to the SLA parameters that we are more interested in: *storage* and *bandwidth*. This is achieved by the highly scalable framework *LoM2HiS* [79]. The monitoring framework of the FoSII architecture is not part of this thesis. More detail on it is also provided in [81].

Knowledge Management The term knowledge management in this thesis means intelligent usage of measured data from the monitoring phase for the decision making process to satisfy SLAs while optimizing resource usage and consequently energy efficiency and minimizing user interactions with the system. In our approach, this includes not only decision making out of current data, i.e., suggesting actions to be executed, but also improving the quality of decisions by keeping track of the success or failure of previous decisions, i.e., learning. Since the KM system uses monitoring information and directly recommends actions to prevent SLA violations and improve energy efficiency, we combine analysis and planning phases with the knowledge to the new *Knowledge Management Phase* (see part 3, Figure 2.1).

2.3 Escalation Levels – Structuring the Problem

This section presents a methodology of dividing the problem of resource-efficient and energy-efficient SLA enactment in Cloud computing infrastructures into smaller subproblems using a hierarchical approach. This section demonstrates which actions can be executed on what level to achieve SLA adherence and efficient resource allocation for Cloud infrastructures.

In general, we can think of the following reallocation actions:

1. for individual applications:
   a) Increase incoming bandwidth share by $x\%$.
   b) Decrease incoming bandwidth share by $x\%$.
   c) Increase outgoing bandwidth share by $x\%$.
   d) Decrease outgoing bandwidth share by $x\%$.
   e) Increase memory by $x\%$.
   f) Decrease memory by $x\%$.
   g) Add allocated storage by $x\%$.
   h) Remove allocated storage by $x\%$.
   i) Increase CPU share by $x\%$.
   j) Decrease CPU share by $x\%$.
   k) Outsource (move application) to other cloud.
   l) Insource (accept application) from other cloud.
   m) Migrate application to different VM.
2. for VMs:
   a) Increase incoming bandwidth share by x%.
   b) Decrease incoming bandwidth share by x%.
   c) Increase outgoing bandwidth share by x%.
   d) Decrease outgoing bandwidth share by x%.
   e) Increase memory by x%.
   f) Decrease memory by x%.
   g) Add allocated storage by x%.
   h) Remove allocated storage by x%.
   i) Increase CPU share by x%.
   j) Decrease CPU share by x%.
   k) Outsource (move VM) to other cloud.
   l) Insource (accept VM) from other cloud.

m) Migrate VM to different PM.

3. for physical machines (computing nodes):

   a) Add x computing nodes

   b) Remove x computing nodes

4. Do nothing.

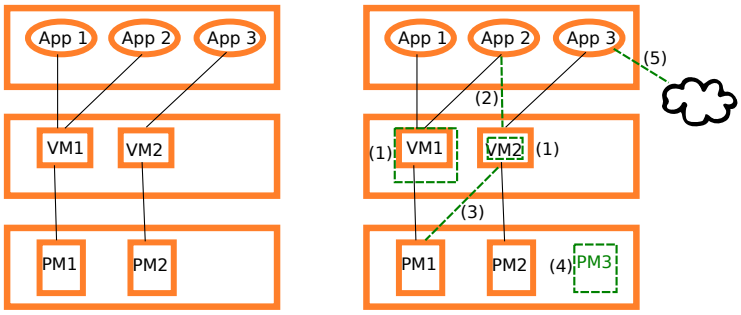
These actions are then grouped into so called escalation levels that are defined in Table 2.2. The idea is that every problem that occurs should be solved on the lowest escalation level. Only if this is not possible, the problem is tried to be solved on the next level, and again, if this fails, on the next one, and so on. The levels are ordered in a way such that lower levels offer faster and more local solutions than higher ones. Escalation level 0 is where no action should be executed. It is important to know when to do nothing, since every reallocation action is time- and energy consuming. In the following, however, we will consider the escalation levels, where actions *are* executed. The first escalation level ("change VM configuration") works locally on a PM and tries to change the amount of storage or memory, e.g., that is allocated to the VM from the PM resources. Then, migrating applications (escalation level 2) is more light-weight than migrating VMs (escalation level 3) and turning PMs on/off (escalation level 4). Already for escalation levels 2-4 the whole system state has to be taken into account to find an optimal solution. The problem stemming from escalation level 3 alone can be formulated into a Binary Integer Programming (BIP) problem, which is known to be NP-hard [105]. The proof is presented in Section 5.2. The last escalation level has least locality and greatest complexity, since the capacity of other Cloud infrastructures have to be taken into account, too, and negotiations have to be started with them as well.

0. Do nothing.
1. Change VM configuration.
2. Migrate applications from one VM to another.
3. Migrate one VM from one PM to another or create new VM on appropriate PM.
4. Turn on / off PM.
5. Outsource to other Cloud provider.

Table 2.2: Escalation levels

Also the rule-based approach benefits from this hierarchical action level model, because it provides a salience concept for contradicting rules. Without this concept it would be troublesome to determine which of the actions, e.g., "Power on additional PM with extra-storage and migrate VM to this PM", "Increase storage for VM by 10%" or "Migrate application to another VM with more storage" should be executed, if a certain threshold for allocated storage has been exceeded.

Figure 2.2 visualizes the escalation levels from Table 2.2 in the context of Infrastructure as a Service (IaaS) before and after actions are executed. Figure 2.2a shows applications App1 and App2 deployed on VM1 that is itself deployed on PM1, whereas App3 runs on VM2 running on PM2. Figure 2.2b shows example actions for all five escalation levels. The legend numbers correspond to the respective numbering of the escalation levels.



(a) Before action execution (b) After action execution

Figure 2.2: Actions used in 5 escalation levels: before and after action execution

- *Escalation level 1*: At first, the autonomic manager tries to change the VM configuration. Actions (1) show *VM1* being up-sized and *VM2* being down-sized.
- *Escalation level 2*: If the attempt to increase a certain resource for a VM in escalation level 1 fails, because some resource cannot be increased anymore due to the constraints of the PM hosting the VM, in level 2 the autonomic manager tries to migrate the application to another larger VM that fulfills the required specifications from level 1. So if, e.g., provided storage needs to be increased from 500 to 800GB, but only 200 GB are available on the respective VM, then the application has to be migrated to a VM that has at least the same resources as the current one plus the remaining 100GB of storage. Action (2) shows the re-deployment of *App2* to *VM2*. Due to possible confinements of some applications to certain VMs, e.g., a user deployed several applications that need to work together on one VM, this escalation might be skipped in some scenarios.
- *Escalation level 3*: If there is no appropriate VM available in level 2, in level 3 the autonomic manager tries to create a new VM on an appropriate PM or migrate the VM to a PM that has enough available resources. Action (3) shows the re-deployment of *VM2* to *PM1*.
- *Escalation level 4*: Again, if there is no appropriate PM available in level 3, the autonomic manager suggests turning on a new PM (or turning it off if the last VM was emigrated from this PM) in level 4. Action (4) shows powering on a new PM (*PM3*).
- *Escalation level 5*: Finally, the last escalation level 5 tries to outsource the application to another Cloud provider as explained, e.g., in the Reservoir project [185]. Action (5) outsources *App3* to another Cloud provider.

The proposed KM approaches in Chapter 4 will present a solution for escalation levels 0 and 1, whereas the solutions presented in Chapter 5 will present solutions for escalation levels 3 and 4. Finally, Chapter 7 presents a KM concept for escalation level 5. Thus, this thesis tackles all the presented escalation levels except for escalation level 2. In Chapters 4-6 we will assume that one application resides on exactly one VM. Under this assumption escalation level 2 becomes obsolete, because VMs can be reconfigured (escalation level 1) or migrated (escalation level 3) if necessary.

CHAPTER 3

SLA Generation and Adaptive SLA Mapping

This chapter will describe the SLA mapping approach, the lifecycle of a public SLA template, and three adaptation methods to change the public template. Furthermore, this chapter will introduce a utility and a cost model to evaluate the adaptation approaches in an emulation environment.

3.1 Outline

In order to facilitate SLA creation and SLA management, SLA templates have been introduced. SLA templates represent popular SLA formats. They comprise elements such as names of trading parties, names of SLA attributes, measurement metrics, and attribute values [184].

Despite the existence of SLAs, buyers and sellers of computing resources face the problem of varying definitions of computing resources in Cloud computing markets. Computing resources are described through different non-standardized attributes, e.g., CPU cores, execution time, inbound bandwidth, outbound bandwidth, and processor type [182]. Sellers use them to describe their supply of resources. Buyers use them to describe their demand for resources. As a consequence, a large variety of different SLAs exists in the market. The success of matching offers from sellers and bids from buyers becomes very unlikely, i.e., the market liquidity (the likelihood of matching offers and bids) becomes very low [184].

Approaches that tackle this plethora of SLA attributes include the use of standardized SLA templates for a specific consumer base [3, 7], downloadable predefined provider-specific SLA templates [4], and the use of ontologies [168, 75]. These approaches clearly define SLA templates and require users to agree a priori on predefined requirements. These SLA templates are static meaning that they do not change nor adapt over time.

Consequently, the existing approaches for the specification of SLA templates cannot easily deal with demand changes. Demand changes of users are caused through different factors (e.g.,

changing market conditions). For example, the emergence of multi-core architectures in computing resources required the inclusion of the new attribute "number of cores", which was not present in an SLA template a couple of years ago. The existing approaches for the specification of SLA templates involve heavy user-interactions to adapt existing SLA templates to demand changes.

In this chapter, we apply adaptive SLA mapping, a new, semi-automatic approach that can react to changing market conditions [184]. This approach adapts public SLA templates, which are used in the Cloud market, based on SLA mappings. SLA mappings, which have been defined by users based on their needs, bridge the differences between existing public SLA templates and the private SLA template, i.e., the SLA template of the user. In our context private templates do not necessarily imply that they are inaccessible to others, but the word "private" is used to differentiate it from the "public" template of the (public) registry. So, all consumers' and providers' templates are called "private", whereas the registry's template is called "public". Since a user cannot easily change the private SLA template due to internal or legal organizational requirements, an SLA mapping is a convenient workaround.

Our adaptive SLA mapping approach can use different adaptation methods. The benefit of using an adaptation method is decreased by some cost for the user. Costs are only incurred, if a user has to define a new SLA mapping to a public SLA template due to its adaptation. Within this chapter, we investigate these costs. In particular, we investigate how public SLA templates can be adapted to the demand of Cloud users and how the costs and benefits differ with respect to the public SLA template adaptation method used.

After introducing a reference adaption method for our analysis, we compare two additional adaptation methods which differ in the heuristics applied. The heuristics have been introduced in order to find a balance between the benefit of having a public SLA template that is identical to most of the private SLA templates and the cost of creating new SLA mappings and new public SLA templates. As the metrics for assessing the quality of the adaptation method, we define the overall system net utility of all users. The net utility considers the benefit of having the same attribute and attribute name in the public SLA template as in the private SLA template, as well as the cost of defining a new SLA attribute mapping.

The benefits of the adaptive SLA mapping approach for market participants are threefold. Firstly, traders can keep their private templates, which are required for other business processes. Secondly, based on their submitted mappings of private SLA templates to public SLA templates, they contribute to the evolution of the market's public SLA templates, reflecting all traders' needs. Thirdly, if a set of new products is introduced to the market, our approach can be applied to find a set of new public SLA templates. All these benefits result in satisfied users, who continue to use the market, therefore increasing liquidity in the Cloud market. However, these benefits come with some cost for the user. Whenever a public SLA template has been adapted, the users of this template have to re-define their SLA mappings.

The four contributions of this chapter are: (1) the definition of three adaptation methods for adapting public SLA templates to the needs of users; (2) the investigation of conditions under which SLA templates should be adapted; (3) the formalization of measures, i.e., utility and cost, to assess SLA adaptations and SLA adaptation methods; and (4) the introduction of an emulation approach for the defined use cases.

3.2 Use Case

This section presents a use case for adaptive SLA mapping.

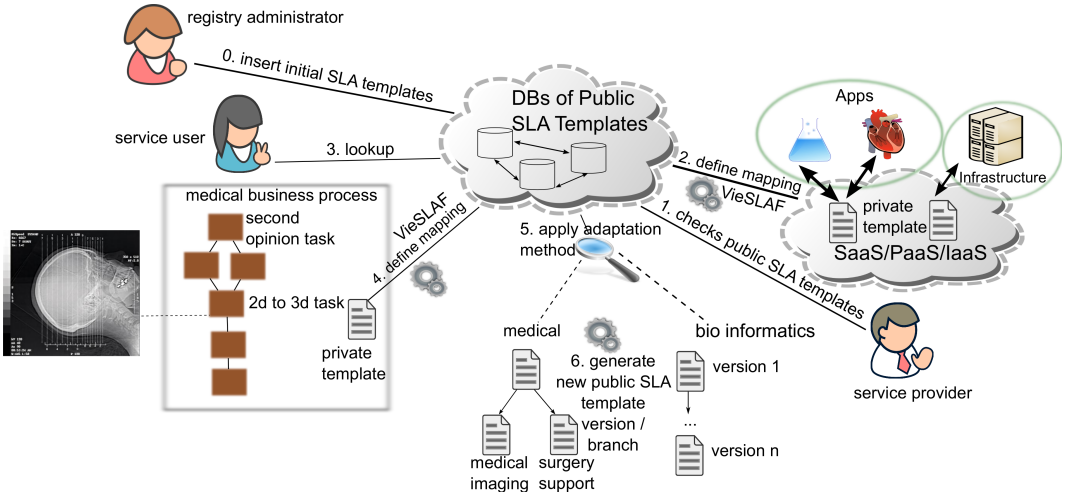


Figure 3.1: Use case of SLA mapping.

At the beginning the registry administrator inserts the initial SLA templates into particular databases (step 0, DBs of public SLA templates, Figure 3.1). As the next step, since resources can be exposed as services using typical Cloud deployment technologies (i.e., SaaS/PaaS/IaaS), we assume that the service provider of Figure 3.1 registers his resources, e.g., infrastructure, software, platforms, to the mentioned databases (step 1, DBs of public SLA templates, Figure 3.1). If some differences between his resources, i.e., his private SLA templates, and the public templates exist, the provider defines SLA mappings, which can transform the private template into the public template and vice versa (step 2, Figure 3.1). Non-technical experts as, e.g., business experts, can easily create their mappings with Web Interfaces or DSLs to define SLA mappings in the simple form "My private template parameter *number of CPUs*" translates to "Public template parameter *CPU cores*". Then, XSLTs can automatically be generated out of this information. The generation and management of SLA mappings, which is performed with VieSLAF, is explained in detail in [51].

In step 3 of Figure 3.1, Cloud users can look up Cloud services that they want to use in their workflow. Looking for public templates (steps 1 and 3) is not affected (slowed down) by the number of issued mappings to the public template, because users still look for the original public template. The figure exemplifies a business process (i.e., workflow) for medical treatments [50]. It includes various interactions with human beings, e.g., the task of getting a second opinion on a diagnosis, as well as an interaction with different infrastructure services. Some of these tasks, e.g., the reconstruction of 2-dimensional SPECT images to 3-dimensional SPECT images, can be outsourced to the Cloud [50]. Thereby, we assume that the private SLA template (representing the task) cannot be changed, since it is also part of some other local business processes and has to comply with different legal guidelines for electronic processing of medical data. Therefore,

in case the user decides to outsource a task and discovers differences between the private SLA template and the public SLA template, the user defines an SLA mapping. In general, the SLA mapping describes the differences between the two SLA templates (step 4). A typical mapping is the mapping of an attribute name to another attribute name (e.g., *number of CPUs* to *cores*) or the inclusion of a new SLA attribute (e.g., *parallel programming models*) into the SLA template. Concerns like patient confidentiality can be enforced by the SLA compliance model proposed in [53].

The public SLA templates are stored in searchable repositories using SQL and non-SQL-based databases (e.g., HadoopDB). The SLA mappings, which have been provided by users and providers to the registry administrator, are evaluated after certain time periods, in order to adapt the public SLA templates to the needs of the users. Then, the adapted public SLA templates replace the existing public SLA templates in the repository, constituting our novel approach of adaptive SLA mapping.

The adaptation method, which adapts the public SLA templates, does this in a way such that the new public SLA templates represent user needs better than the old SLA templates (step 5). The adaptation of attributes, attribute names, and attribute values cannot only replace SLA templates but also create new versions and branches of public SLA templates (step 6). A new branch of a public SLA template can be created, if specialization needs to be captured (e.g., a medical SLA template can be substituted by more specialized templates on medical imaging and surgery support). The creation of new branches has been more thoroughly examined in [54]. The definition of different versions of a particular public SLA template occurs, if different attribute combinations in the templates are used. Figure 3.1 shows n template versions in the bioinformatics domain.

3.3 Public SLA Template Life Cycle

To illustrate the life cycle of public SLA templates, Figure 3.2 shows a short example first.

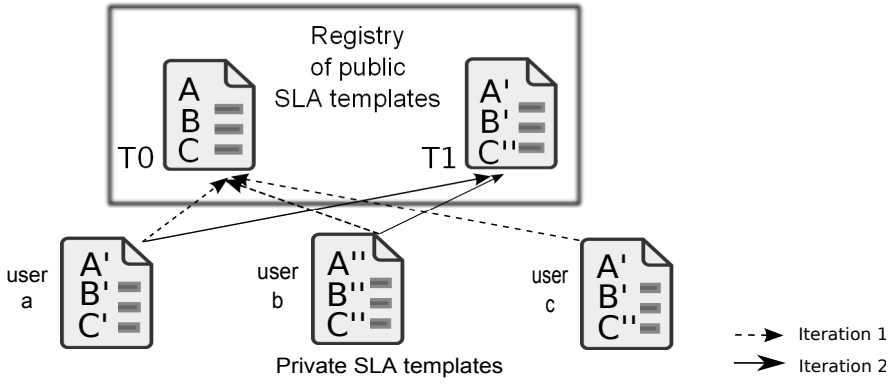


Figure 3.2: SLA mapping process.

Initially, the SLA template registry only holds the initial public SLA template T_0. In iteration 1, all users define mappings from their private templates to T_0. Since the attribute names of the

public SLA template *(A, B, C)* and the attribute names of each user differ, all users have to create 3 attribute mappings. Based on these mappings, the new version T_1 of the public template is generated (according to the adaptation method used), containing the attribute names *A', B', C''*.

Since the public SLA template has changed, users need to change their mappings as well (iteration 2). Consequently, user *a* only needs one attribute mapping, user *b* needs two attribute mappings, and user *c* does not need to issue any attribute mapping, since the public template is completely identical to her private template. This example shows how our adaptive SLA mapping approach adapts a public SLA template to the needs of users. In addition to this, since adapted public SLA templates represent the need of market participants, it is most likely that new requests of users need less attribute mappings, reducing the cost for these users.

The formalized public SLA template life cycle, which consists of five steps, is shown in Figure 3.3.

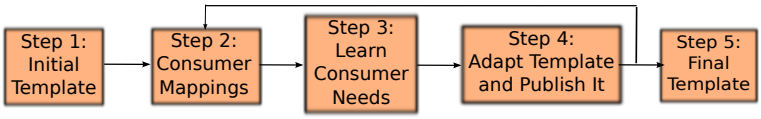


Figure 3.3: Formalized public SLA template life cycle.

An initial template is created in the beginning of the life cycle (step 1, Figure 3.3). Afterwards, consumers perform SLA mappings to their private SLA templates (step 2). Based on their needs, inferred from these mappings (step 3), and the predefined adaptation method, the public SLA template is adapted (step 4). Assuming that the demand of market participants does not change, a final template is generated (step 5). If the demand has changed during a fixed time period (i.e., new tasks have to be executed or new users joined the marketplace), the process continues with step 2. In practice, the time between two iterations could correspond to a time period of one week, e.g., but can be set to any value depending on the volatility of the market. During that time new SLA mappings are solicited from users (i.e., consumers and providers).

3.4 Adaptation Methods

This section introduces the utility and cost model for assessing SLA adaptation approaches.

The adaptation methods determine for every attribute name of the public SLA template separately, whether the current attribute name should be adapted or not. In this chapter, we investigate three adaptation methods. The first adaptation method is the maximum method (which has been applied in the example shown in Figure 3.2). The remaining two adaptation methods apply heuristics, in order to find a balance between benefit and cost.

Maximum Method

Applying this method, the SLA attribute name, which has the highest number of attribute name mappings, is selected (maximum candidate). The selected attribute name will become the next attribute name of the next public SLA template.

Example: If we assume that all attribute names have the same count, this method would select any of the four possible attribute names randomly. If a public SLA template already exists, the method will choose the attribute name that is currently used in the public SLA template.

Threshold Method

In order to increase the requirements for selecting the maximum candidate, this method introduces a threshold value. If an attribute name is used more than this threshold (which can be adapted) and has the highest count, then this attribute name will be selected. If more than one attribute name is above the threshold and they have the same count, the method proceeds as described for the maximum method. If none is above the required threshold, then the method sticks to the currently used attribute name. Note, throughout the examples in this chapter, we fix the threshold to 60%. A smaller threshold makes this method more similar to the maximum method. A threshold of 0% would make this method identical to the maximum method. A greater threshold makes changes in the SLA attribute name more unlikely and very similar to the static approach that does not change SLA templates at all.

Example: Assuming an example in which none of the attribute names has a mapping percentage above 60% and all counts are equal, the threshold method sticks to the attribute name that is currently used in the public SLA template.

Maximum-Percentage-Change Method

This method is divided into two steps. In the first step, the attribute name is chosen according to the maximum method.

In the second step, which comprises τ iterations, attribute names will be changed, only if the percentage difference between the highest count attribute name and the currently selected attribute name exceeds a threshold. The threshold σ_T is set to 15% within this chapter. A low threshold leads to more mappings, whereas a high threshold leads on average to fewer mappings. After τ iterations (e.g., $\tau = 10$), the method re-starts with executing the first step. This allows slighter changes to take effect.

Example: Let us suppose the mapping count resulted in attribute name A' having the highest count. By applying the maximum method, A' is selected. In the next iteration, the number of mappings for each attribute name has changed. Attribute name A accounted for 10%, A' for 28%, A'' for 32%, and A''' for 30% of all mappings. Assuming a threshold of 15%, the chosen attribute does not change. The percentage difference between attribute name A' and the attribute name A'' with the highest count is only $32/28 - 1.0 = 14.3\%$.

3.5 Utility and Cost Model

Since the aim of this chapter is to assess the benefit and the cost of using the adaptive SLA mapping approach for finding the optimal standardized goods in a Cloud market, we define a utility and cost model. At its core, the model defines the utility function and the cost function. The utility function and the cost function, which take attributes of the private SLA template of

the customer and the attributes of the public SLA template as input variables, help to quantify the benefit and the cost.

The model assumes an increase in benefit, if an attribute (or attribute name or attribute value) of both templates is identical. This is motivated by the fact that the Cloud resource traded is identical to the need of the buyer (or, in the other case, the provisioned resource of the provider) and, therefore, no inefficiency through resource over-provisioning occurs. The model also captures the effort (i.e., cost) of changing an SLA mapping. The cost is only incurred, if the user needs to change her SLA mapping because of a change in the public SLA template.

To formally introduce these functions, we introduce some definitions. The set of SLA attributes is defined as T_{var}. As an example, we set $T_{var} = \{\alpha, \beta\}$, where α represents *Number of Cores in one CPU* and β represents *Amount of CPU Time* (Note, α and β could also represent attribute values). All possible attribute names that a user can map to a $\pi \in T_{var}$ are denoted as $Var(\pi)$. Within this example, we set $Var(\alpha) = \{A, A^{'}, A^{''}, A^{'''}\}$, representing *Var("Number of Cores in one CPU") = {CPU Cores, Cores of CPU, Number of CPUCores, Cores}*, and $Var(\beta) = \{B, B^{'}, B^{''}, B^{'''}\}$.

Assuming a set of private SLA templates $C = \{c_1, c_2, \ldots, c_n\}$ of customers, we can now define the relationship of a specific SLA attribute to a specific attribute name of this SLA attribute at a specific point in time (i.e., iteration) $i \in N$ for an SLA template p, $p \in C \cup \{T\}$ (i.e., private or public SLA template) as

$$SLA_{p,i} : T_{var} \rightarrow \bigcup_{\pi \in T_{var}} Var(\pi). \tag{3.1}$$

With respect to our example, we assume $SLA_{T,0}(\alpha) = A$ and $SLA_{T,0}(\beta) = B$ as our initial public template T at time 0 (i.e., iteration 0).

Based on these definitions and the utility function exemplified in [65], we define the utility function $u^{+}_{c,i}$ and the cost function $u^{-}_{c,i}$ for consumer c, attribute $\pi \in T_{var}$, and iteration $i \geq 1$ with $W^{+} \geq W^{-} \geq 0$ as

$$u^{+}_{c,i}(\pi) = \begin{cases} W^{+}, & SLA_{c,i}(\pi) = SLA_{T,i}(\pi) \\ 0, & SLA_{c,i}(\pi) \neq SLA_{T,i}(\pi) \end{cases} \tag{3.2}$$

$$u^{-}_{c,i}(\pi) = \begin{cases} 0, & SLA_{c,i}(\pi) = SLA_{T,i}(\pi) \\ 0, & SLA_{c,i}(\pi) \neq SLA_{T,i}(\pi) \wedge \\ & SLA_{T,i-1}(\pi) = SLA_{T,i}(\pi) \\ W^{-}, & SLA_{c,i}(\pi) \neq SLA_{T,i}(\pi) \wedge \\ & SLA_{T,i-1}(\pi) \neq SLA_{T,i}(\pi) \end{cases} \tag{3.3}$$

The utility function states that a consumer c receives a utility of W^{+}, if the name of the attribute of the private SLA template matches the name of the public SLA template attribute, and a utility of 0 otherwise.

In this context cost is defined as the negative utility for a consumer relating to the effort of generating a new SLA mapping. The cost function states that a consumer has a cost of W^{-}, if the attribute names do not match and the public template attribute of the previous iteration has

been adapted to a new one. In this case, the consumer has to define a new attribute mapping, as he cannot use the old one anymore. The cost of issuing a new mapping should be lower than the utility of standardizing SLA attributes by achieving the same attribute names. This is why $W^+ \geq W^-$. Here we set $W^+ = 1$ and $W^- = 1/2$.

In the other two cases, the consumer has no cost, since either the attribute names match or the public template attribute name did not change since the previous iteration. That means he does not need any new mapping. Thus, for attribute π, the consumer c at iteration i gets the net utility

$$u^o_{c,i}(\pi) = u^+_{c,i}(\pi) - u^-_{c,i}(\pi). \tag{3.4}$$

The net utility for all attributes at iteration i for consumer c is defined as the sum of the net utilities $u^o_{c,i}(\pi)$:

$$u^o_{c,i} = \sum_{\pi \in T_{var}} u^o_{c,i}(\pi). \tag{3.5}$$

In addition to this, the overall utility and overall cost (i.e., the utility and cost of all users C and attributes π at iteration i) are defined as:

$$U^+_i = \sum_{c \in C} \sum_{\pi \in T_{var}} u^+_{c,i}(\pi) \tag{3.6}$$

$$U^-_i = \sum_{c \in C} \sum_{\pi \in T_{var}} u^-_{c,i}(\pi) \tag{3.7}$$

Consequently, the overall net utility at iteration i is defined as the difference between the overall utilities minus the overall cost or as the sum of the net utility of all consumers c for all attributes at iteration i:

$$U^o_i = {U^+}_i - U^-_i = \sum_{c \in C} u^o_{c,i}. \tag{3.8}$$

3.6 Simulation Environment

In order to evaluate the performance of the three adaptation methods with respect to the proposed utility and cost model, we set up a simulation environment.

Testbed

For the simulation, we use a testbed that is composed of a scientific prototype (*VieSLAF*) [51] and software that simulates SLA mappings of users. Figure 3.4 illustrates our *emulation* testbed. The components that are drawn in white belong to *VieSLAF*. It comprises the knowledge base, the middleware for managing SLA mappings provided by consumers and providers, and the adaptation methods. The grey components indicate the components that simulate SLA mappings of users.

A sample provider and a sample consumer are shown in the lower part of Figure 3.4.

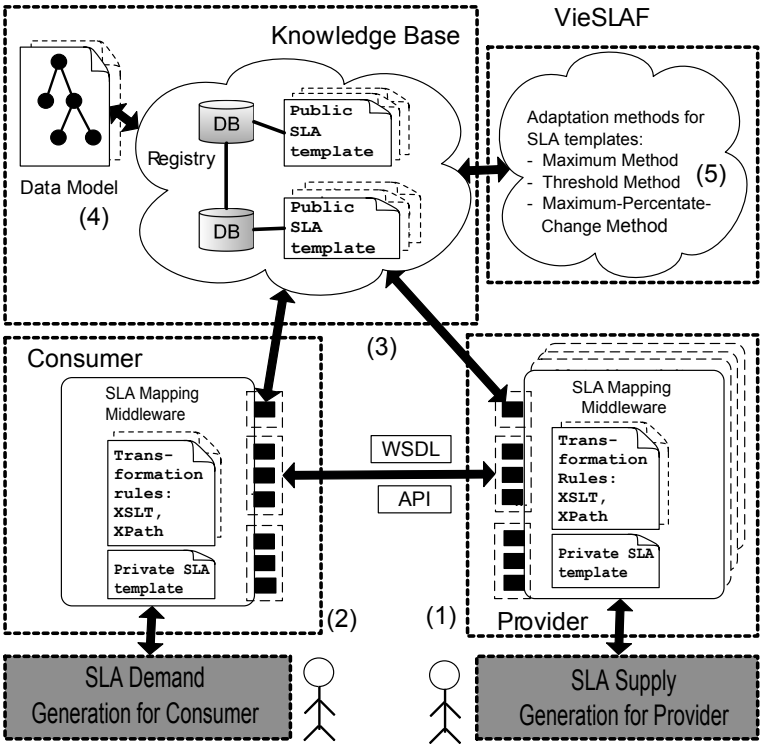


Figure 3.4: Adaptive SLA mapping architecture using VieSLAF.

The SLA mapping middleware, which follows a client-server design, facilitates the access by the provider and the consumer to registries. It provides to users a GUI for browsing public SLA templates. The SLA mapping middleware is based on different Windows Communication Foundation (WCF) services, of which only a few are mentioned in the following paragraph.

The *RegistryAdministrationService* provides methods for the manipulation of the database. This service requires administrator rights. An example for these methods is the creation of template domains. Another service of the SLA mapping middleware is the *SLAMappingService*, which is used for the management of SLA mappings by service consumers and service providers (cf. (3) of Figure 3.4). Providers and consumers may also search for appropriate public SLA templates through *SLAQueryingService* and define appropriate SLA mappings by using the method *createAttributeMapping*. With each service request, it is also checked whether the user has also specified any new SLA mappings. The SLA mappings (i.e., transformation rules) are stored in the private database of the user and can be re-used by the user for her next SLA mapping.

The knowledge base for storing the SLA templates in a predefined data model ((4) of Figure 3.4) is implemented as registries representing searchable repositories. Currently, we have implemented an MS-SQL 2008 database with a Web service frontend. To handle scalability issues, we intend to utilize non-SQL DBs (e.g., HadoopDB) with SQL-like frontends (e.g., Hive [197]). SLA templates are stored in a canonical form, enabling the comparison of the XML-based SLA

templates. The registry methods are also implemented as WCF services and can be accessed only with appropriate access rights. The access rights distinguish three access roles: *consumer*, *provider* and *registry administrator*. The registry administrator may create new SLA templates. A service consumer and a service provider may search for SLA templates and can submit their SLA mappings.

Based on the submitted SLA mappings, public SLA templates are adapted by the registry administrator, using one of the adaptation methods ((5) of Figure 3.4), introduced in section 3.4.

Simulation Parameter Settings

For the simulation, we define five scenarios on how often attribute names occur in private SLA templates on average. In particular, each scenario defines an occurrence distribution of four different SLA attribute names. Our observations indicate that four different SLA attribute names seem to be a reasonable number, especially when referring to the example given in Section 3.5 with SLA attribute names *CPU Cores*, *Cores of CPU*, *Number of CPUCores*, and *Cores*. Another example would be the four names *Cost*, *Charge*, *Rate* and *Price* for one SLA attribute. With four attributes set, we can partition all possible *and* interesting (i.e., leading to a different outcome in any of the adaptation methods) situations into exactly five different scenarios that are defined as follows:

- Scenario a: All attribute name counts of an attribute are equal.
- Scenario b: The counts of three attribute names are equally large and larger than the remaining one.
- Scenario c: Two attribute name counts are equally large and are larger than the other two, which are equally large as well.
- Scenario d: One attribute name, which has been picked as the attribute name for the initial setting, has a larger count than the counts of the remaining three attribute names, which are equally large.
- Scenario e: One attribute name, which has not been picked as the attribute name for the initial setting, has a larger count than the counts of remaining three attribute names, which are equally large.

The actual values of each of the five scenarios are shown in Table 3.1. The four attribute names chosen for this example are: A, A', A'', A'''. The initial setting of attribute α is the attribute name A.

As an example for the use of the scenarios, we take scenario c. If attribute α (*Number of Cores in one CPU*) is distributed according to scenario c, then the four attribute names occur in average as follows: 10% of the attribute names is A, 10% of the attribute names is A', 40% of the attribute names is A'', and 40% of the attribute names is A'''. However, as we intend to account for slight changes in the demand for attribute names by users, we draw randomly the attribute names according to the distribution given in Table 3.1 instead of generating the exact number of

Table 3.1: Average occurrence of attribute names in all scenarios.

	Scenarios [%]				
	a	b	c	d	e
A	25	10	10	30.0	23.3
A'	25	30	10	23.3	30.0
A"	25	30	40	23.3	23.3
A"'	25	30	40	23.3	23.3

attribute names. Consequently, the actual counts of attribute names might vary compared to the average values shown in Table 3.1. As an example, the attribute names generated according to the distribution of scenario *c* might be 9%, 12%, 37%, and 42% instead of 10%, 10%, 40%, and 40%. This process of generation of attribute names is executed for each iteration.

Furthermore, another three simulation parameters are set. First, the number of iterations is limited to 20. This number is chosen, because from iteration to iteration the consumer base does not evolve (the consumers obey to the same distribution in every iteration, but the quite low number of users reveals different random samples of the distributions). 20 iterations are long enough to examine the natural market fluctuations, but more iterations would not reveal any new information. At each iteration, 100 users perform SLA mappings to all SLA attributes. The number is not set higher in order to mimic natural market fluctuations. At the end of an iteration, a new public SLA template is generated, which is based on the adaptation method and the SLA mappings of the user. As in our evaluation setting the market will not stabilize, a final template as described in Figure 3.3 will not be achieved. For each of the three adaptation methods we execute one separate simulation run. Moreover, the SLA template consists of five SLA attributes, whose attribute names are distributed according to scenarios a-e, respectively. This way, the results for utility and cost will be averaged values over all five scenarios. Table 3.2 summarizes these settings.

Table 3.2: Simulation parameter settings.

Simulation Parameter	**Value**
Number of scenarios	5
Number of users (consumers & providers)	100
Number of SLA attributes per SLA template	5
Number of SLA attributes names per attribute	4
Number of adaptation methods applied	3
Number of iterations	20

3.7 Experimental Results and Analysis

Net Utilities of Adaptation Methods

Using our SLA mapping approach, the user benefits by having access to public SLA templates that reflect the overall market demand (i.e., the demand of all users). This benefit of a user is expressed by Equation (3.2). However, this benefit comes with the cost for defining new SLA mappings whenever the public SLA template changed (Equation (3.3)).

Within this section, we investigate the cost for all users (Equation (3.7)), the utility of all users (Equation (3.6)), and the net utility of all users (Equation (3.8)) with respect to three adaptation methods. The net utility metric is used to decide which of the three investigated adaptation methods is superior.

The first investigated adaption method is the maximum method. It is our reference method, since it does not use any heuristics. The simulation results, which are shown in this section, have been obtained from running the simulation with parameter settings as described in Section 3.6.

Figure 3.5 shows the resulting public SLA templates over the iterations. For every of the five possible parameter attributes a line indicates which SLA parameter name has been chosen for the specific iteration.

The advantage of the maximum method is that the public SLA template generated with this method minimizes the differences to all private SLA templates of all users. This method, however, requires many SLA mappings.

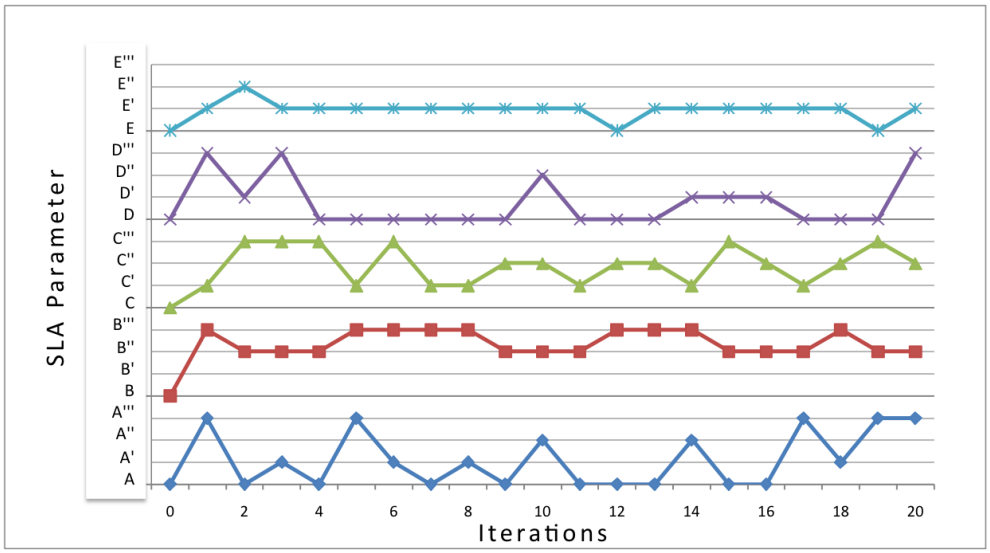


Figure 3.5: Public templates for the maximum method.

Figure 3.6 shows, as expected, that the maximum method generates a high utility, since it achieves many matchings of attribute names of the public SLA template and the private SLA templates. Its utility stays around its initial utility value of about 170 for each iteration. However, as expected as well, it requires many new mappings and, thus, incurs high costs to the users.

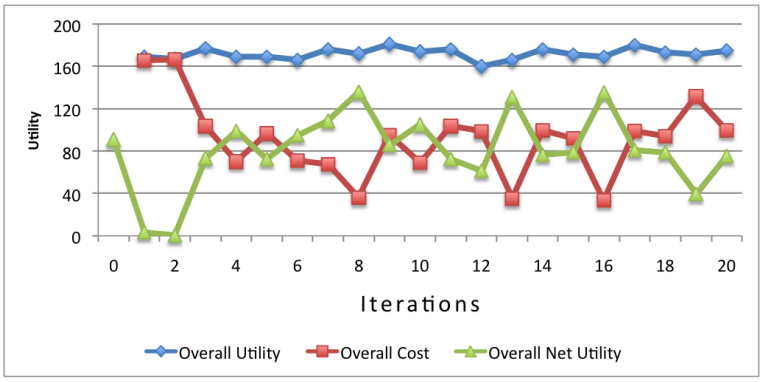


Figure 3.6: Utility, cost, and net utility for the maximum method.

Consequently, the net utility is far lower than the utility.

In order to address this issue of high cost of the maximum method, we use heuristics in the following two adaptation methods. The heuristics help to find a balance between the utility of having a public SLA template, whose attribute names are identical to most of the attribute names of the private SLA templates, and the cost of creating new SLA attribute mappings. The first heuristics-based adaptation method, which we investigate, is the threshold method. The simulation results are shown in Figures 3.7 and 3.8.

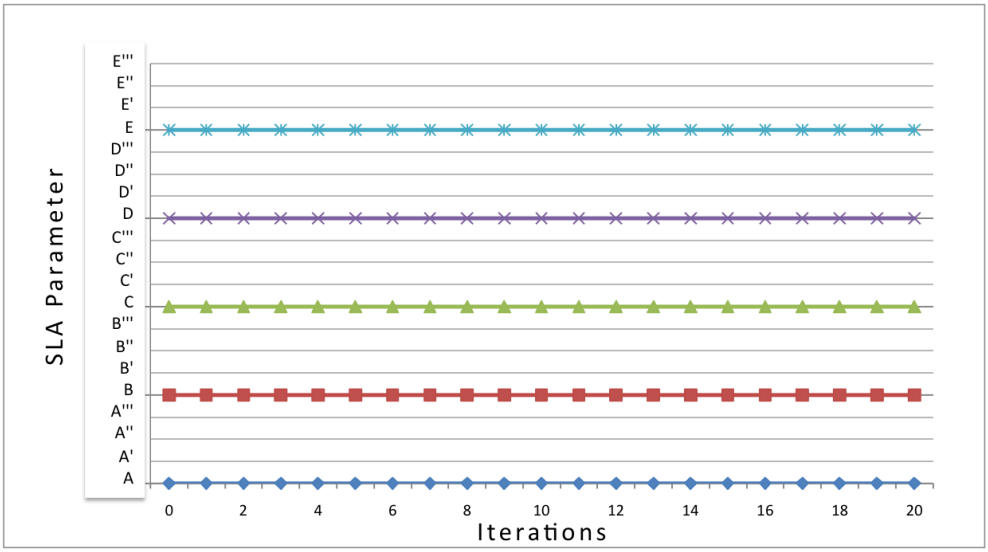


Figure 3.7: Public templates for the threshold method.

Figure 3.8 illustrates that the threshold method does not incur any cost to users at all, because Figure 3.7 does not reveal any changes to the initially set parameter name at all. This is due to the high threshold (i.e., a threshold of 60%), resulting in no changes of the public SLA template attribute names. Nevertheless, the utility (and net utility) is not higher than the ones of the

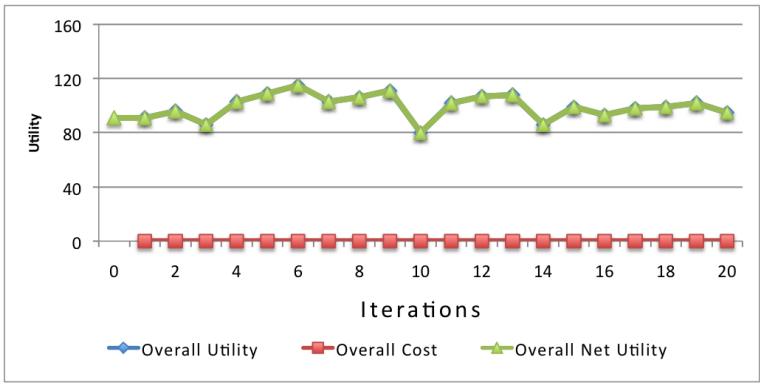


Figure 3.8: Utility, cost, and net utility, for the threshold method.

maximum method, just more stable across the 20 iterations. Therefore, the threshold method with a threshold of 60% could be considered the opposite strategy to the maximum method. That means, the initial public SLA template does not get adapted at all. By lowering the threshold parameter such that the threshold parameter for a few iterations is lower than the highest count of an attribute name, it is expected that the net utility improves. If the threshold parameter is lower than the minimum count of an attribute name in all iterations, then this method is identical to the maximum method.

The maximum-percentage-change method is the second investigated heuristics-based adaptation method. The results are shown in Figures 3.9 and 3.10.

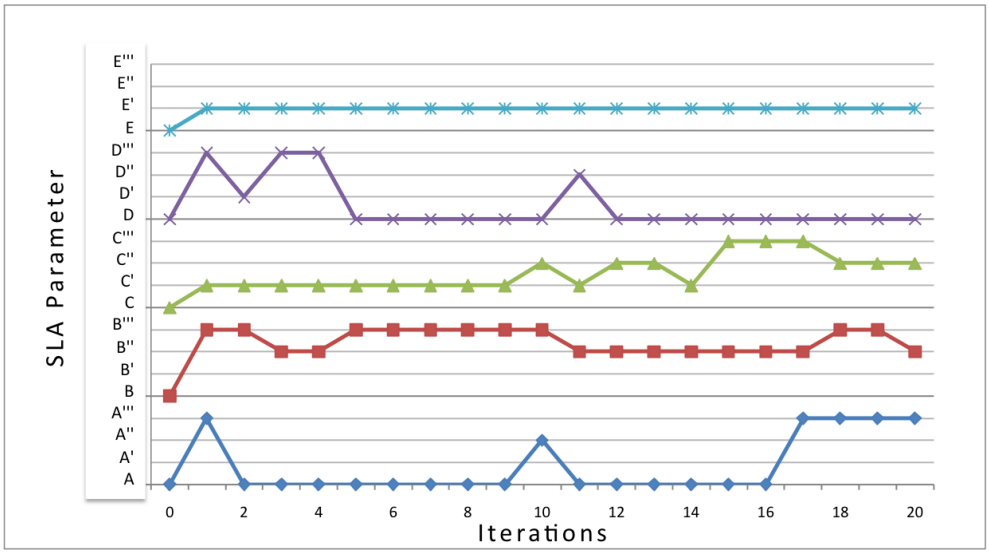


Figure 3.9: Public templates for the maximum-percentage-change method with $\tau = 10$.

The simulation results show that in the first iteration and every tenth iteration ($\tau = 10$) the overall net utility decreases significantly due to the high amount of new SLA mappings needed

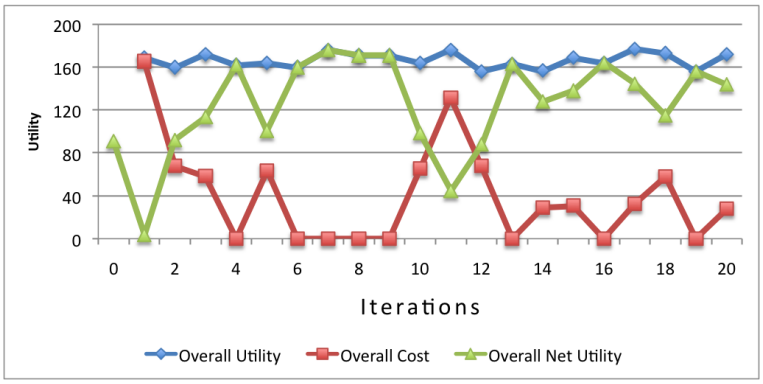


Figure 3.10: Utility, cost, and net utility for the maximum-percentage-change method with $\tau = 10$.

(compare the same iteration within Figures 3.9 and 3.10). At these iterations, the cost of the SLA mappings is very high, since this method chooses the attribute names with the maximum number of counts (not considering the threshold of 15%). In the subsequent iterations, however, the cost is low and, therefore, the overall net utility increases significantly. It achieves even higher values than the other two methods.

Average Cost and Average Net Utility

Table 3.3 shows the average overall utility, average overall cost, and the average overall net utility for all three adaptation methods. The averages are calculated over all iterations. The maximum method has achieved the highest average overall utility. It satisfies the largest number of users. However, since it also incurs the highest costs, it becomes the method with the lowest average overall net utility.

Table 3.3: Overall utility, overall costs, and overall net utilities averaged across all iterations (The best values are highlighted in bold).

	Maximum	**Threshold**	**Max.-Perc.-Change**
avg. overall utility	**171.9**	99.5	166.6
avg. overall cost	91.3	**0.0**	39.95
avg. overall net utilities	80.6	99.5	**126.65**

The threshold method does slightly better with respect to the average net utility than the maximum method. This is due to the zero cost. The threshold method (with a high threshold) stays with the initial SLA attribute name for the public SLA template.

The best adaptation method with respect to the average overall net utility is the maximum-percentage-change method. We observe that the average overall net utility is better than the ones of the other two adaptation methods, although the average overall utility is not the highest among the three adaptation methods. The reason is that the cost is low. The low cost is a result of the

fact that the SLA attribute names of the public SLA template are not changed frequently. They are only changed in iterations $k\tau + 1, k \in N_0$ (i.e., when the method behaves like the maximum method) and whenever the threshold of 15% is exceeded.

Based on the result shown in this section, we can state that the adaptive SLA mapping approach is a good way of generating standardized goods, which address the needs of the market. To reduce the cost for creating SLA mappings frequently, the introduction of heuristics into the adaptation methods is helpful. Results show that a significant reduction of costs can be achieved while preserving the benefit of adapted public SLA templates.

CHAPTER 4

Self-adaptive and Resource-Efficient SLA Enactment for Cloud Computing Infrastructures Using Knowledge Management

In this chapter we conduct a preliminary evaluation of knowledge management techniques suitable for Cloud computing infrastructures. From this preliminary evaluation we concentrate on the two most promising techniques: case based reasoning and rules. We will design and implement these approaches, and finally devise a methodology to self-adapt all crucial parameters for the rule-based approach using a method based on utility and another one based on workload volatility, i.e., the intensity of workload dynamism.

4.1 Methods of Knowledge Management for SLA Management

This section describes some well-known knowledge management methods and presents a preliminary analysis for the use of SLA enactment in a Cloud infrastructure following a use case.

Use Case

This section defines a use case that will be utilized for the examination of the knowledge management methods. An example SLA is depicted in Table 2.1, from which we consider four Service Level Objectives (SLOs) for this analysis: incoming bandwidth (IB), outgoing bandwidth (OB), storage (St), and availability (Av). The corresponding SLO values are shown on the right hand side in Table 2.1. In order to evaluate the knowledge management approaches we describe the status of the system in terms of running *physical machines (PMs)* and a specific application running under this SLA at three different time points t_1, t_2, t_3. We assume that one

application is running on one virtual machine (VM), but one VM can run on (1,*) PMs, and on one PM, there can run (1,*) VMs. Table 4.1 summarizes the system states we have measured.

	IB	**OB**	**St**	**Av**	**PMs**
t_1	12.0	20.0	1200	99.50	20
t_2	14.0	18.5	1020	99.47	17
t_3	20.0	25.0	1250	99.60	19

Table 4.1: Sample system states

Rule-based System

A rule-based system such as Jess [8] or Drools [15] contains rules in the "IF *Condition* THEN *Action*" format, e.g.,

(1) IF IB $<$ TT_IB THEN Add physical machine to VM.
(2) IF IB $<$ TT_IB THEN Increase IB share by 5% for VM.
(3) IF Av $<$ TT_Av THEN Add 2 comp. nodes to the cloud.
(4) IF Av $<$ TT_Av THEN Outsource app. to other cloud.

Here we use *threat thresholds* (TTs) to trigger some action *before* an SLA is violated. There are two drawbacks to this mechanism, though:

First, the question of how these TTs are obtained, has to be answered. They are very different from one SLA parameter to another, e.g., for "SLO Storage $>$ 1024 GB", the *TT* could be already at 1300 GB (127% of the original SLO), whereas for the SLO "IB $>$ 10 Mbit/s" the TT could be at 11 Mbit/s (110% of the original SLO), as one might say that reallocating bandwidth shares is a lot quicker than reallocating storage. They can even differ a lot for the same parameter in a different domain, e.g., the TT for availability in some medical domain, where human lives can be at stake, must be much higher than for a 3D rendering service in the architectural domain. A way to get around this would be to have the TTs specified in DSLs or to include them in the SLA document. However, this would heavily depend on subjective estimations. Nevertheless, it would be possible to find some experience values that make sense for the most common parameters. Furthermore, it has to be specified whether these thresholds are derived from a constant function of the parameter's value, e.g., always add 5 units to the SLA parameter value, from a linear one, e.g., always add 10% to the value, or even from an exponential or from any other function. So to solve this in a universally valid way, one would have to find an appropriate function for every SLA parameter in every domain.[1]

The second question is how to solve two contradicting rules. Consider rules (3) and (4) depicted at the begining of this section. If availability for a certain service drops below the pre-specified TT, should the rule engine rather add computing nodes or outsource an application, or both? Using a salience concept to decide this, leads to a difficultly manageable load of rules.[2] A good examination of this problem can also be found in [109].

[1] This is actually what we do in Section 4.5.

[2] The introduced escalation levels (cf. Chapter 2) will help to mitigate this problem.

In our use case from Table 4.1, the rules (1) - (4) above, and with $TT_{IB} = 12.5$ for incoming bandwidth and $TT_{Av} = 99.48$ for availability, the rule engine would fire rules 1 and/or 2 at time t_1; at t_2 it would fire rules 3 and/or 4, and at t_3 it would do nothing.

Default Logic

Default Logic [33] is a version of a rule-based system whose rules are no longer simple IF-THEN Rules, but can be described as *IF condition - and there are no reasons against it - THEN action*. We write such a rule as $\delta = \frac{\phi:\psi_1,\ldots,\psi_n}{\chi}$, where ϕ represents the condition, and χ is the action to execute, if the statements $\psi_1, \ldots, \psi_n$ are consistent with the current assumptions we hold of our system. A sample rule considering our case study can be written as

$$d_1 = \frac{IB < TT_{IB} : IncreaseIBshare}{IncreaseIBshare}. \tag{4.1}$$

The rule means: *If incoming bandwidth is smaller than its threat threshold, and if there is no reason against increasing bandwidth share, then increase bandwidth share*. Reasons against could be that the bandwidth share is already at its maximum or that other (possibly more important) services issued a request for an increase at the same time. Contrary to ordinary rules in a rule-based system, it is easy for default rules to understand that resources cannot be increased indefinitely. However, default logic does not offer a remedy against the issues of retrieving TTs and dissolving contradicting rules.

Furthermore, default logic is especially used in fields with a lot of contradicting information. For Cloud computing, however, we are rather interested in determining the reason of the current measurement information, e.g., why current incoming bandwidth has decreased. For example, we want to know whether the current bandwidth problem is caused by internal problems (e.g., too many service requests but too little resources provided), which the Cloud is capable of solving on its own, or by external factors (e.g., a DDoS attack), which cannot be influenced directly. Thus, we are rather confronted with more incomplete information than with contradicting one.

Situation Calculus

Situation Calculus [135] describes the world we observe in *states*, the so called *fluents*, and *situations*. *Fluents* are first-order logic formulae that can be true or false based on the situation in which they are observed. *Situations* themselves are a finite series of actions. The situation before any action has been performed - the starting point of the system - is called the initial situation. The state of a situation s is the set of all fluents that are valid in s. Predefined *actions* can advance situations to new ones in order to work towards achieving a pre-defined goal by manipulating the fluents in this domain. For a world of three bricks that can be stacked upon each other lying on a table, fluents are quite easy to find: First, a brick can be on the table or not. Second, a brick can have another brick on it or not. Third, a brick x can lie on a brick y or not. Two possible actions are: Stack brick x on brick y and unstack brick y, i.e., put brick y onto the table. Now, a goal could be to have one pile of all three bricks in a specified order with an initial situation of them being piled in the reverse order. In each state of a situation, different fluents

are true (e.g., brick x lies on brick y, brick y does not lie on brick x, brick z lies on the table), and stacking or unstacking generates a new situation.

To map this analogy to Cloud Computing is not as easy. As far as fluents are concerned, in a Cloud we have to consider the current value of each specific parameter, and whether the respective SLO is fulfilled or not. Furthermore, all the states of the Cloud itself like number of running virtual machines, number of physical machines available, etc., have to be modeled as fluents as well. Fluents for a specific application could be the predicate *has_value(SLAParameter p, Value v)* with $v \in \mathbb{R}^3$ meaning that the SLAParameter p holds the value v in the current situation, and *fulfills(SLO s)* meaning that the specified application fulfills a certain SLO s. The predicate *has_value(SLAParameter p1, x)* is valid for only one $x \in \mathbb{R}$ in a certain situation. The possible actions are provided by our use case.

Since we always observe the Cloud with all its states as a whole, it can be very difficult to derive exactly one action that could lead to an advancement of achieving a goal. The solution could be to view applications isolated from each other and to have one overall view that only takes into account some higher-level information like *fulfillsSLA(Application app)* meaning an application fulfills all its current SLOs at the moment. A doable way of defining goals could be to define utility functions that state the utility of a service fulfilling its SLA. Parameters of this utility function can be the importance of the consumer and the penalty that has to be paid when violating each SLO. The system then tries to find actions to maximize the utility.

Consider a Cloud servicing 100 applications with five SLA parameters each. This leads to $100*(5+1) = 600$ different fluents, like *has_value(SLAParameter p1, x)*, *has_value(SLAParameter p2, y)*, etc., and *fullfills(SLO s)* for every application. Thus, the largest obstacles to this approach are the large number of fluents and the immense search space for the possible actions as a result thereof.

Case Based Reasoning

Case Based Reasoning is the process of solving problems based on past experience [27]. In more detail, it tries to solve a *case* (a formatted instance of a problem) by looking for similar cases from the past and reusing the solutions of these cases to solve the current one. In general, a typical CBR cycle consists of the following phases assuming that a new case was just received:

1. Retrieve the most similar case or cases to the new one.
2. Reuse the information and knowledge in the similar case(s) to solve the problem.
3. Revise the proposed solution.
4. Retain the parts of this experience likely to be useful for future problem solving.

In step 4, the new case and the found solution is stored in the knowledge base. In the following section, we will show how we adapt CBR to the needs of SLA enactment in the field of Cloud computing.

[3] Instead of $\mathbb{R}$ one could consider using different sets with an only finite number of elements, as the set of floating point numbers.

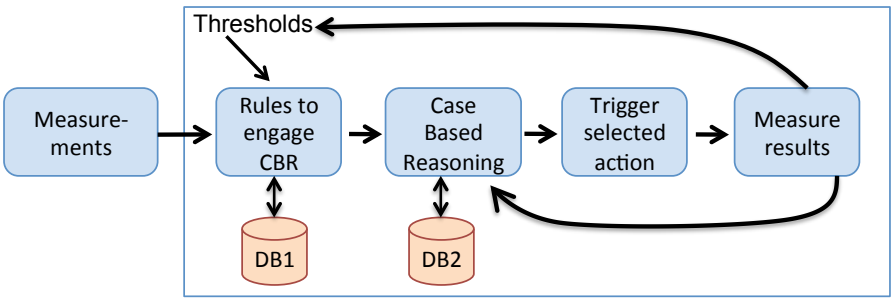


Figure 4.1: The process of Case Based Reasoning

CBR Adapted to SLA Enactment

This section discusses the basic CBR model used for SLA enactment and some of its variations.

Following the diagram in Figure 4.1, the basic idea is to have rules stored in database 1 that engage the CBR system once a TT value has been reached for a specific SLA parameter. The measurements are fed into the CBR system, surrounded by the frame, as a new case by the monitoring component. Then, CBR prepared with some initial meaningful cases stored in database 2, chooses the set of cases which are most similar to the new case by various means as described in Section 4.1. From these cases we select the one with the highest utility measured before. Now we trigger the action that was executed in the selected case. Finally, we measure the result of this action in comparison to the initial case some time intervals later and store it with the calculated utilities as a new case to CBR. Summing up, we have the following basic process (cf. Figure 4.1): New Measurements arrive (Measurements) → Check whether the TTs reached for some parameter (Rules to engage CBR). If yes, choose a set of most similar cases in CBR and from them choose the one with the highest utility (Case Based Reasoning) → Execute action of this case (Trigger 1 action) → Calculate utility of this action by measuring results (Measure results) → Store case in CBR (Feedback). Doing this, we can constantly learn new cases and evaluate the usefulness of our triggered actions. By measuring the utility after more than one time interval, CBR is also able to learn whether an action was carried out too late (when utilities improved following the time intervals, but the improvement was too late in order to prevent an SLA violation) or even unnecessary. Thus, the TTs, which tell us when to engage the CBR mechanism, can be constantly ameliorated as well.

Further thoughts on the base concept lead to the following variations:

a) Instead of using rules with TTs, CBR continuously receives new cases by the measurement device. Thus, CBR is not triggered due to TTs, but constantly active. This way we can get rid of TTs, which is especially useful in the early stage when the system does not have historical measurements.

b) As depicted in Figure 4.2, we divide the system status into (1) a manual phase, where we create or adapt cases manually, (2) an active CBR phase as usual, and (3) a passive rule-based phase, where we only do something, if the TT is attained, which we learned in the active phase. When in phase 3, we also calculate utilities of our actions as in phase 2. If the

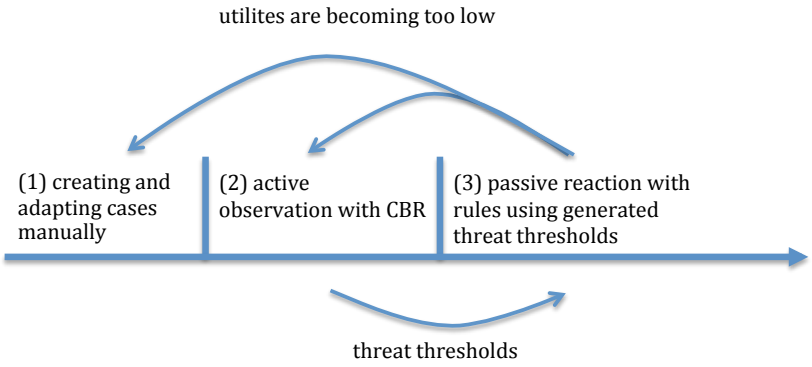


Figure 4.2: Active and Passive phases in the CBR management

utilities get too low, depending on the severity, we either reactivate the active phase (phase 2) to learn new cases or even go into the manual phase (phase 1). When utilities ameliorate, we finally go back to the passive phase (phase 3).

c) For simple parameters (parameters whose causes are easy to understand and model), we have simple TTs and actions using rules instead of using CBR, which helps to relieve computing resources.

Preliminary Implementation of CBR

This section describes implementation details of CBR and methods we used for learning and reacting, as well as the utility measurements employed. The implementation follows variation (a) of the previous section.

We implemented the testbed in Java, based on FreeCBR [6], a generic implementation of step (1) of the list explaining CBR, i.e., "retrieving the most similar case or cases to the new one". As can be seen in Figure 4.3 a complete case consists of: (a) the id of the application being concerned (i.e., instance ID) (line 2), (b) the initial case (measurements by the monitoring component) consisting of the SLO values of the application and global Cloud information like number of running virtual machines (lines 3 – 10), (c) the executed action (line 11), (d) the resulting case (measured some time interval later as in (b) (lines 12 – 19), and (e) the resulting utility (line 20).

To evaluate the actual utility a specific action helped in a specific case, we compare the utility of the initial case to the utility of the resulting case. Let α_{old} and α_{new} be the actual values of some parameter α measured at the initial and the resulting case, respectively, and α_T the specified SLO value. We define the relative utility for a parameter α, whose SLO is $\alpha \geq \alpha_T$. In case the SLO might be $\alpha \leq \alpha_T$, the definition has to be multiplied by -1. We define utility $u(\alpha)$ for $\alpha_T \neq 0$ as

$$u(\alpha) = \frac{\alpha - \alpha_T}{\alpha_T}. \tag{4.2}$$

```
1.  (
2.   (SLA,  1),
3.   (
4.    ((Incoming Bandwidth, 12.0),
5.    (Outgoing Bandwidth, 20.0),
6.    (Storage, 1200),
7.    (Availability, 99.5),
8.    (Running on PMs, 1)),
9.    (Physical Machines, 20)
10.  ),
11.   "Increase Incoming Bandwidth share by 5%",
12.  (
13.   ((Incoming Bandwidth, 12.6),
14.   (Outgoing Bandwidth, 20.1),
15.   (Storage, 1198),
16.   (Availability, 99.5),
17.   (Running on PMs, 1)),
18.   (Physical Machines, 20)
19.   ),
20.  0.002
21. )
```

Figure 4.3: CBR case example

The gain in (or maybe loss of) utility from the initial to the resulting case for a parameter α can be described as

$$u(\alpha_{old}, \alpha_{new}) = \frac{\alpha_{new} - \alpha_T}{\alpha_T} - \frac{\alpha_{old} - \alpha_T}{\alpha_T} = \frac{\alpha_{new} - \alpha_{old}}{\alpha_T}. \tag{4.3}$$

As a next step we have to define the utility for parameters not stated in the SLA of the application, like "running on PMs" or the global parameter "Physical Machines". Considering our use case we define that the fewer PMs the application runs on, the better it is, since this frees up resources for other applications. The same is true for the impact of the number of running physical machines. Shutting down every physical machine that is not needed to guarantee the SLAs is seen as a positive effect on our utility. Thus, we also compare the number of running PMs from the resulting to the initial case with $u(PMs_{old}, PMs_{new}) = \frac{PMs_{old} - PMs_{new}}{PMs_{old}}$. The same principle is true for "running on PMs".

We now derive the final utility by taking the average of the utilities $u(\alpha_{old}, \alpha_{new})$ for all SLA parameters α, of the utilities of running PMs, and of the global parameters. Of course, one could also take into consideration building a weighted average. Generally speaking, there may be more sophisticated methods to define utilities than this linear approach, but for simplicity we decided to start with this one.

For our complete case depicted in Figure 4.3 and the SLA from our use case in Table 2.1,

the utility is thus calculated as follows:

$$u(case) = \frac{(\frac{12.6-12.0}{10.0} + \frac{20.1-20.0}{12.0} + \frac{1198-1200}{1024} + \frac{99.5-99.5}{99.0}) + \frac{1-1}{1} + \frac{20-20}{20.0}}{6} \approx 0.011 \quad (4.4)$$

The similarity of the cases is evaluated by the Euclidean distance, which for two cases takes the square root of the sum of the squared differences of each of the parameters. Of course, as for the utility, one could also weigh these parameters, which we chose to renounce for the beginning.

Furthermore, for the retrieval of similar cases, we implemented two methods. Each method seeks some cases, from which it chooses the one with the highest utility. The first method, which we call *t-neighborhood method*, looks for the case with the highest match percentage and takes all cases into consideration that have a distance of $t\%$ to the case with the highest match percentage. The second method, the *clustering method*, uses a k-means clustering algorithm [94] to group the cases into k clusters, from which we choose the one that includes the case with the highest match percentage. We try the clustering for several ks, and finally choose the k that has the lowest variance among all clusters.

Preliminary Evaluation

In this section we compare the outcomes of the test case using CBR with what we had expected a rational administrator to do. Thus, e.g., if storage for an application is extremely scarce, but all other values are in normal range, we expect the administrator to add allocated storage by the highest possible percentage – we will refer to this as the *intensity* of an action –, and not to increase any other parameter, do nothing or even decrease storage.

After feeding the knowledge base with 9 different cases, we test it against the SLA defined in our use case with 6 new cases and evaluate the results. The initial cases are displayed in Table 4.2, where each column holds one of the cases 1-9. The upper part of the Table (parameters with subscript *b*), shows values as they were measured *before* any action took place. The row *Action* indicates the triggered actions in the specific cases followed by the measured parameters *after* the suggested action (parameters with subscript a). The Row *Utility* shows the utilities gained by these actions.

The six test cases that are stored one after the other into the knowledge base are presented in Table 4.3. The columns depict the cases 1-6, whereas the rows show the parameters at the beginning of the CBR cycle.

The result, i.e., what action was triggered, can be seen in Tables 4.4 and 4.5 for the *clustering* and the *neighborhood method*, respectively. In Table 4.4, the *expected action* column shows what action one could expect to be triggered in the test case (the same column is valid for Table 4.5 and is not repeated therein). The *recommended action* columns in Tables 4.4 and 4.5 define which action was actually recommended by the CBR mechanism. The variance column of Table 4.4 gives us an insight on how compact these clusters are. A low variance signifies high coherence (the points of one cluster have a small distance to each other), whereas high variance signifies the opposite. Additionally, in Table 4.5, where we present results for $t = 3\%$ and $t = 5\%$, we show the number of cases in the t-neighborhood of the case with the highest match percentage. This shows how large the set of cases was to choose the one with the highest utility. The more

	1	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**
IB_b	15.0	11.0	10.5	15.0	15.0	15.0	15.0	15.0	15.0
OB_b	20.0	20.0	20.0	13.0	12.5	20.0	20.0	20.0	20.0
St_b	1200	1200	1200	1200	1200	1050	1000	1000	1200
Av_b	99.5	99.5	99.5	99.5	99.5	99.5	99.5	99.45	99.4
$RPMs_b$	1	1	1	1	1	1	1	1	1
PMs_b	20	20	20	20	20	20	20	20	20
Action	Do nothing.	IBW + 5%	IBW + 10%	OBW + 5%	OBW + 10%	St + 5%	St + 10%	M + 5%	M + 10%
IB_a	15.0	11.55	11.55	15.0	15.0	15.0	15.0	15.0	15.0
OB_a	20.0	20.0	20.0	13.65	13.75	20.0	20.0	20.0	20.0
St_a	1200	1200	1200	1200	1200	1103	1100	1200	1200
Av_a	99.5	99.5	99.5	99.5	99.5	99.5	99.5	99.5	99.5
$RPMs_a$	1	1	1	1	1	1	1	1	1
PMs_a	20	20	20	20	20	20	20	20	20
Utility	0.0	0.009	0.0175	0.009	0.017	0.009	0.016	$8.41 \cdot 10^{-5}$	$1.68 \cdot 10^{-4}$

Table 4.2: Initial Cases for CBR

	1	**2**	**3**	**4**	**5**	**6**
IB	15.0	11.0	10.5	15.0	20.0	10.0
OB	20.0	20.0	20.0	13.0	25.0	18.0
St	1200	1200	1200	1200	1250	1450
Av	99.5	99.5	99.5	99.5	99.6	99.5
$RPMs$	1	1	1	1	1	1
PMs	20	20	20	20	20	20

Table 4.3: Test cases for CBR

cases there are, the higher the chance to catch a case with a higher utility, but at the same time the smaller the similarity is to the original one.

Based on the evaluation results presented in Table 4.4 and 4.5 we conclude that the actions are pretty much the same for both algorithms and relate to the expected action. Only the intensity of the action is always higher than one would expect to be necessary, because greater improvements always have higher utility values (cf. Equation (4.3)). This could be ameliorated by modifying the utility function to allow for *more moderate* actions to have higher utilities. Nevertheless, the problematic SLA parameter, i.e., the parameter whose resources were scarce, is always identified correctly. With the exception of case 5, which has excellent SLA parameter values and does not require any action to be executed, all methods recommend an action to trigger except the neighborhood method for $t = 3\%$. This is explained by the same argument why higher intensities have always been chosen: Doing more than is necessary always achieves a higher utility than doing less or nothing. Thus, the value of doing nothing could also

Case	Expected Action	Recommended Action	Variance
1	IBW + 5%	IBW + 10%	23
2	OBW + 5%	OBW +10%	18
3	St + 5%	St + 10%	208
4	St + 10%	St + 10%	14
5	None	St + 10%	96
6	IBW + 10%	IBW + 10%	40

Table 4.4: Evaluation results using the clustering algorithm

	$\mathbf{t = 5\%}$		$\mathbf{t = 3\%}$	
Case	**Recomm. Action**	**Cases in t-neighborhood**	**Recomm. Action**	**Cases in t-neighborhood**
1	IBW + 10%	2	IBW + 10%	2
2	OBW + 10%	4	OBW +10%	4
3	St + 10%	2	St + 10%	2
4	St + 10%	3	St + 10%	2
5	M + 5%	2	None	1
6	IBW + 10%	8	IBW + 10%	5

Table 4.5: Evaluation results using the neighborhood algorithm

be appreciated more in the definition of the utilities.

Preliminary Conclusion

In this section we discussed several approaches for knowledge management in self-adaptable Clouds, which were rule-based systems, default logic, situation calculus, and case based reasoning. We adopted the *case based reasoning (CBR)* method for the interpretation of measurement data with the goal of preventing SLA violations by triggering appropriate actions. Additionally, we designed a CBR based mechanism for the automatic re-configuration or even avoidance of threat thresholds topped with the introduction of general utility functions, which we were able to design without any semantic knowledge of the SLA parameters.

Currently, the CBR approach has been evaluated only against one SLA. A big issue, however, is that concurring SLAs may prevent other applications from being executed, especially if resources are scarce. Also, we have only used predefined SLA parameters, which could be extended to generate user defined SLA parameters including the development of appropriate DSLs. Furthermore, we want to validate this approach by generating an extensive simulation model of a cloud environment over several time steps - using that, we will be able to evaluate not only CBR, but also other knowledge management methods from a hands-on point of view.

Nevertheless, we provided a means of proactively gearing the cloud infrastructure against SLA violations regardless of the SLA parameters in use. We have presented the proof of concept for the realization of the CBR-based knowledge management systems for self-adaptable Clouds.

4.2 Speculative Approach

After the preliminary evaluation and conclusion, this section subsumes all the common assumptions for both approaches that will be presented next in more detail: CBR and the (self-adpating) rule-based approach.

We assume that customers deploy applications on an IaaS Cloud infrastructure. SLOs are defined within an SLA between the customer and the Cloud provider for every application. Furthermore, there is a 1:1 relationship between applications and VMs. One VM runs on exactly one PM, but one PM can host an arbitrary number of VMs with respect to supplied vs. demanded resource capacities. After allocating VMs with an initial capacity (by estimating initial resource demand) for every application, we continuously monitor actually used resources and re-allocate resources according to these measurements.

Provided (1)	Utilized (2)	Agreed (3)	Violation?
500 GB	400 GB	$\geq 1000GB$	NO
500 GB	510 GB	$\geq 1000GB$	YES
1000 GB	1010 GB	$\geq 1000GB$	NO

Table 4.6: Cases of (non-) SLA violations using the example of storage

For tackling the resource allocation for VMs, we need to define how measured, provided and agreed values interrelate, and what actually constitutes an SLA violation. An example is provided in Table 1. At first, we deal with the measured value (1), which represents the amount of a specific resource that is currently used by the customer. Second, there is the amount of allocated resource (2) that can be used by the customer, i.e., that is allocated to the VM which hosts the application. Third, there is the SLO agreed in the SLA (3). A violation therefore occurs, if less is provided (2) than the customer utilizes (or wants to utilize) (1) with respect to the limits set in the SLA (3). Considering Table 1 we can see that rows 1 and 3 do not represent violations, whereas row 2 does represent an SLA violation. In order to save resources we envision a **speculative approach**: Can we allocate less than agreed, but still more than used in order not to violate an SLA? The most demanding questions are how much can we lower the provisioning of a resource without risking an SLA violation. This heavily depends on the characteristics of the workload of an application, especially its volatility.

4.3 Case Based Reasoning

After explaining CBR in Section 4.1 and taking the preliminary evaluation into account, three issues have to be solved to better adapt CBR to our problem. First, it has to be decided how to format an instance of the problem. Second, it has to be decided when two cases are similar. Third, good reactions have to be distinguished from bad reactions.

As to the first problem we assume that each SLA has a unique identifier id and a collection of SLOs. SLOs are predicates of the form

$$SLO_{id}(x_i, comp, \pi_i) \text{ with } comp \in \{<, \leq, >, \geq, =\}, \tag{4.5}$$

where $x_i \in P$ represents the parameter name for $i = 1, \ldots, n_{id}$, π_i the parameter goal, and $comp$ the appropriate comparison operator. Then, a CBR case c is defined as

$$c = (id, m_1, p_1, m_2, p_2, \ldots, m_{n_{id}}, p_{n_{id}}), \tag{4.6}$$

where id represents the SLA id, and m_i and p_i the measured (m) and provided (p) value of the SLA parameter x_i, respectively.

To use the SLA parameters *storage* and *incoming bandwidth* for example, a typical use case looks like this: SLA $id = 1$ with SLO_1("Storage", $\geq$, 1000) and SLO_1 ("Bandwidth", $\geq$, 50.0). A corresponding case received by the measurement component is therefore written as $c = (1, 500, 700, 20.0, 30.0)$. A result case $rc = (c^-, ac, c^+, utility)$ includes the initial case c^-, the executed action ac, the resulting case c^+ measured some time interval later, which corresponds to one iteration in the simulation engine, and the calculated $utility$ described later. In order to give the KB some knowledge about what to do in specific situations, several initial cases are stored in the KB as described in [150] in more detail.

Secondly, to define similarity between two cases is not straightforward, because due to their symmetric nature Euclidean distances, e.g., do not recognize the difference between over- and under-provisioning. Following the principle of semantic similarity from [96] for the summation part this leads to the following equation

$$d(c^-, c^+) = \min(w_{id}, \left|id^- - id^+\right|) + \sum_{x \in P} w_x \left| \frac{(p_x^- - m_x^-) - (p_x^+ - m_x^+)}{max_x - min_x} \right|, \tag{4.7}$$

where $w = (w_{id}, w_{x_1}, \ldots, w_{x_n})$ is the weight vector; w_{id} is the weight for non-identical SLAs; w_x is the weight, and max_x and min_x the maximum and minimum values of differences $p_x - m_x$ for parameter x.

As far as the third issue is concerned, every action is evaluated by its impact on violations and utilization. This way CBR is able to learn whether an action was appropriate for a specific measurement or not. The utility of an action is calculated by comparing the initial case c^- with the resulting final case c^+. The *utility function* is composed by a violation and a utilization term weighed by the factor $0 \leq \alpha \leq 1$:

$$utility = \sum_{x \in P} violation(x) + \alpha \cdot utilization(x) \tag{4.8}$$

Higher values for α strengthen the utilization of resources, whereas lower values strengthen the non-violation of SLA parameters. We further note that $c(x)$ describes a case only with respect to parameter x. E.g., we say that a violation has occurred in $c(x)$, when in case c the parameter x was violated.

We define the $violation$ function for every parameter x as follows:

$$violation(x) = \begin{cases} 1, & \text{No violation occurred in } c^+(x), \text{ but in } c^-(x) \\ 1/2, & \text{No violation occurred in } c^+(x) \text{ and } c^-(x) \\ -1/2 & \text{Violation occurred in } c^+(x) \text{ and } c^-(x) \\ -1 & \text{Violation occurred in } c^+(x), \text{ but not in } c^-(x) \end{cases} \tag{4.9}$$

The $utilization$ function is calculated by comparing the used resources to the provided ones. We define the distance $\delta(x, y) = |x - y|$, and utilization for every parameter as

$$utilization(x) = \begin{cases} 1, & \delta(p_x^-, m_x^-) > \delta(p_x^+, u_x^+) \\ -1, & \delta(p_x^-, m_x^-) < \delta(p_x^+, u_x^+) \\ 0, & \text{otherwise.} \end{cases} \tag{4.10}$$

A utilization utility of 1 is retrieved if less over-provisioning of resources takes place in the final case than in the initial one, and a utilization utility of -1 if more over-provisioning of resources takes place in the final case than in the initial one.

The whole CBR process works as follows: Before the first iteration, we store the mentioned initial cases consisting of an initial measurement, an action and a resulting measurement. Then, when CBR receives a new measurement, this measurement is compared to all cases in the KB. From the set of closest cases grouped by a clustering algorithm we choose the one with the highest utility and execute exactly the same action as in the chosen case. Afterwards, this action, the resulting measurement and the utility of the action is added to the initial measurement, and stored as a complete case.

4.4 Rule-based Approach

Using the escalation levels presented in Section 2.3 we mitigate the problems pointed out in Section 4.1 for a rule-based approach.

For the rule-based approach we first introduce several resource policy modes to reflect the overall utilization of the system in the VM configuration rules. Dealing with SLA-bound resource management, where resource usage is paid for on a "pay-as-you-go" basis with SLOs that guarantee a minimum capacity of these resources as described above, raises the question, whether the Cloud provider should allow the consumer to use more resources than agreed. We will refer to this behavior as *over-consumption*. Since the consumer will pay for every additional resource, it should be in the Cloud provider's interest to allow over-consumption as long as this behavior does not endanger the SLAs of other consumers. Thus, Cloud providers should not allow over-consumption when the resulting penalties they have to pay are higher than the expected revenue from over-consumption. To tackle this problem, we introduce five policy modes for every resource that describe the interaction of the five escalation levels. As can be seen in Table 4.7 the policy modes are green, green-orange, orange, orange-red and red. They range from low utilization of the system with lots of free resources left (policy mode green) over a scarce resource situation (policy mode orange) to an extremely tight resource situation (policy mode red), where it is impossible to fulfill all SLAs to its full extent and decisions have to be made which SLAs to deliberately break and which applications to outsource.

In order to know whether a resource r is in danger of under-provisioning or already is under-provisioned, or whether it is over-provisioned, we calculate the current utilization $ut^r = \frac{\text{use}^r}{\text{pr}^r} \times 100$, where use^r and pr^r signify how much of a resource r was used and provided, respectively, and divide the percentage range into three regions using the two "threat thresholds" TT^r_{low} and TT^r_{high}:

green	Plenty of resources left. Over-consumption allowed.
green-orange	Heavy over-consumption is forbidden. All applications that consume more than $\tau\%$ (threshold to be specified) of the agreed resource SLO are restrained to $\tau/2\%$ over-consumption
orange	Resource is becoming scarce, but SLA demand can be fulfilled if no over-consumption takes place. Thus, over-provisioning is forbidden.
orange-red	Over-provisioning forbidden. Initiate outsourcing of some applications.
red	Over-provisioning forbidden. SLA resource requirements of all consumers cannot be fulfilled. If possible, a specific choice of applications is outsourced. If not enough, applications with higher reputation points or penalties are given priority over applications with lower reputation points/penalties. SLAs of latter ones are deliberately broken to ensure SLAs of former ones.

Table 4.7: Resource policy modes

- Region -1: Danger of under-provisioning, or under-provisioning ($> TT^r_{high}$)
- Region 0: Well provisioned ($\leq TT^r_{high}$ and $\geq TT^r_{low}$)
- Region $+1$: Over-Provisioning ($< TT^r_{low}$)

The idea of this rule-based design is that the ideal value that we call *target value* $tv(r)$ for utilization of a resource r is exactly in the center of region 0. So, if the utilization value after some measurement leaves this region by using more (Region -1) or less resources (Region +1), then we reset the utilization to the target value, i.e., we increase or decrease allocated resources so that the utilization is again at

$$tv(r) = \frac{TT^r_{low} + TT^r_{high}}{2}\%.$$

As long as the utilization value stays in region 0, no action will be executed. E.g., for $r =$ storage, $TT^r_{low} = 60\%$, and $TT^r_{high} = 80\%$, the target value would be $tv(r) = 70\%$. Figure 4.4 shows the regions and measurements (expressed as utilization of a certain resource) at time steps $t_1, t_2, \ldots, t_6$. At t_1 the utilization of the resource is in Region -1, because it is in danger of a violation. Thus, the KB recommends to increase the resource such that at the next iteration t_2 the utilization is at the center of Region 0, which equals the target value. At time steps t_3 and t_4 utilization stays in the center region and consequently, no action is required. At t_5, the resource is under-utilized and so the KB recommends the decrease of the resource to $tv(r)$, which is attained at t_6. Additionally, if over-provisioning is allowed in the current policy mode, then the adjustment will always be executed as described regardless of what limit was agreed in the SLA. On the other hand, if over-provisioning is not allowed in the current policy mode, then the rule will allocate at most as much as agreed in the SLA (SLO^r).

The concept of a rule increasing resource r is depicted in Figure 4.5. The rule executes if the current utilization ut^r and the predicted utilization $ut^r_{predicted}$ of the next iteration (cf. next paragraph) both exceed TT^r_{high} (line 2). Depending on what policy level is active the rule either sets the provided resource pr^r to the target value $tv(r)$ for policy levels green and green-orange (line 3) or to at most what was agreed in the SLA (SLO^r) plus a certain percentage ϵ to account for rounding errors when calculating the target value in policy levels orange, orange-red and red

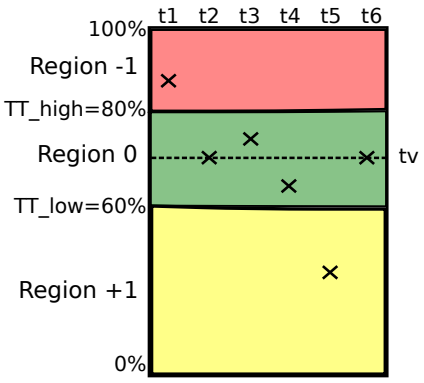


Figure 4.4: Example behavior of actions at time intervals t1-t6

(line 5). A similar rule scheme for decreasing a resource can be seen in Figure 4.6. The main difference is that it does not distinguish between policy modes and that it sets the provisioned resource to at least a minimum value $minPr^r$, which may be 0, that is needed to keep the application alive (line 4). The rule is executed if the current utilization ut^r and the predicted utilization $ut^r_{predicted}$ of the next iteration both lie below TT^r_{low} (line 2).

A large enough span between the thresholds TT^r_{low} and TT^r_{high} helps to prevent oscillations of repeatedly increasing and decreasing the same resource. However, to further reduce the risk of oscillations, we suggest to calculate a prediction for the next value based on the latest measurements. Thus, an action is only invoked when the current AND the predicted measurement exceed the respective TT. So, especially when only one value exceeds the TT, no action is executed.

1 **IF**
2 $ut^r > TT^r_{\text{high}}$ AND $ut^r_{predicted} > TT^r_{\text{high}}$
3 **THEN**
4 Set pr^r to $\frac{\text{use}^r}{tv(r)}$ for policy modes green, green-orange.
5 Set pr^r to $\min(\frac{\text{use}^r}{tv(r)}, SLO^r * (1 + \epsilon/100))$ for policy modes orange, orange-red, red.

Figure 4.5: Rule scheme for increasing a resource

1 **IF**
2 $ut^r < TT^r_{\text{low}}$ AND $ut^r_{predicted} < TT^r_{\text{low}}$
3 **THEN**
4 Set pr^r to $\max(\frac{\text{use}^r}{tv(r)}, minPr^r)$.

Figure 4.6: Rule scheme for decreasing a resource

The rules have been implemented using the Java rule engine Drools [15]. The Drools engine sets up a knowledge session consisting of different rules and a working memory. Rules get

activated when specific elements are inserted into the working memory such that the conditional "when" part evaluates to true. Activated rules are then triggered by the simulation engine. In our case, the simulation engine inserts measurements and SLAs of applications into the working memory. Different policy modes will load slightly modified rules into the Drools engine and thus achieve a high adaptability of the KM system reacting to the general performance of the Cloud infrastructure. As opposed to the CBR based approach in [150], the rule-based approach is able to fire more than one action at the same iteration, which inherently increases the flexibility of the system. Without loss of generality we can assume that one application runs on one VM (several applications' SLAs can be aggregated to form one VM SLA) and we assume the more interesting case of policy modes orange, orange-red or red, where over-provisioning is not allowed.

Listing 4.1 shows the rule to increase parameter storage formulated in the Drools language following the pattern presented in Figure 4.5. Line 1 defines the name of the rule that is split into a condition part (**when**, lines 2-12) and an execution part (**then**, lines 13-17). Line 4 tries to find the SLA of an application, stores its id into `$slaID` and the SLA into `$slaApp`. Line 6 looks for a set of actions for this `$slaID` where no storage action has been added yet (`storage == false`) in order to avoid contradicting actions for storage for one measurement. Line 8 searches for a measurement for the appropriate VM (`vmID == $slaID`) that has been inserted into working memory that is no prediction (`$prediction == false`) and where the percentage of utilized storage exceeds TT^r_{high}, i.e.,

```
storage_utilized > storage_HighTT,
```

and stores used and provided values into `$s_used` and `$s_provided`, respectively. The predicted measurement for the next iteration is handled similarly in line 10. Finally, line 12 checks whether provided storage is still below the agreed value in the SLA. This is done, because in policy modes orange to red over-consumption is prohibited. The rules for policy modes green and green-orange would omit this line. Now, if all these conditions are met, the rule gets activated. When fired, line 15 calculates the new value for pr^r as explained in Figure 4.5. This line (as line 12) would also be altered for policy modes green and green-orange. Line 17 then modifies the action container `$as` and inserts the appropriate storage action with the value for provided storage to be set. Other rules follow the same pattern as described here and in Figure 4.5 for rules increasing resource allocations and in Figure 4.6 for rules decreasing resource allocations.

Listing 4.1: Rule "storage_increase"

```
1  rule "storage_increase"
2  when
3   //Remember SLA id of application
4   $SLA_app : Application($slaID : id)
5   //Look for set of actions that has no storage action yet
6   $as : Actions(slaID == $slaID, storage == false)
7   //Look for measurement that has high utilization of storage
8   $m : Measurement (prediction == false, storage_utilized > storage_HighTT,
        vmID == $slaID, $s_used: storage_used, $s_provided: storage_provided)
9   //Look for predicted measurement that will have high utilization of storage
10  $m_pred : Measurement (prediction == true, storage_utilized > storage_HighTT
        , vmID == $slaID)
```

```
11   //Check whether we provide less than SLO value
12   eval($s_provided <= Double.valueOf( $SLA_app.getThresholdByName("storage")))
13  then
14   //Calculate tv
15   double newStorage = Math.min($s_used/((storage_HighTT+ storage_LowTT)/2),
         Double.valueOf (SLA_app.getThresholdByName("storage"))* (1+eps/100));
16   //Add storage action to set of actions
17   modify ($as) addAction(new StorageActionDirect(newStorage, "GB")),
         setStorage();
18  end
```

4.5 Self-adapting the Rule-based Approach

As will be seen in Section 6.4 the TTs of the rule-based approach have a high impact on its performance. This section will explain how the autonomic adaptation and configuration of the autonomic manager, i.e., the TT adaptation, works. We will describe two basically different approaches: The first approach is based on changes within a cost function, whereas the second one relies on changes in the workload.

Approaches based on the cost function

In this approach the autonomic adaptation of the TTs is based on the definition of the cost function in [151]. The general idea is that if cost has increased for some time, TTs should be adapted. If this is the case two different subproblems have to be solved:

1. Determine the most appropriate TT(s) to adapt.

2. Determine for how much the chosen TT(s) should be adapted.

The cost function sums up costs incurred while enacting an SLA on a VM. These costs consist of SLA penalties, resource wastage, and the VM reconfiguration actions. The used cost function is defined as

$$c(p, w, c) = \sum_r \mathfrak{p}^r(p^r) + \mathfrak{w}^r(w^r) + \mathfrak{a}^r(a^r), \tag{4.11}$$

where, for a certain resource r, $\mathfrak{p}^r(p^r) : [0, 100] \rightarrow \mathbb{R}^+$ defines the costs due to the penalties that have to be paid according to the relative number of SLA violations (as compared to all possible SLA violations) p^r; $\mathfrak{w}^r(w^r) : [0, 100] \rightarrow \mathbb{R}^+$ defines the costs due to unutilized resources w^r; and $\mathfrak{a}^r(a^r) : [0, 100] \rightarrow \mathbb{R}^+$ the costs due to the executed number of actions a^r (as compared to the number of all possible actions).

During the *Analysis* phase the KB does not only observe the cost for one resource r, which naturally is defined as $c^r(p, w, c) = \mathfrak{p}^r(p^r) + \mathfrak{w}^r(w^r) + \mathfrak{a}^r(a^r)$, but also each individual component $\mathfrak{p}^r$, $\mathfrak{w}^r$, and $\mathfrak{a}^r$ for each resource. If the cost has increased for a resource over a certain period of time (called *look-back horizon* k and defined later in this section), the KB starts to investigate which of the components caused this increase.

Subproblem 1 (Selecting TTs). To solve subproblem 1, at first the most problematic cost factor has to be determined. From this, we can relate to a specific TT increase/decrease action. To achieve this one can basically imagine two different methodologies: Either, the maximum cost parameter of the current iteration, or the parameter with the maximum increase in the last k iterations is chosen.

Since our cost function c^r works by relative and not total costs, the first method would yield the following problem: Suppose that no violation has occurred for 10 iterations. Thus, $p^r = 0$ at iteration 10. At iteration 11, though, a violation occurs which makes $p^r = 1/11$. In the following iterations, where $p^r = 1/12, 1/13, 1/14, \ldots$ (if no further violations occurs) $\mathfrak{p}^r$ could be easily greater than $\mathfrak{w}^r$ and $\mathfrak{a}^r$ as violations are usually punished more severely than wastage or actions. Thus, for these iterations the algorithm would always decide to act based on violations, even though violations are not occurring any more in the same time.

Let $p^{r,t}$ signify the relative amount of violations at iteration t, and let $w^{r,t}$ $a^{r,t}$ be defined similarly. Then, since an increase in, e.g., violations $p^{r,t}$ occurs iff $\mathfrak{p}^{r,t}$ is strongly monotonically increasing, we choose to opt for the second methodology. According to a look-back horizon k we calculate the difference between the current cost and the minimum cost of the last k iterations. The maximum of these differences then points to the cost summand (arg) that needs attention:

$$\begin{aligned}\arg\max(&\mathfrak{p}^{r,t} - \min_{1\leq j\leq k}(\mathfrak{p}^{r,t-j}), \mathfrak{w}^{r,t} - \min_{1\leq j\leq k}(\mathfrak{w}^{r,t-j}),\\ &\mathfrak{a}^{r,t} - \min_{1\leq j\leq k}(\mathfrak{a}^{r,t-j})).\end{aligned} \tag{4.12}$$

This results into three different cases, where either the $\mathfrak{p}$, $\mathfrak{w}$, or $\mathfrak{a}$ terms yield the maximum. (We omit cases where some arguments of the maximum function are equal. In such a case, the order to choose the arg max is $\mathfrak{p}$ over $\mathfrak{w}$ over $\mathfrak{a}$. We prioritize like this, because we assume that penalties incur higher costs than wastage, and wastage incurs higher costs than reconfiguration actions.) We define three options which TT(s) to increase or decrease.

- *Option A:*
  1. $\mathfrak{p}^{r,t} - \min_{1\leq j\leq k}(\mathfrak{p}^{r,t-j})$ is maximal: Decrease TT^r_{high} and TT^r_{low}.
  2. $\mathfrak{w}^{r,t} - \min_{1\leq j\leq k}(\mathfrak{w}^{r,t-j})$ is maximal: Increase TT^r_{low}.
  3. $\mathfrak{a}^{r,t} - \min_{1\leq j\leq k}(\mathfrak{a}^{r,t-j})$ is maximal: Decrease TT^r_{low} and increase TT^r_{high}.
- *Option B:*
  1. $\mathfrak{p}^{r,t} - \min_{1\leq j\leq k}(\mathfrak{p}^{r,t-j})$ is maximal: Decrease TT^r_{high} and TT^r_{low}.
  2. $\mathfrak{w}^{r,t} - \min_{1\leq j\leq k}(\mathfrak{w}^{r,t-j})$ is maximal: Increase TT^r_{high} and TT^r_{low}.
  3. $\mathfrak{a}^{r,t} - \min_{1\leq j\leq k}(\mathfrak{a}^{r,t-j})$ is maximal: Decrease TT^r_{low} and increase TT^r_{high}.
- *Option C:*
  1. $\mathfrak{p}^{r,t} - \min_{1\leq j\leq k}(\mathfrak{p}^{r,t-j})$ is maximal: Decrease TT^r_{high}.
  2. $\mathfrak{w}^{r,t} - \min_{1\leq j\leq k}(\mathfrak{w}^{r,t-j})$ is maximal: Increase TT^r_{low}.

3. $\mathfrak{a}^{r,t} - \min_{1 \leq j \leq k}(\mathfrak{a}^{r,t-j})$ is maximal: Decrease TT^r_{low} and increase TT^r_{high}.

The difference between options A and B is that if the $\mathfrak{w}$ term causes the maximum, it will increase both low and high TTs in option B, whereas it will only increase TT_{low} in option A. The main feature of option C is that it only decreases TT_{high} (instead of also decreasing TT_{low}). So option B and even more option A could be seen as more cautious as far penalties for SLA violations are concerned than option C.

Moreover, we present a fourth methodology, *option D*, differing from the former three ones. This methodology does not only consider the maximum cost summand increase, but handles all cost parameters that show an increase, but only for the recent iteration. This may promise that the actual situation of which parameter needs to be adapted is assessed more precisely. Thus, one can distinguish seven different cases:

1. $\mathfrak{p}^r$ increased: Decrease TT^r_{high}.
2. $\mathfrak{w}^r$ increased: Increase TT^r_{low}.
3. $\mathfrak{a}^r$ increased: Decrease TT^r_{low}, increase TT^r_{high}.
4. $\mathfrak{p}^r$ and $\mathfrak{w}^r$ increased: Increase TT^r_{low}, decrease TT^r_{high}.
5. $\mathfrak{p}^r$ and $\mathfrak{a}^r$ increased: Decrease TT^r_{low}.
6. $\mathfrak{w}^r$ and $\mathfrak{a}^r$ increased: Increase TT^r_{high}.
7. $\mathfrak{p}^r$ and $\mathfrak{w}^r$ and $\mathfrak{a}^r$ increased: Choose the two factors with the highest increase and act according to the cases 4-6.

Subproblem 2 (Adapting TTs). After subproblem 1 has been solved, for subproblem 2 it is important to determine the value by how much the respective TT(s) should be moved. Again, one could imagine several techniques to determine a good value for the TTs as case based reasoning (adapting the approach as described in [150]), or using fixed or random increasing/decreasing steps. Observing that for the TTs the following inequalities must hold

$$0\% < TT_{low} < TT_{high} < 100\%, \tag{4.13}$$

we choose to use the following approach. If we need to decrease TT_{low} or increase TT_{high}, we set it to a certain fraction $1/\alpha < 1$ of the distance from TT_{low} to 0, and from TT_{high} to 100, respectively, expressed as

$$TT^{r,t+1}_{low} = TT^{r,t}_{low} - \frac{TT^{r,t}_{low}}{\alpha} \tag{4.14}$$

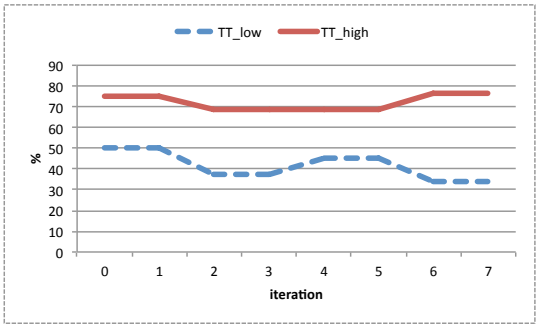
$$TT^{r,t+1}_{high} = TT^{r,t}_{high} + \frac{100 - TT^{r,t}_{high}}{\alpha}. \tag{4.15}$$

(Superindex t indicates the time iteration for which the respective TT is valid. It is omitted, if not relevant.) If we need to increase TT_{low} or decrease TT_{high}, we shrink the distance d between TT_{low} and TT_{high} to $\frac{d(\alpha-1)}{\alpha}$ by moving the TT in question towards the other, i.e.,

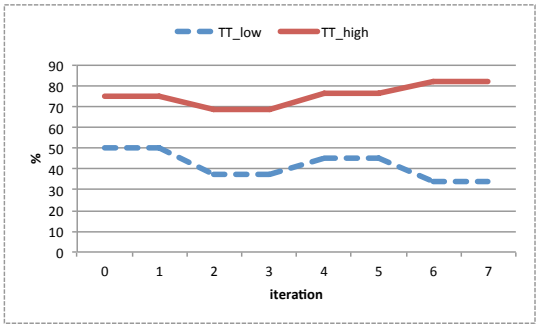
$$TT_{low}^{r,t+1} = TT_{low}^{r,t} + \frac{TT_{high}^{r,t} - TT_{low}^{r,t}}{\alpha} \tag{4.16}$$

$$TT_{high}^{r,t+1} = TT_{high}^{r,t} - \frac{TT_{high}^{r,t} - TT_{low}^{r,t}}{\alpha}. \tag{4.17}$$

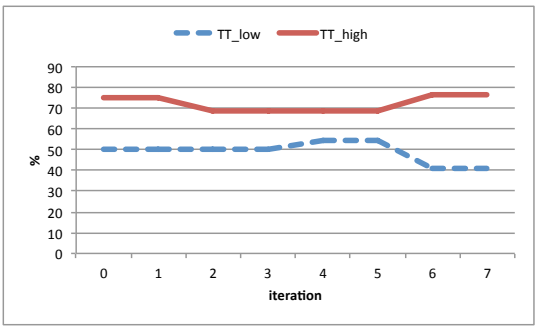
This especially makes sure that Equation (4.13) also holds in this situation. When both TT_{low} and TT_{high} are to be increased and decreased, respectively, simultaneously (cf. case 4 in option D), we have to set $\alpha > 2$ in order not to violate Equation (4.13).



(a) TT example for option A



(b) TT example for option B



(c) TT example for option C

Figure 4.7: TT examples for options A-C

Summarizing both subproblems, the graphs in Figure 4.7 show how the TTs behave for the different options A-C according to the following scenario: All options start with $TT_{low} = 50\%, TT_{high} = 75\%$. At iteration 2 we encounter a maximum in penalties, at iteration 4 a maximum in wastage and at iteration 6 a maximum in actions.

Approach Based on Workload Volatility

As an alternative to the cost function dependent approach, we investigate an approach depending on the change in the workload, i.e., the workload volatility (WV).

We define *workload volatility* ϕ as the intensity of change in the measured workload traces of a certain resource. We calculate this intensity as the percentage relating the current value of the workload $m^{r,t}$ to the previous one $m^{r,t-1}$, i.e.,

$$\phi^{r,t}(m^{r,t}, m^{r,t-1}) = |(\frac{\max(m^{r,t}, r_{min})}{\max(m^{r,t-1}, r_{min})} - 1) \cdot 100|$$

for $t \geq 1$ and $r_{min} > 0$. The variable r_{min} stands for the lower bound for a certain resource stated in the Service Level Objective (SLO). E.g., we have $r_{min} = 10$ for the SLO "$10GB \leq storage \leq 1000GB$". This amount will always be provided, even if an application uses less. So measurements below this value should not influence the behavior of the system, neither the classification into a WV class. To give an example for $r = \textit{storage}$, let us assume that $m^{r,t} = 20, m^{r,t-1} = 15$. We would get $\phi^{r,t}(m^{r,t}, m^{r,t-1}) = 33.\dot{3}\%$. If at the next iteration we have $m^{r,t+1} = 18$, then $\phi^{r,t+1}(m^{r,t+1}, m^{r,t}) = 10\%$.

This is useful, because a problem inherent in options A-C is that the new parameter k to be tuned is introduced. Its relevance to WV is the following: When WV is low, a long look-back horizon is helpful, because a short one would trigger more TT adaptation situations, which in reality are just insignificant changes in workload. On the opposite, when WV is high, changes can get very fast very significant, and thus a short look-back horizon should be favored.

For this methodology, we introduce WV classes, into which we automatically categorize workload on the fly. We define the following WV classes: LOW, MEDIUM, MEDIUM_HIGH, and HIGH. Algorithm 4.1 dynamically decides to which WV class a specific workload trace belongs. Dynamically means that the classification might change at every iteration, if the workload behavior changes significantly. Significant in this context means that the current value for WV is compared to the recent behavior of the workload. Only if the maximum value for the WV from recent and current behavior falls into a different category, the classification is altered. From the second iteration on, the algorithm first calculates ϕ and determines the maximum value in ϕQ, which is a queue of size $\phi Q\_maxsize$ (lines 2-7). The method $addLast()$ adds the input element as last element to the queue, whereas the method $remove()$ removes the first element of the queue. Lines 9-18 classify the workload according to the found maximum element of the queue. An ϵ is added to this comparison in order to hinder small statistical outliers from altering the classification outcome. Table 4.8 summarizes all constants used for the evaluation.

Based on this classification the following two options E and F alter their behavior accordingly. *Option E* chooses a "good" set of TTs from a-priori evaluation for different WV classes. This can be tested offline, and altered if specified in the SLA. E.g., for high-risk applications both TTs could be lowered, whereas for energy-aware applications, the TTs could be increased for all workloads. For our case, Table 4.9 shows the TTs for the mentioned volatility classes. The values were chosen according to the definition of the WV classes in Section 6.1.

Also from a-priori experience, *option F* chooses the best option with its best k according to the best result in the corresponding WV class. As will be seen in Section 6.6, the best results for every WV class can be achieved by the options captured in the right-hand side of Table 4.9.

Input: $r, m^{r,t}, m^{r,t-1}, \phi Q^r$
Output: Workload volatility class

if $t \geq 1$ **then**
  {Calculate ϕ and determine maximum in ϕQ^r}
  $\phi Q^r.addLast(\phi^{r,t}(m^{r,t}, m^{r,t-1}))$
  if $\phi Q^r.size() > \phi Q\_maxsize$ **then**
    $\phi Q^r.remove()$
  end if
  $\phi Q^r_{max} \leftarrow \max(\phi Q^r)$

  {Classify workload volatility}
  if $\phi Q^r_{max} \leq$ LOW_THRESHOLD $+ \epsilon$ **then**
    return LOW
  else if $\phi Q^r_{max} \leq$ MEDIUM_THRESHOLD $+ \epsilon$ **then**
    return MEDIUM
  else if $\phi Q^r_{max} \leq$ MEDIUM_HIGH_THRESHOLD $+ \epsilon$ **then**
    return MEDIUM_HIGH
  else if $\phi Q^r_{max} \leq$ HIGH_THRESHOLD $+ \epsilon$ **then**
    return HIGH
  end if
end if

Algorithm 4.1: On-the-fly Classifying of Workload into its Workload Volatility Class

Parameter	Value
LOW_THRESHOLD	10
MEDIUM_THRESHOLD	50
MEDIUM_HIGH_THRESHOLD	75
HIGH_THRESHOLD	100
$\phi Q\_maxsize$	10
ϵ	4

Table 4.8: Parameters used for Algorithm 4.1

	Option E)		*Option F)*
WV	TT_{low}	TT_{high}	Choose Option
LOW	70%	90%	C), $k = 5$
MEDIUM	45%	70%	A), $k = 20$
MEDIUM_HIGH	30%	60%	A), $k = 5$
HIGH	20%	50%	A), $k = 2$

Table 4.9: A-priori defined TTs and options based on workload volatility classes for options E) and F)

CHAPTER 5

Energy-efficient SLA Enactment in Cloud Computing Infrastructures

This chapter focuses on the energy-efficient aspect of SLA enactment for Cloud computing infrastructures. It presents an energy model, as well as several heuristics, for VM migrations and PM power management. The chapter also formulates and formalizes the *IaaS management problem* and proves it to be an instance of the NP-hard *binary integer programming* problem. It is shown that the NP-hardness is also of relevance for this instance in practice, as a standard heuristic used to solve this problem cannot solve this specific instance in reasonable time even for small Clouds.

5.1 Formalization of the IaaS Management Problem

This section formalizes the Cloud environment together with the IaaS management problem.

We define the set of virtual machines (VMs) as $VM = \{vm_1, \ldots, vm_n\}$ and the set of physical machines (PMs) as $PM = \{pm_1, \ldots, pm_m\}$. For the available resources $Res = \{r_1, \ldots, r_k\}$ we define resource functions

$$r_l^{vm} : VM \to \mathbb{R}_{\geq 0}, \tag{5.1}$$

$$r_l^{pm} : PM \to \mathbb{R}_{\geq 0} \tag{5.2}$$

describing the desired amount of resource l for VMs and the available amount of resource l for PMs, respectively. Of course, for one resource l the range and the units the resource is measured in have to be the same for r_l^{vm} and r_l^{pm}. For the set of resources we may consider, as in the chapters before, $Res = \{\text{storage}, \text{incoming bandwidth}, \text{outgoing bandwidth}, \text{CPU power}, \text{memory}\}$, but this approach is not tied to these parameters in any way. The only assumption made is that such resources can be required by a VM and provided by a PM.

For each time step t we know, whether a VM should be running (1) or not (0), defined by

$$on^{(t)} : VM \to \{0, 1\}. \tag{5.3}$$

What we are looking for is an instance of the function

$$f^{(t)} : VM \to PM \cup \{\emptyset\},$$
$$f^{(t)}(vm) = \begin{cases} pm \in PM & \text{if } on^{(t)}(vm) = 1, \\ \emptyset & \text{if } on^{(t)}(vm) = 0. \end{cases} \tag{5.4}$$

that maps each virtual machine to a physical machine or to the empty set, if it is not yet or no longer deployed, at a specific point of time t. Furthermore, we want to reduce the overall costs, i.e., energy consumption, of our Cloud environment. Thus, we define the cost function

$$c : PM \to \mathbb{R} \tag{5.5}$$

that describes the energy consumed by a running PM. If we want to take the CPU frequency a PM is running at into consideration, we can define the cost function as $c : PM \times$ Frequency class $\to \mathbb{R}$, where the set of frequencies a PM is capable of running at is partitioned into some frequency classes. This is important when we want to fine-tune our PMs, since a PM running at a lower frequency class consumes less energy than a PM running at a higher one. In this thesis we will omit this fine-tuning. When we know $f^{(t)}$, we can find all running PMs at time t by

$$PM^{(t)}_{active} = \{pm_j | \exists i : pm_j = f^{(t)}(vm_i)\}. \tag{5.6}$$

As already stated, we want to minimize energy costs,

$$\text{minimize} \sum_{pm_j \in PM^{(t)}_{active}} c(pm_j), \tag{5.7}$$

while complying to some resource constraints:

$$\forall j \in \{1, \ldots, m\}, \forall l \in \{1, \ldots, k\} : \sum_{\forall i : f(vm_i) = pm_j} r^{vm}_l(vm_i) \leq r^{pm}_l(pm_j). \tag{5.8}$$

Furthermore, we assume that $f^{(t)}$ is a total function. Thus, every VM is deployed on exactly one PM or is shut down. Of course, the function is not injective, because a PM should be able to host more than one VM.

As a next step, we want to integrate the costs of migrating VMs to different PMs or even other clouds, and of booting PMs. As to the first part, we define the migration cost of a VM vm from one PM $pm_{old} = f^{(t-1)}(vm)$ to another $pm_{new} = f^{(t)}(vm)$ as

$$mc : VM \times PM \times PM \to \mathbb{R}_{\geq 0},$$
$$mc(vm, f^{(t-1)}(vm), f^{(t)}(vm)) \mapsto x \in \mathbb{R}_{\geq 0}. \tag{5.9}$$

Until further measurements are available, mc may be assumed a constant function yielding an average value for VM migration. However, as soon as we consider migration from one cloud to another, this function has to be updated.

As to the second part, we define the cost of booting a shut down PM as

$$bc : PM \rightarrow \mathbb{R}_{\geq 0}. \tag{5.10}$$

Finally, we can reformulate our target function as

$$\begin{aligned} \text{minimize} \quad & \sum_{pm_j \in PM_{active}^{(t)}} c(pm_j) \quad + \\ & \sum_{\forall i: f^{(t-1)}(vm_i) \neq f^{(t)}(vm_i)} mc(vm_i) \quad + \\ & \sum_{\forall j: pm_j \notin PM_{active}^{(t-1)} \wedge pm_j \in PM_{active}^{(t)}} bc(pm_j). \end{aligned} \tag{5.11}$$

We know the functions $bc, mc, c, on^{(t)}, f^{(t-1)}$. The last one directly implies $PM_{active}^{(t-1)}$. For $t = 0$ we assume $on^{(0)}(vm) = 0 \quad \forall vm \in VM$, implying $f^{(0)}(vm) = \emptyset \quad \forall vm \in VM$ and $PM_{active} = \emptyset$.

Definition 1 (IaaS management problem)**.** *An IaaS management problem is the problem of determining $f^{(t)}$ as in Equation* (5.4) *(implying $PM_{active}^{(t)}$) subject to the target function (Equation* (5.11)*) and to the constraints formulated by Equation* (5.8)*.*

As a further possibility one may want to relax Equation ((5.8)) for $l =$ storage by introducing a *storage pool* that can be used to satisfy storage needs of a VM that cannot be satisfied by one PM.

5.2 Formulation as a Binary Integer Programming Problem

In order to solve the IaaS management problem with efficient standard algorithms, we want to reformulate it as a *Binary Integer Programming* (BIP) problem. After this formulation we can test the feasibility and scalability of this BIP problem instance by using the built-on MATLAB algorithm `bintprog` [13] and a "hand-made" Matlab algorithm, where we integrated more specific knowledge about this specific instance like infeasible solutions that should not be evaluated. We provide the reader with comparisons of the two algorithms looking for optimal solutions in terms of computation speed and give a limit of their capability to solve problems in reasonable time.

A binary integer programming problem is stated as

$$\min_x f^T x \text{ such that} \tag{5.12}$$
$$A \cdot x \leq b, \tag{5.13}$$
$$Aeq \cdot x = beq, \tag{5.14}$$
$$x \text{ binary}. \tag{5.15}$$

Vector x is a binary vector, f in our case contains the cost function, and A, b, Aeq and beq will be used for our constraints. Calling the MATLAB function `bintprog(f,A,b,Aeq,beq)` will solve the BIP problem with a linear programming (LP)-based branch-and-bound algorithm [13].

We now show the structure of the specified matrices and vectors for our IaaS management problem. At first, we have to define which decision variables we are going to use. Our vector x will consist of the $m \times n$ (of course binary) decision variables x_{ji} for $i \in \{1, \ldots, n\}$ and $j \in \{1, \ldots, m\}$, where x_{ji} signifies whether pm_j hosts vm_i. The following m variables y_j state whether pm_j is turned on. Next, n variables m_i state whether vm_i was migrated from the last iteration to the current one, and the m variables b_j whether pm_j was booted from the last iteration to the current one. Together, this forms

$$x = \begin{pmatrix} x_{11} \\ \vdots \\ x_{m1} \\ x_{12} \\ \vdots \\ x_{m2} \\ \vdots \\ x_{1n} \\ \vdots \\ x_{mn} \\ y_1 \\ \vdots \\ y_m \\ m_1 \\ \vdots \\ m_n \\ b_1 \\ \vdots \\ b_m \end{pmatrix}, \tag{5.16}$$

an $m \times n + 2m + n$ dimensional column vector. Furthermore, the column vector f defines

the cost incurred by vector x.

$$f = \begin{pmatrix} \mathbf{0^{m \cdot n \times 1}} \\ c_1 \\ \vdots \\ c_m \\ mc_1 \\ \vdots \\ mc_n \\ bc_1 \\ \vdots \\ bc_m \end{pmatrix}, \tag{5.17}$$

where $c_i = c(pm_i)$, $mc_j = mc(vm_j, \_, \_)$ and $bc_i = bc(p,_i)$ are defined in the sense of Equation (5.11). The null vector $0^{m \cdot n}$ ignores the allocation of the x_{ji}'s in x, since the allocation itself is not interesting when it comes to the cost the allocation produced.

The biggest part of this definition are the matrices A and Aeq. From the definition of vector x, we know that they both have to have $mn+2m+n$ columns. Matrix A states that the resource demands for every vm_j and every resource k is met and that VMs can only run on powered on PMs. This gives $mk + m$ rows. For the sake of notation, we write r_{lj}^{pm} as an abbreviation for $r_l^{pm}(pm_j)$, and r_{li}^{vm} as an abbreviation for $r_l^{vm}(vm_i)$ as defined in Equation (5.8). At first, we introduce the $mk \times m$-dimensional matrix R_i that accommodates the resource demands for every resource for a specific vm_i,

$$R_i = \begin{pmatrix} r_{1i}^{vm} & 0 & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ r_{ki}^{vm} & 0 & \cdots & 0 \\ 0 & r_{1i}^{vm} & \cdots & 0 \\ \vdots & \vdots & & \vdots \\ 0 & r_{ki}^{vm} & \cdots & 0 \\ & & \ddots & \\ 0 & 0 & \cdots & r_{1i}^{vm} \\ \vdots & \vdots & & \vdots \\ 0 & 0 & \cdots & r_{ki}^{vm} \end{pmatrix}. \tag{5.18}$$

Then, after defining E_n as the n-dimensional identity matrix, we can now define A as

$$A = \begin{pmatrix} R_1 & R_2 & \cdots & R_n & 0^{mk \times m} & 0^{(mk+m) \times (m+n)} \\ E_m & E_m & \cdots & E_m & -n \cdot E_m & \end{pmatrix}. \tag{5.19}$$

The corresponding values for the $m \cdot k + m$ dimensional vector b are therefore

$$b = \begin{pmatrix} r_{11}^{pm} \\ \vdots \\ r_{k1}^{pm} \\ r_{12}^{pm} \\ \vdots \\ r_{k2}^{pm} \\ \vdots \\ r_{1m}^{pm} \\ \vdots \\ r_{km}^{pm} \\ \mathbf{0^{m \times 1}} \end{pmatrix}. \tag{5.20}$$

Finally, we come to the matrix Aeq and its corresponding vector beq. They want to make sure that every VM runs on exactly one PM and consider migrations and powering ups.

The $n \times mn$-dimensional block diagonal matrix $\hat{E}$ ensures that every VM runs on exactly one PM and is formed by n m-dimensional row vectors of 1's, abbreviated $\mathbf{e_m} = (1, 1, \ldots, 1)$. To see how this works, we will give the definition of beq right away. The first vector $\mathbf{e_m}$ in $\hat{E}$ is multiplied by the first m $x_{11}, \ldots x_{m1}$ of $\mathbf{x}$ resulting into 1, thus making sure that vm_1 is deployed on exactly one of the PMs $pm_1, \ldots, pm_m$. This is done for all n VMs.

$$\hat{E} = \begin{pmatrix} e_m & & & \\ & e_m & & \\ & & \ddots & \\ & & & e_m \end{pmatrix}$$

$$beq = \begin{pmatrix} e_n^T \\ e_n^T \\ 0^m \end{pmatrix}$$

Next, we need to consider the allocations from the former iteration stored in x_{ji}^{t-1}, and do this by defining the $n \times mn$ - dimensional matrix X^{t-1}. For the first iteration $t = 1$ we set all $x_{ji}^0 = 0$. Here again, e.g., the first row $x_{11}^{t-1}, \ldots, x_{m1}^{t-1}$ is multiplied by the m decision variables $x_{11}, \ldots x_{m1}$ of $\mathbf{x}$, which gives 1 if and only if vm_1 will not be migrated. Additionally, as we will see with the formulation of Aeq we set an identity matrix E_n after it to be multiplied by the migration variables $m_1, \ldots, m_n$ of $\mathbf{x}$. The sum of both values have to equal 1, thus we conclude that either the VM will not migrate (first summand) or it will migrate by setting the appropriate $m_i = 1$ (second summand).

$$X^{t-1} = \begin{pmatrix} x_{11}^{t-1} & \cdots & x_{m1}^{t-1} & 0 & \cdots & 0 \\ & & & \ddots & & \\ 0 & \cdots & 0 & x_{1n}^{t-1} & \cdots & x_{mn}^{t-1} \end{pmatrix}$$

What remains to be defined are the "powered on" variables $y_1, \ldots, y_m$, as well as the "booting" variables. To see this, we will give the complete definition of Aeq before.

$$Aeq = \begin{pmatrix} \hat{E} & & 0^{n \times 2m+n} & \\ X^{t-1} & 0^{n \times m} & E_n & 0^{n \times m} \\ 0^{m \times mn} & Y^{t-1} & 0^{m \times n} & -E_m \end{pmatrix} \tag{5.21}$$

Consequently, we define Y^{t-1} by the values y_j^{t-1} that state whether pm_j was powered on in the previous time slot. The corresponding variables of vector $\mathbf{x}$ are $y_1, \ldots, y_m$ and $b_1, \ldots, b_m$, and the multiplication results into 0 for every corresponding row. For the first iteration $t = 1$ we set all $y_j^0 = 0$. The concept is similar to the migration variables described before. Either the specific PM had already been powered on in the previous time slot or the corresponding booting variable has to be set to 1, if it is turned on in the current time slot.

$$Y^{t-1} = \begin{pmatrix} 1 - y_1^{t-1} & & & \\ & 1 - y_2^{t-1} & & \\ & & \ddots & \\ & & & 1 - y_m^{t-1} \end{pmatrix}$$

Theorem 1. *The IaaS management problem is an instance of the NP-hard binary integer programming problem.*

Proof. We show that we can reduce every instance of the IaaS management problem to a binary integer programming problem that has been shown to be NP-hard by [105]. To do this reduction, we simply define the necessary parameters A, Aeq, b, beq, f as in Equations (5.19), (5.21), (5.20), (5.2), and (5.17), respectively. All entries of the mentioned matrices and vectors are either pre-specified or determined by the IaaS management problem (Def. 1). □

5.3 Consequences of the NP-hardness

As already stated in the beginning of this chapter we applied two algorithms to solve this problem. The first one, the Matlab `bintprog` algorithm takes the following input with all variables f, A, b, Aeq, and beq as defined above. The variable x stores the solution to our problem.

```
options = optimset('BranchStrategy', 'mininfeas',
     'Diagnostics', 'on', 'Display', 'final',
     'MaxRLPIter', 7800000, 'TolRLPFun', 1.0e-06);
[x, fval, exitflag, output] =
     bintprog(sparse(f),sparse(A),b,sparse(Aeq),beq, [], options)
```

The function `sparse(S)` creates a sparse matrix out of matrix S. The sparse matrix is a more compact writing for matrices that contain a lot of zeros, as it only stores non-zero entries [14]. For the options we evaluate the two branching strategies *mininfeas* and *maxinfeas*. The strategy *mininfeas* selects the "branch variable in the search tree with the minimum integer

infeasibility (the variable whose value is closest to 0 or 1, but not equal to 0 or 1)", whereas *maxinfeas* selects "the variable with the maximum integer infeasibility (the variable whose value is closest to 0.5)". Furthermore, we set `MaxRLPIter` $= 7.8 \cdot 10^6$, which is "the maximum number of iterations the LP-solver performs to solve the LP-relaxation problem at each node." Finally, `TolRLPFun` is set to 10^{-6}, which is the "termination tolerance on the function value of a linear programming relaxation problem" [13]. The shown code snippet just exemplifies one iteration t. For the subsequent iteration $t + 1$ the parameters f, A, b, Aeq, and beq are updated with the x^t and the new demand of VMs, whose changes in demand stem from escalation level 1.

As to the second algorithm, which we call *selected tries*, we generate all m^n possibilities for allocating vm_j on pm_i. Thus, we omit such cases beforehand, where VMs are located on multiple PMs, or on no PM at all. Then, we test all these possibilities and see, whether they represent a valid solution, and if yes, which one achieves the best results. Of course, there are still many invalid solutions tested (for validity), but the search space is still reduced to a certain extent, namely from 2^{mn} to m^n.

As this problem has to be solved for at least more than 100 VMs and PMs and five resource types in practice, we test the scalability of both algorithms. Runtime results are depicted in Table 5.1. We see that already for 6 VMs, 6 PMs and 3 resource types the LP-based algorithm takes around a third of a minute. This would still be acceptable, but for 8 VM and 7 PMs the algorithm does not finish within half an hour, and for 10 VMs, 9 PMs and 4 resource types the algorithm does not terminate within 2 hours. For the selected tries algorithm, finding a solution for 7 VMs, 6 PMs and 3 resource types already takes more than two hours. This is why this algorithm was not evaluated for larger instances.

n	m	k	LP-based branch-and-bound algorithm	Selected Tries
3	3	3	1.73s (mininfeas) - 0.81s (maxinfeas)	0.39s
5	3	5	0.56s (mininfeas) - 0.16s (maxinfeas)	0.05s
6	5	5	3.56s (mininfeas) - 21.72s (maxinfeas)	18.21s
6	6	2	21.99s (mininfeas) - 392.75s (maxinfeas)	182.79s
6	6	3	19.4s (mininfeas) - 402.91s (maxinfeas)	188.06s
6	6	5	0.85s (mininfeas) - 282.51s (maxinfeas)	184.14s
7	5	3	11.63s (mininfeas) - 58.06s (maxinfeas)	616.45s
7	6	3	37.14s (mininfeas) - 1244.5s (maxinfeas)	7523.61s
7	6	4	42.93s (mininfeas)	n/a
8	7	3	> 1800s (mininfeas)	n/a
10	9	4	> 7200s (mininfeas)	n/a

Table 5.1: Runtimes for finding optimal solutions

Thus, we see that the NP-hardness of the BIP problem is also of great relevance in practice for this problem instance.

5.4 Energy-Efficient SLA Enactment

In the following we will present several heuristics to solve the IaaS management problem in a scalable manner combining it with work from Chapter 4. We use a multi-level approach based on the escalation levels introduced in Section 2.3, where we subsume levels 0, 1, 3, and 4. We sequentially work off these levels; levels 0 and 1 are processed by the rule-based approach presented in Section 4.4. Processing levels 3 and 4 is explained in the following.

Energy Model

We find a very elegant way to relate the cost factors for energy c, migrations mc, and booting PMs bc to a natural energy model. Thus, we achieve to relate these costs to realistic values gained from our model and free ourselves from determining arbitrary values for the aforementioned costs. We even enhance the model by incorporating costs for turning off PMs.

As far as our energy model is concerned, we define the energy consumption E of a PM j as

$$E^j = E^j_{min} + ut^{CPU,j} \cdot (E^j_{max} - E^j_{min}), \tag{5.22}$$

where E^j_{min} and E^j_{max} represent respectively the minimum and maximum energy consumption of a certain PM j, and $ut^{CPU,j}$ signifies the utilization of the CPU of PM j, with values between 0 and 1. Thus, in our model energy consumption only depends on CPU utilization, and we make a linear interpolation of the PM's energy consumption at idle state (E_{min} when $ut^{CPU} = 0$) and when fully loaded (E_{max} when $ut^{CPU} = 1$). While this energy model might not be fully realistic, it is corroborated by experiments in the literature such as [82, 41].

We assume that one VM resides on exactly one PM except when it is migrated, then it resides on exactly two PMs. We consider a heterogeneous system, thus VMs and PMs with possibly different amounts of resources, and PMs with different energy characteristics. Furthermore, we assume that the amount of resources a VM is provided can be adapted from one iteration to the next, and that PMs can be powered on and off. However, VM migrations and powering PMs on and off is not considered "free of charge", as far as energy is concerned. We define *migration_time*, *startup_time* and *shutdown_time* as the time (in iterations through the MAPE-K cycle.) it takes a VM to migrate, and a PM to start up or shut down, respectively. Figure 5.1 shows how migrations are handled in our energy model. We do not simply add any arbitrary value as a penalty for migrations, but we place the VMs on both PMs, the PM the VM is migrating from and the PM it is migrating to, for *migration_time* iterations. Thus, the energy cost of a migration is indirectly measured by occupying the resources of two PMs and thus increasing CPU utilization of the other PM as well for a certain period of time. We similarly proceed with the power management of PMs. We do not assume that powering on and off PMs happens instantly, but takes some time as defined in *startup_time* and *shutdown_time*. This is shown in Figures 5.2 and 5.3, respectively.

Figure 5.4 shows a possible configuration of a sample Cloud infrastructure at six time steps. There are 2 PMs and 2 VMs at time step t, one PM is powered off. At $t + 1$, the CPU and memory consumption of the VMs are increasing requiring the second PM to be powered on. The machine is effectively powered on at $t + 2$ and can be used, so we begin the migration of

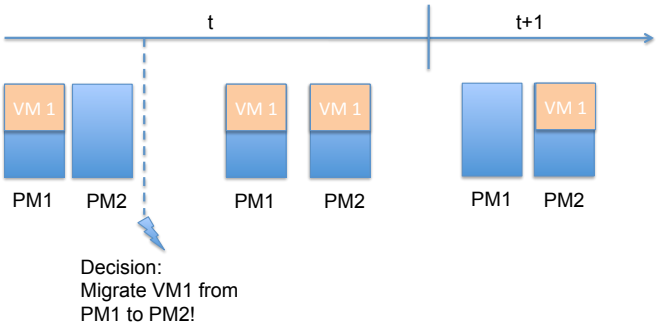


Figure 5.1: Concept for migrations with *migration_time*=1

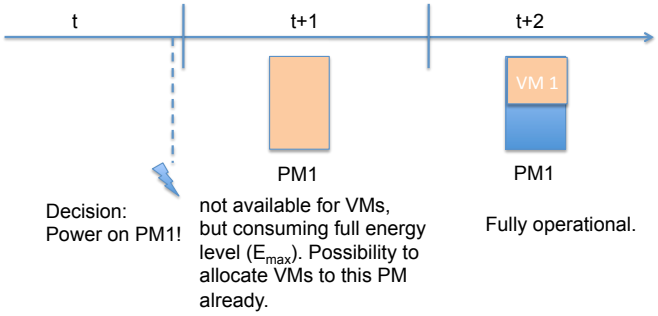


Figure 5.2: Concept for powering on PMs with *startup_time*=1

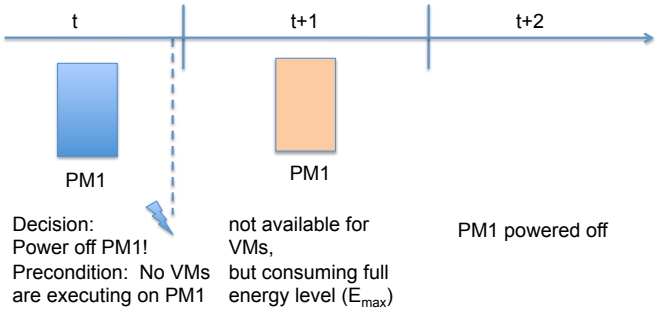


Figure 5.3: Concept for powering off PMs with *shutdown_time*=1

VM 1 from PM 1 to PM 2. The migration will last for one time step. At $t + 3$ the migration ended and we do not need to modify the system again. At $t + 4$ the resource needs of VM 2 has greatly decreased, making it possible to consolidate the system safely by beginning to migrate VM 1 back from PM 2 to PM 1. At the last time step we can shut down PM 2, which is unused.

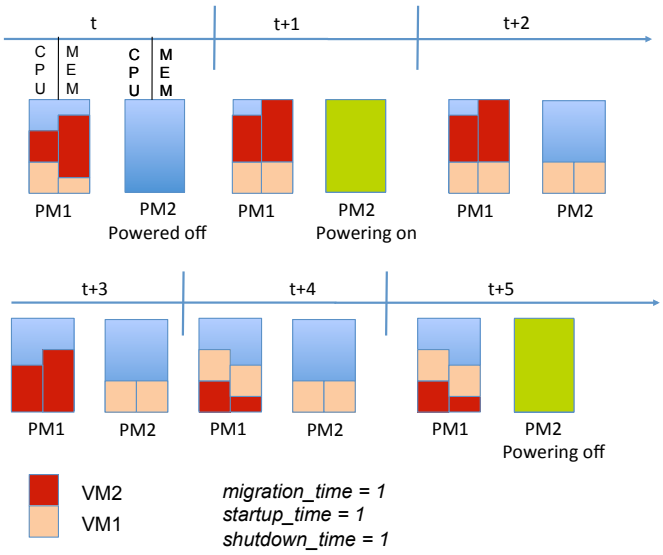


Figure 5.4: Configuration sample at six time steps

Power-Aware Reallocation: VM Migrations

In order to achieve power-aware allocation and reallocation of the VMs to the PMs, we implemented several algorithms with different behaviors. For each algorithm we implemented a first allocation version, which will do the initial mapping of the virtual machines to the physical machines. We then implemented a reallocation algorithm that will output a mapping out of an initial allocation, using the proposed VM migration model.

First Fit

The FIRSTFIT algorithm for the first allocation problem is the well-known mapping algorithm that will allocate each VM to the first PM on which it fits. We, however, added a power-aware component to the algorithm, as we will try to allocate first on the PM that will have the smallest maximum power consumption which, as shown in [48] has proven to consume less energy.

When reallocating the VMs, the usual packing algorithms cannot be used, since we have to take into account where the VMs were allocated at the previous timestep, and the fact that they will consume resources on both source and destination PMs. The algorithm works in two steps. First we pick the most loaded PM and we distribute half its load in a first fit fashion. Then we pick the least loaded PM, and distribute all its load in a first fit fashion.

Round Robin

The ROUNDROBIN algorithm for the first allocation allocates one VM to each PM until no VM is left to allocate. We added the same power-aware component as for the FIRSTFIT algorithm.

When reallocating the VMs, we first take the most loaded PM and spread its load on other non empty PMs in a round robin fashion, then we take one VM on each PM and put it on an

empty PM. This reallocation algorithm, even if it does not perform well power-wise, will serve as a baseline for other algorithms.

Monte Carlo

The MONTECARLO algorithm works on the basis of the well known Monte Carlo method, which uses probabilistic techniques to compute a numerical value. In our case, we will do the first allocation using the ROUNDROBIN algorithm, which allows us to have an evenly balanced system, and that way converge faster to a good solution.

For the reallocation, the algorithm computes the cost of the current allocation using several parameters in order to:

- increase the cost for each VM migrating.
- increase the cost for each overloaded PM.
- decrease the cost for each PM that is empty or will be empty.

Each parameters can be changed to modify the weight of the migrations, PM overloads and PM powering down. In our tests, we will use the values 1 for the migrating VMs, 4 for the overloaded PMs and -10 for a PM that will be empty. This means that if we take for instance a system with 3 PMs and 1 VM migrating from PM 1 to PM 2 we will get the cost : $1 \times 1 + 4 \times 0 + (-10 \times 2) = -15$. The algorithm then computes a random set of VM migrations, in order to get the new cost of the reallocation. If the cost is lower, we keep the set of migrations. The algorithm repeats these operations a fixed number of times, to emerge the best set of migrations to effect for the next iteration.

Vector Packing

The VECTORPACKING algorithm tries for the first allocation to allocate each VM, beginning with the VMs with the highest resource needs on the PMs with the lowest maximum power consumption. It uses a vector packing technique that sorts the VMs according to their highest resource need. It then allocates each VM picking it from the list that will counter the current imbalance in the PM's resource loads.

For the reallocation the algorithm consolidates the VMs as much as possible. It then load-balances some VMs on the most loaded PMs that will not be empty in the future iterations. That way, the algorithm aims to pack the VMs to a minimum number of PMs and then load balance the VMs between those PMs.

PM Power Management

Eventually, actual power can only be saved when PMs are powered off. However, the question of how many PMs should be powered off when in order not to risk future SLA violations is not trivial. Thus, we designed the following powering off strategy: We consider all empty PMs at

the current iteration, i.e., all PMs that have no VMs running on them. We decide to switch off a certain fraction $\frac{1}{a}$ of them, i.e.,

$$\text{Number of PMs to switch off} = \frac{\text{Number of empty PMs}}{a}.$$

This means that when the number of empty PMs stays constant, this technique turns off all but one PMs in exponential manner. Thus, when n represents the (positive) number of empty PMs, we want to know when there will be only 1 PM left. So we need to solve $n \cdot a^{-t} = 1$ for t, which results into $t = -\log_a \frac{1}{n}$. Thus, $\lceil -\log_a \frac{1}{n} \rceil$ is the number of iterations it will take to power off all (but one) PMs. This last PM is kept as a spare PM in order to serve sudden increases in demand as a first resort. Consequently, this technique allows to power off all machines very quickly in case of stable VMs, but always keeps a certain fraction powered on in case VM resource needs start to increase again.

As far as powering on machines is concerned, we monitor average utilization for every resource on all PMs and define – similarly to the rule-based approach (cf. Section 4.4) – resource-dependent threat thresholds. If *any* of these resources exceeds its TT, we power on as many PMs such that the average resource utilization again falls below its TT.

In both cases, we always power on most energy-efficient PMs first, and power off least energy-efficient PMs first.

CHAPTER 6

Evaluation

In this chapter we evaluate the presented approaches from Chapters 4 and 5 with several synthetic and real-world workload data. For this purpose, we present a KM-agnostic simulation engine that implements the autonomic control loop and simulates executed actions and evaluates their quality responding to the workload data at stake.

6.1 Simulation Engine and Workload Generation

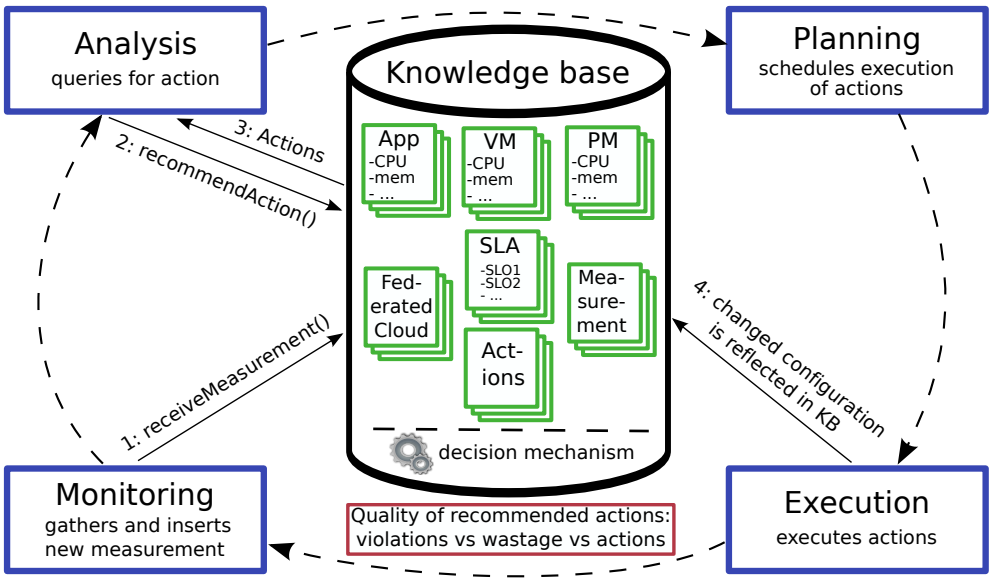


Figure 6.1: Simulation Engine implementing MAPE-K loop

The goal of the simulation engine is to evaluate the quality of a KM system with respect to the number of SLA violations, the utilization of the resources and the number of required reallocation actions. Furthermore, the simulation engine serves as an evaluation tool for any KM technique in the field of Cloud Computing, as long as it can implement the two methods of the KB management interface:

1. `public void receiveMeasurement(int slaID, String[] provided, String[] measurements, List<String> violations);` and

2. `public Actions recommendAction(int slaID);`.

The parameter `slaID` describes the ID of the SLA that is tied to the specific VM, whose provided and measured values are stored in the arrays `provided` and `measurements`, respectively (cf. Section 4.2). The list `violations` contains all SLA parameters being violated for the current measurements. The method `receiveMeasurement` inputs new data into the KB, whereas the method `recommendActions` outputs an action specific to the current measurement of the specified SLA. The simulation engine traverses all parts of the MAPE-K loop as can be seen in Figure 6.1 and described in Section 2.1. The simulation engine is iteration-based, meaning that in one iteration the MAPE-K loop is traversed exactly once. (In reality, one iteration could last from some minutes to about an hour depending on the speed of the measurements, the length of time the decision making takes, and the duration of the execution of the actions, like for example migrating a resource intensive VM to another PM.) The *Monitoring* component receives monitoring information from either synthetic or real-world workload from the current iteration. It forwards the data into the *Knowledge base* (1). The Knowledge base contains representations of all important objects in the Cloud and their characteristic information. These objects are the running applications, the virtual machines, and the physical machines with the current state of their CPU power, memory, storage, etc., the corresponding SLAs with their SLOs, and information about other Clouds in the same federation. Furthermore, the KB also has representations of the inserted measurements, and the available actions to execute (these have to be pre-defined). Finally, the KB also contains a decision mechanism that interprets the state of available objects in order to recommend a reconfiguration action. This mechanism can be substituted by any KM technique; as already mentioned, we used CBR and a rule-based mechanism. The next step in the MAPE loop is the *Analysis* component, which queries the KB for actions to recommend (for a specific SLA id) (2); these actions are then returned to the analysis component (3). The *Planning* component schedules the suggested actions, and the *Execution* component executes them. The changed state configuration of the Cloud objects are automatically reflected in the KB (4). The Monitoring and the Execution components are simulated. This means that the monitoring data is not measured on a real system during the simulation, even though it handles input measured at a real system or synthetic workloads generated beforehand. The Execution component updates the object representation of the manipulated objects in the KB, but obviously does not actually manipulate real-world objects. The quality of the decision making can ultimately be judged by the number of occurred SLA violations, resource wastage and the number of needed reallocation actions.

To evaluate a great variety of workload data, one approach is to create them synthetically. For this, we extended the workload generator as described in [150] to allow a categorization of data volatility.

The workload generator is intended to generate very general workloads for IaaS platforms dealing with slower developments as well as rapid changes. For one parameter, the workload is generated as follows: The initial value of the workloads is randomly drawn from a Gaussian distribution with $\mu = \frac{\text{SLO}}{2}$ and $\sigma = \frac{\text{SLO}}{8}$, where SLO represents the Service Level Objective value agreed in the SLA. Then, an up- or down-trend is randomly drawn, as well as a duration of this trend between a pre-defined number of iterations (for our evaluation this interval of iterations equals $[2, 6]$), both with equal probability. For every iteration, as long as the trend lasts, the current measured value is increased or decreased (depending on the trend) by a percentage evenly drawn from the interval $[iBegin, iEnd]$. After the trend is over, a new trend is drawn and the iterations continue as described before.

Clearly, the values for $iBegin$ and $iEnd$ determine the difficulty for handling the workload. A workload that operates with low $iBegin$ and $iEnd$ values exhibits only very slight changes and does, consequently, not need a lot of dynamic adaptations. Large $iEnd$ values, on the contrary, need the enforcement mechanisms to be very elastically tuned. For the evaluation and comparison of CBR and the rule-based approach we defined a LOW_MEDIUM workload volatility class with $iEnd = 18\%$. For the further evaluation of the rule-based approach we defined and tested LOW, MEDIUM, MEDIUM_HIGH and HIGH workload volatility classes with $iEnd = 10\%, 50\%, 75\%$, and 100%, respectively. As a minimum change we set $iBegin = 2\%$ for all classes.

6.2 Performance Indicators

The subsequent evaluations will be based on the following performance indicators: violations, utilization, actions, resource allocation efficiency (RAE), costs, and time efficiency. Whereas the first three and the last one are rather self-explanatory, costs and RAE need a little more explanation. So violations and actions measure (in percentage) the amount of occurred violations/actions in relation to all possible violations/actions, and utilization the average utilization over all iterations (and over all SLA parameters, if they are not shown explicitly). Time efficiency measures the average time that is needed to handle one VM in one iteration. For resource allocation efficiency we want to relate violations and utilization. The basic is idea is that RAE should equal utilization ($100\% - w$, where w stands for wastage, see below) if no violations occur ($p = 0\%$, where p stands for penalty, see below), equal 0 if the violation rate is at 100%, and follow a linear decrease in between. Thus, we define

$$\text{RAE} = \frac{(100 - w)(100 - p)}{100}. \tag{6.1}$$

A more general approach also taking into account the cost of actions represents the definition of a generic cost function that maps SLA violations, resource wastage and the costs of executed actions into a monetary unit, which we want to call *Cloud EUR*. The cost function is defined by Eq. (4.11). We assume functions $\mathfrak{p}^r$, $\mathfrak{w}^r$ and $\mathfrak{a}^r$ for this evaluation with $\mathfrak{p}^r(p) = 100p$,

$\mathfrak{w}^r(w) = 5w$, and $\mathfrak{a}^r(a) = a$ for all r. The intention behind choosing these functions is (i) to impose very strict fines in order to proclaim SLA adherence as top priority, (ii) to weigh resource wastage a little more than the cost of actions.

Except for the evaluation in Section 6.6 the cost function is not evaluated within the simulation engine. It is a value calculated after the simulation for comparison reasons. Thus, the recommended actions do not depend on the specific functions we assumed. However, in the self-adapting approach explained in Section 4.5 and evaluated in Section 6.6 the cost function is incorporated into the KB in order to adjust and learn the TTs for every resource r.

6.3 Evaluation and Comparison of CBR and Rules

As the crucial parameters for CBR and the rule-based approach differ, we define scenarios for both approaches separately, but still compare them to the aforementioned six performance indicators.

As resources for IaaS one can use all parameters that can be adapted on a VM. For the evaluation we chose to take the following parameters and SLOs for CBR: *storage* $\geq$ 1000GB, *incoming bandwidth* $\geq$ 20 Mbit/s, and the following parameters and SLOs for the rule-based approach: *storage* $\geq$ 1000GB, *incoming bandwidth* $\geq$ 20 Mbit/s, *outgoing bandwidth* $\geq$ 50 Mbit/s, *memory* $\geq$ 512 MB, and *CPU power* $\geq$ 100 MIPS (Million Instructions Per Second).

As far as CBR is concerned, its behavior differs by the α value in Equation (3.2) (setting importance to avoiding violations or achieving high utilization), by the number of executed iterations, because of its inherent learning feature, and the initial cases. At the beginning, we configure all 50 VMs exactly equally with 80% of the storage SLO value and 2/3 of the bandwidth SLO value provided. Then, we execute 2, 5, 10 and 20 iterations with values for α being $0.1, 0.2, 0.3, 0.4, 0.5, 0.6$ and 0.8. We omit values 0.2 and 0.4 in the evaluation because their outcomes do not differ enough from the values shown, and all values > 0.5, because they reveal unacceptable high SLA violation rates. Setting up the initial cases was done by choosing one representative case for each action that could be triggered. For our evaluation the SLA parameters *bandwidth* and *storage* (even though not being tied to them in any way – we could have also named them, e.g., *memory* and *CPU time*) were taken into consideration resulting into 9 possible actions "Increase/Decrease bandwidth by 10%/20%", "Increase/Decrease storage by 10%/20%", and "Do nothing". Taking storage for example, we divide the range of distances for storage St between measured and provided resources into five parts as depicted in Figure 6.2. We choose some reasonable threshold for every action as follows: If $p_{St} - m_{St} = -10$ then action "Increase Storage by 20%" as this already is a violation; if $p^-_{St} - m_{St} = +50$ then action "Increase Storage by 10%" as resources are already scarce but not so problematic as in the previous case; if $p_{St} - m_{St} = +100$ then action "Do nothing" as resources are neither very over- nor under-provisioned; if $p_{St} - m_{St} = +200$ then action "Decrease Storage by 10%" as now resources are over-provisioned; and we set action "Decrease Storage by 20%" when we are over the latest threshold as then resources are extremely over-provisioned. We choose the values for our initial cases from the center of the respective intervals. Ultimately, for the initial case for the action, e.g., "Increase Storage by 20%" we take the just mentioned value for storage and the "Do nothing" value for bandwidth. This leads to $c = (id, 0, -10, 0, 7.5)$, and because only the

differences between the values matter, it is equivalent to, e.g., $c = (id, 200, 190, 7.5, 15.0)$.

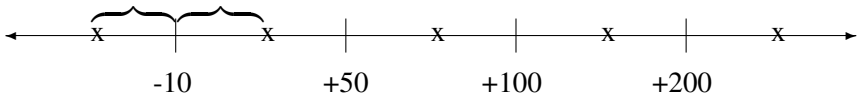


Figure 6.2: Choosing initial cases for CBR using the example of storage

As far as the rule-based approach is concerned, its behavior differs by the set threat thresholds. Thus, we investigate low, middle and high values for TT^r_{low} and TT^r_{high} (as defined in Section 4.4), where $TT^r_{\text{low}} \in \{30\%, 50\%, 70\%\}$ and $TT^r_{\text{high}} \in \{60\%, 75\%, 90\%\}$ for all resources stated above. We combine the TTs to form eight different scenarios as depicted in Table 6.1. We execute 100 iterations with 500 applications, and set the "safety slack" $\epsilon = 5\%$ (cf. Listing 4.1).

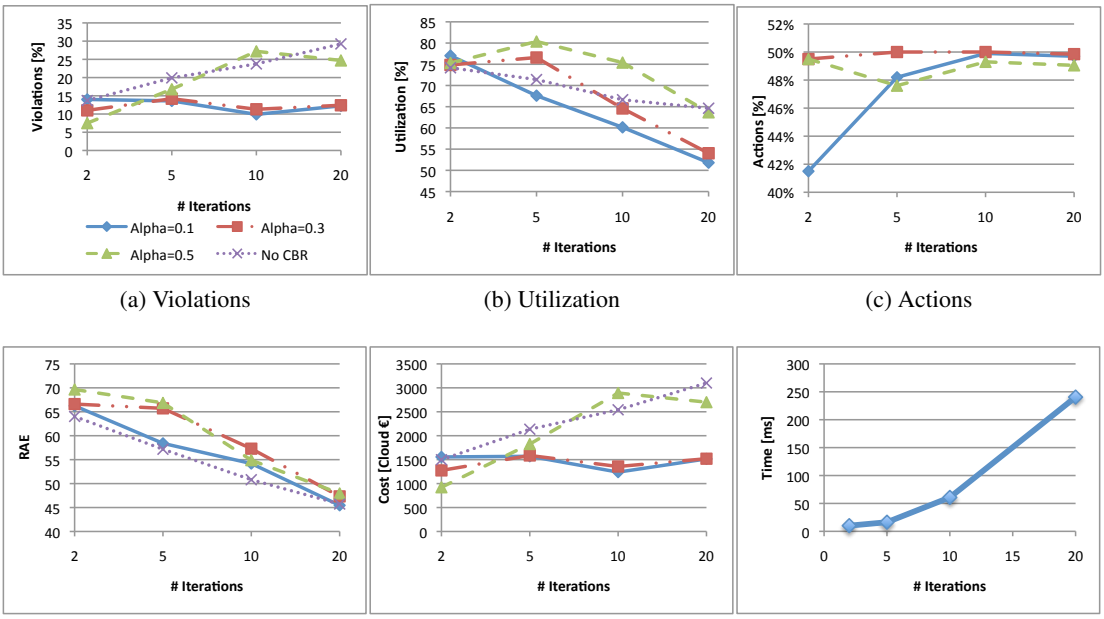
	Scenarios							
	1	2	3	4	5	6	7	8
TT_{low}	30%	30%	30%	50%	50%	50%	70%	70%
TT_{high}	60%	75%	90%	60%	75%	90%	75%	90%

Table 6.1: 8 Simulations Scenarios for TT_{low} and TT_{high}

Figure 6.3 presents the aforementioned performance indicators of CBR. The "No CBR" line means that the autonomic manager is turned off, which implies that the configuration of the VMs is left as set at the beginning, i.e., no adaptation actions due to changing demands are executed. In Figure 6.3a we see that up to more than half of the violations can be avoided when using $\alpha \in \{0.1, 0.3\}$ instead of no autonomic management. However, fewer SLA violations result in lower resource utilization (cf. Figure 6.3b), as more resources have to be provided than can actually be utilized. Reconfiguration actions as depicted in Figure 6.3c lie slightly below or at 50%, except for "No CBR", of course. Another point that can be observed is that after a certain amount of iterations the quality of the recommended actions decreases. This is probably due to the fact that the initial cases get more and more blurred when more cases are stored into CBR, as all new cases are being learned and there is no distinction made between "interesting" and "uninteresting" cases. Nevertheless, when we relate SLA violations and resource utilization in terms of RAE, all CBR methods are generally better than the default method, especially for $\alpha \in \{0.3, 0.5\}$ after five iterations. Yet, RAE decreases strictly monotonically for all α. Furthermore, costs – relating violations, utilization and reconfiguration actions – can also be reduced to half for $\alpha \in \{0.1, 0.3\}$. However, there is a seemingly exponential increase in the average execution time per VM (cf. Figure 6.3f) due to higher number of cases stored in the KB.

Summing up, the simulation shows that learning did take place (and cost some time) and that CBR is able to recommend right actions for many cases, i.e., to correctly handle and interpret the measurement information that is based on a random distribution not known to CBR.

Figure 6.4 shows the same evaluation for the rule-based approach evaluating the aforementioned eight scenarios. From Figure 6.4a we learn that in terms of SLA violations Scenario 1



(a) Violations (b) Utilization (c) Actions

(d) Resource allocation efficiency (e) Cost (f) Average execution time per VM

Figure 6.3: Evaluation of CBR with respect to SLA violations, utilization, actions, RAE, costs, and scalability

achieves the best result, where only 0.0908% of all possible violations occur, and Scenario 8 yields the worst result, with a still very low violation rate of 1.2040%. In general, the higher the values are for TT_{high}, the worse is the outcome. The best result achieved with CBR was at 7.5%. Thus, the rule-based approach achieves an up to 82 times better performance with the right TTs set, and still a 6 times better performance in the worst case.

Figure 6.4b shows resource utilization. We see that the combination of high TT_{low} and high TT_{high} (Scenario 8) gives the best utilization (84.0%), whereas low values for TT_{low} and TT_{high} lead to the worst utilization (62.0% in Scenario 1). Still, compared to CBR which scored a maximum of 80.4% and a minimum of 51.8%, the rule-based approach generally achieves better results.

The percentage of all executed actions as compared to all possible actions that could have been executed is shown in Figure 6.4c. One observes that the greater the span between TT_{low} and TT_{high} is, the fewer actions have to be executed. Most actions (60.8%) are executed for Scenario 7 (span of only 5% between TT values), whereas least actions (5.5%) are executed for Scenario 3 (span of 60% between TT values). CBR almost always recommended exactly one (out of two possible) actions and hardly ever (in about 1% of the cases) recommended no action.

As violations are very low in general, the resource allocation efficiency is very similar to the utilization. The best value can be achieved with Scenario 8 (84.0%), the worst with Scenario 1 (62.0%). CBR achieves a RAE of at most 69.7% ($\alpha = 0.5$ at iteration 2), and at least 45.5% ($\alpha = 0.1$ at iteration 20).

Figure 6.3e shows the costs for each scenario using Equation (4.11) with the parameters set in Section 6.2. The best trade-off between the three terms is achieved by Scenario 5 that has medium values for TT^r_{low} and TT^r_{high}. It has a very low violation rate of 0.0916%, a quite elaborate utilization of 72.9%, but achieves this with only 19.8% of actions. Scenario 7 achieves a better violation and utilization rate but at the cost of an action rate of 60.8%, and consequently has higher costs. The lowest cost value for CBR is 923.0 Cloud EUR, the highest 2985.3 Cloud EUR.

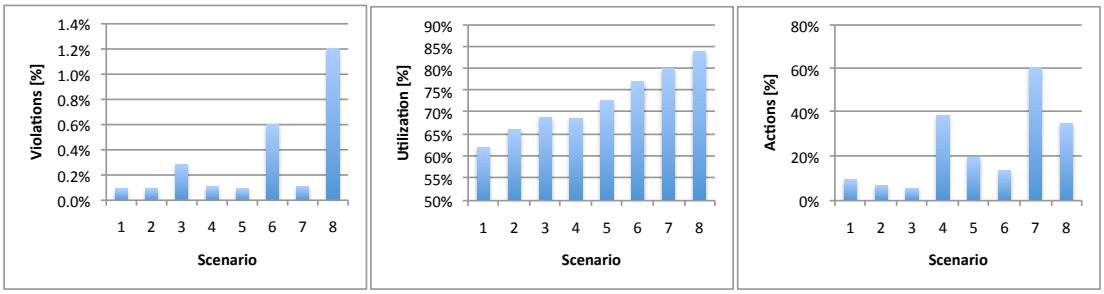
If the utility of the decision decreases for a certain time frame (as cost increases), the KB could determine the cost summand in Equation (4.11) that contributes most to this decrease. For any resource r, if the term is $\mathfrak{p}$, then decrease TT^r_{high}. If the term is $\mathfrak{w}$, then increase TT^r_{low}. Otherwise, if the term is $\mathfrak{c}$, then widen the span of TT^r_{high} and TT^r_{low}, i.e., increase TT^r_{high} and decrease TT^r_{low}. This is one of the basic ideas for Section 4.5.

As far as time performance and scalability are concerned, the performance tests are very encouraging. We executed 100 iterations from 100 to 3000 VMs. We performed every test twice and calculated the average execution time as well as the average time it took for the simulation engine to handle one VM. As shown in Figure 6.4f the execution time per VM stays quite constant for up to 1500 VMs, and thus average execution time is about linear. For 3000 VMs, it took $647s/100 = 6.47s$ for one iteration to treat all VMs. The high time consumption per VM for 100 VMs in Figure 6.4f is due to the initialization of the rule knowledge base which takes over-proportionally long for just a small number of VMs and does not weigh so much for more VMs.

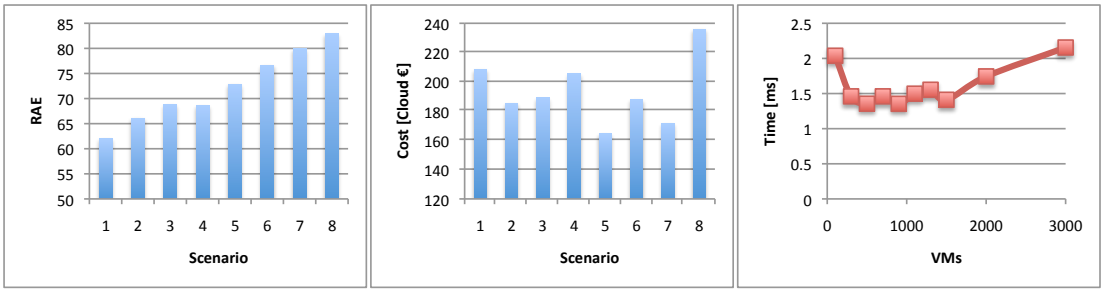
CBR took 240s for 50VMs and 20 iterations. Thus, CBR took $240s/20 = 12s$ for one iteration to treat all VMs, which is twice as long as the rule-based approach takes, which even has 60 times more VMs. However, CBR implements learning features, what the rule-based approach currently does not, and could be sped up by choosing only specific cases to be stored in the KB.

Summarizing, the rule-based approach highly outperforms CBR with respect to violations (up to 82 times better results), actions, cost, and time performance. The rule-based approach also achieves better "best case" and better "worst case" results for the remaining performance indicators utilization and resource allocations efficiency. In more detail, 7 out of 8 scenarios were better than the worst CBR value for utilization, whereas only one scenario was better than the best CBR utilization value. Again, accumulating these results into cost, all rule-based scenarios outperform CBR by a factor of at least 4 (worst rule-based scenario (236) compared to best CBR result (923)), which to a large extent is due to the huge number of violations that the rule-based approach is able to prevent and the high number of actions it can save.

Consequently, we consider the rule-based approach as the better technique to deal with VM reconfiguration in Cloud Computing infrastructures, and we will focus the remaining part of this article on a deeper investigation and understanding of the rule-based approach by evaluating it with different classes of synthetic and real world workload.



(a) Violations (b) Utilization (c) Actions



(d) Resource allocation efficiency (e) Cost (f) Average execution time per VM

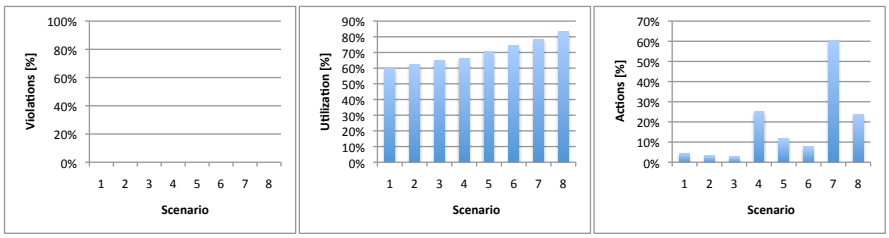
Figure 6.4: Violations, Utilization, Actions and Utility for Scenarios 1-8, Execution time for Rule-based Approach

6.4 In-depth Evaluation of the Rule-based Approach Using Synthetic Data

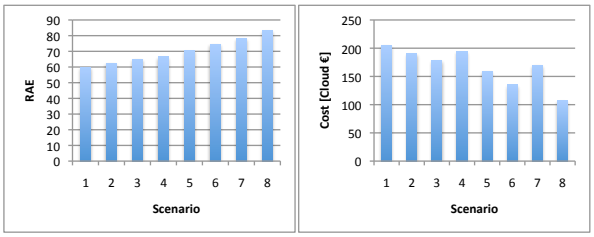
This section deals with the further investigation of the rule-based approach. We evaluated all eight scenarios with different workload classes, namely LOW, MEDIUM, MEDIUM_HIGH, and HIGH as defined in Subsection 6.1.

For the LOW workload volatility class (cf. Figure 6.5) one remarks that all violations can be completely avoided for all TT scenarios. Lowest cost (107.7 Cloud EUR) is achieved with Scenario 8, even though the amount of actions is quite high (23.8%), but the utilization (83.2%), and therefore also RAE (83.2%), is highest.

The MEDIUM workload volatility class (cf. Figure 6.6) already runs into a lot more violations reaching a peak at 15.9% with Scenario 8, but still achieving a very good rate of 0.7% and 0.8% with Scenarios 1 and 4, respectively. Generally, reconfiguration actions are a lot higher, too, with a minimum of 18.9% (Scenario 3), whereas the minimum for the LOW workload is at 2.7% (Scenario 3). RAE differs quite apparently from utilization, and achieves its best rate with Scenario 7 (69.4%), where utilization is (second highest) at 75.2% and violations are (third highest) at 7.7%. Due to the many violations, costs are much higher and go up to 1746.4 Cloud EUR. The best results stem from Scenario 1 with a cost of 294.8 Cloud EUR. The second best scenario, Scenario 4 with a cost of 301.6 Cloud EUR, also achieves a similarly good violation



(a) Violations (b) Utilization (c) Actions



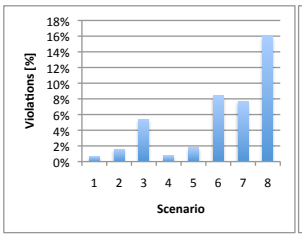
(d) Resource allocation efficiency (e) Cost

Figure 6.5: Violations, Utilization, Actions, RAE and Cost for Scenarios 1-8 with LOW volatility workload

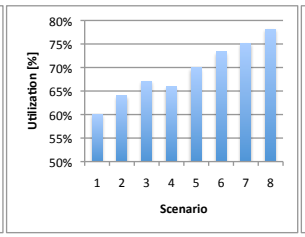
rate, but differs by a higher action rate (50.7% vs. 28.0%) and a lower utilization (60.0% vs. 65.9%).

As to the MEDIUM_HIGH workload volatility class (cf. Figure 6.7) the peak for violations raises up to 22.1% (Scenario 8), but still achieves a very good minimum of 1.9% (Scenario 1), which can be achieved with the third lowest amount of actions (38.2%). Generally, the graphs for the MEDIUM_HIGH and HIGH workloads (cf. Figure 6.8) repeat the pattern of the MEDIUM workloads just with higher amplitudes. Also, the lowest-cost scenario for these workloads is the same, namely Scenario 1. However, the lowest-cost scenarios for LOW and LOW_MEDIUM classes, namely Scenarios 1 and 8, respectively, differ quite significantly. Scenario 1 is a combination of two low TTs, Scenario 5 of two middle TTs, and Scenario 8 of two high TTs.

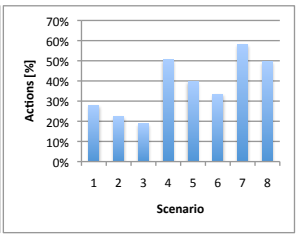
Summarizing we have seen that across all scenarios violations, actions and cost increase from the LOW to the HIGH workload volatility classes. However, we have also seen that by choosing the "right" TTs, the cost can be kept relatively small as compared to "wrong" TTs, or the CBR outcome. E.g., for the HIGH workload class the best violation rate is at 4.1% (Scenario 1), whereas the worst violation rate is at 30.3% (Scenario 8). Consequently, it is crucial to autonomically find good TTs according to the respective workload. This will be investigated in Section 6.6.



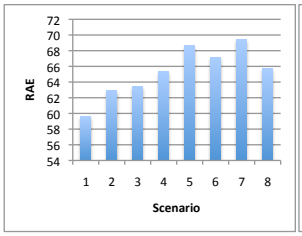
(a) Violations



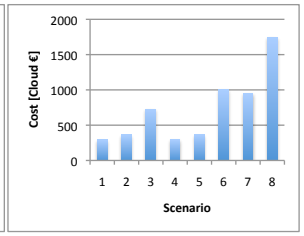
(b) Utilization



(c) Actions

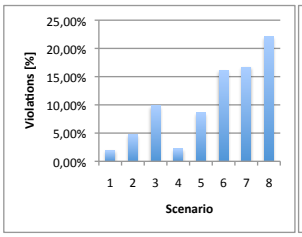


(d) Resource allocation efficiency

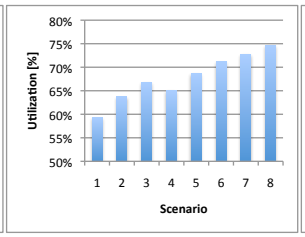


(e) Cost

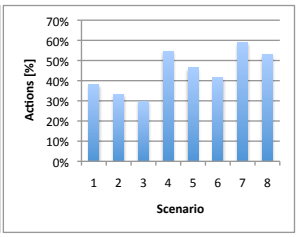
Figure 6.6: Violations, Utilization, Actions, RAE and Cost for Scenarios 1-8 with MEDIUM volatility workload



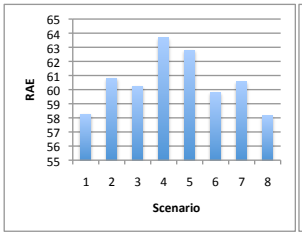
(a) Violations



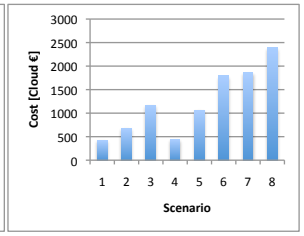
(b) Utilization



(c) Actions

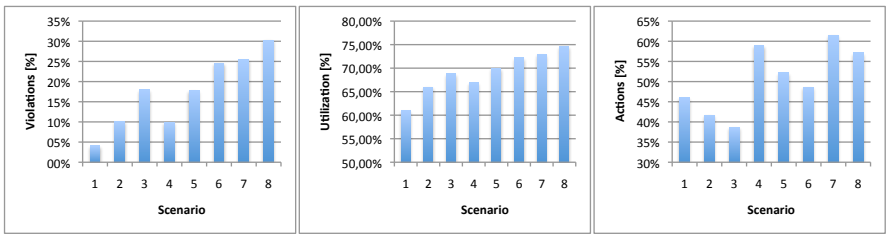


(d) Resource allocation efficiency

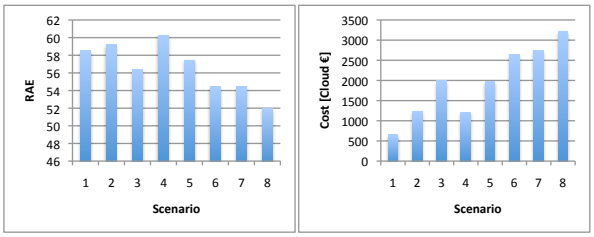


(e) Cost

Figure 6.7: Violations, Utilization, Actions, RAE and Cost for Scenarios 1-8 with MEDIUM_HIGH volatility workload



(a) Violations (b) Utilization (c) Actions



(d) Resource allocation efficiency (e) Cost

Figure 6.8: Violations, Utilization, Actions, RAE and Cost for Scenarios 1-8 with HIGH volatility workload

6.5 Applying and Evaluating a Bioinformatics Workflow to the Rule-based Approach

This section describes the adoption of a bionformatics workflow as a Cloud computing application. We demonstrate by simulation that the rule-based approach can guarantee the resource requirements in terms of CPU, memory and storage for the execution of the workflow in a resource-efficient way.

As detailed in [187, 192], bioinformatics workflows have gained a great need for large-scale data analysis. Due to the fact that these scientific workflows are very resource intensive and can take hours if not days to complete, provisioning them in an environment with fixed resources leads to poor performance. On the one hand, the workflow might run out of resources and thus may have to be restarted on a larger system. On the other hand, too much resources might be provisioned in order not to take risks of a premature abort, which may cause a lot of resources being wasted. Thus, Cloud computing infrastructures offer a promising way to host these sorts of applications [161]. The monitoring data presented in this Section was gathered with the help of the Cloud monitoring framework LoM2HiS [79]. Using LoM2HiS we measured utilized resources of TopHat [198], a typical bioinformatics workflow application analyzing RNA-Seq data [126], for a duration of about three hours [80].

In the following we shortly describe the bioinformatics workflow in more detail. We here consider Next Generation Sequencing (NGS), a recently introduced high-throughput technology for the identification of nucleotide molecules like RNA or DNA in biomedical samples. The

output of the sequencing process is a list of billions of character sequences called ‘reads’, each typically holds up to 35-200 letters that represent the individual DNA bases determined. Lately, this technology has also been used to identify and count the abundances of RNA molecules that reflect new gene activity. We use the approach, called RNA-Seq, as a typical example of a scientific workflow application in the field of bioinformatics.

At first, in the analysis of RNA-Seq data, the obtained sequences are aligned to the reference genome. The aligner presented here, TopHat [198], consists of many sub-tasks, some of them have to be executed sequentially, whereas others can run in parallel (Figure 6.9). These sub-tasks can have different resource-demand characteristics: needing extensive computational power, demanding high I/O access, or requiring extensive memory size.

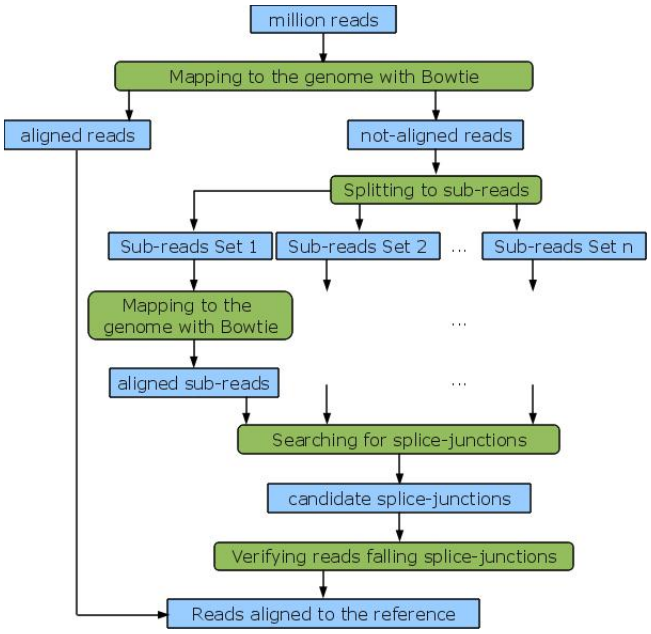


Figure 6.9: Overview of the TopHat Aligning Approach

In Figure 6.9, the green boxes represent simplified sub-tasks of the workflow application, whereas the blue boxes represent the data transfered between the sub-tasks. The first sub-task aligns input reads to the given genome using the Bowtie program [127]. Unaligned reads are then divided into shorter sub-sequences which are further aligned to the reference genome in the next sub-task. If sub-sequences coming from the same read were aligned successfully to the genome, that may indicate that this read was straddling a ‘gap’ in the gene, falling on a so-called splice-junction. After verification of candidate reads falling on splice junctions, these and the reads that were aligned in the first sub-task are combined to create an output with a comprehensive list of localized alignments.

For the simulation we define the SLA shown in Table 6.2 for TopHat with the maximum amount of available resources on the physical machine we are executing it. The physical machine has a Linux/Ubuntu OS with a Intel Xeon(R) 3 GHz CPU, 2 cores, 9 GB of memory, and 19 GB

of storage. For CPU power, we convert CPU utilization into MIPS based on the assumption that an Intel Xeon(R) 3 GHz processor delivers 10000 MIPS for 100% resource utilization of one core, and linearly degrades with CPU utilization.

Service Level Objective (SLO) name	SLO value
CPU Power	$\geq$ 20000 MIPS
Memory	$\geq$ 8192 MB
Storage	$\geq$ 19456 MB

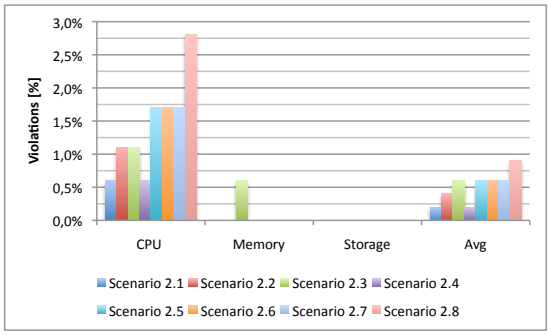
Table 6.2: TopHat SLA

In order to validate our approach, we make three simulations categories, where we set up and manage our VMs differently: In the first category (Scenario 1) we assume a static configuration with a fixed initial resource configuration of the VMs. Normally, when setting up such a testbed as described in [80], an initial guess of possible resource consumption is done based on early monitoring data. From this data on, we assume quite generous resource limits. The first ten measurements of CPU, memory, and storage lie in the range of [140, 12500] MIPS, [172, 1154] MB, [15.6,15.7] GB, respectively. So we initially configured our VM with 15000 MIPS, 4096 MB, and 17.1 GB, respectively. The second category subsumes several scenarios, where we apply our autonomic management approach to the initial configuration in the first category. The eight scenarios in this category depend on the chosen TTs. According to Table 6.1 we define these scenarios as Scenario 2.1., 2.2, ..., 2.8, respectively. As the third category (Scenario 3), we consider a best case scenario, where we assume to have an oracle that predicts the maximal resource consumption that we statically set our VM configuration to. Moreover, according to the first measurements we decide to enforce a minimum of 1 MIPS CPU, 768 MB memory, and 1 GB storage.

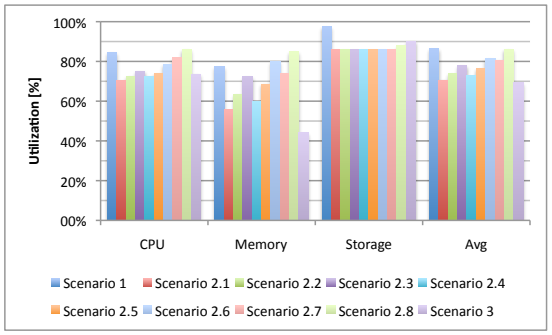
As depicted in Figures 6.10a, 6.10b, and 6.10c one sees violations, utilization, as well as the number of reconfiguration actions, respectively, for every parameter (together with an average value) in the different scenarios. Generally, the bars are naturally ordered beginning from Scenario 1, over Scenarios 2.1, ..., 2.8, ending with Scenario 3. The number of violations in Scenario 1 reach 41.7% for CPU and memory, and 49.4% for storage, which leads to an average of 44.3%. (For better visibility, these results have been excluded from Figure 6.10a.) Thus, we experience violations in almost half of the cases. This is especially crucial for parameters *memory* and *storage*, where program execution could fail, if it runs out of memory or storage, whereas for a violation of the parameter *CPU*, we would "only" delay the successful termination of the workflow.

With Scenarios 2.* we can reduce the SLA violations to a minimum. We completely avoid violations for *storage* in all sub-scenarios, as well as for *memory* in all but one sub-scenarios. Also *CPU* violations can be reduced to 0.6% for Sub-scenarios 2.1 and 2.4, and still achieve a maximum SLA violation rate of 2.8% with Scenario 2.8. The average SLA violation rate can be lowered to 0.2% in the best case. Scenario 3, of course, shows no violations. However, it is unlikely to know the maximum resource consumption before workflow execution.

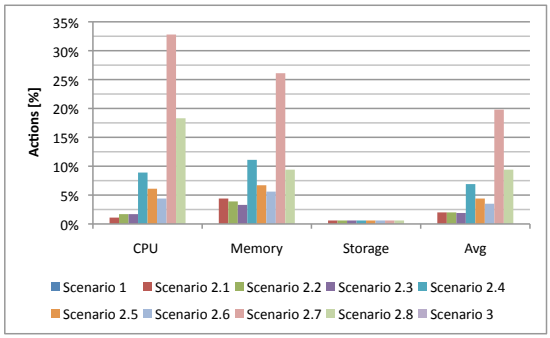
As to the utilization of the resources, it is clearly higher when a lot of violations occur, so



(a) Violations

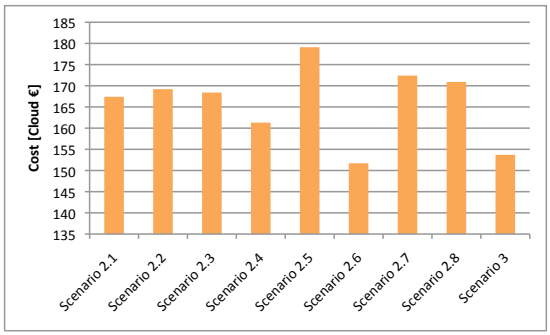


(b) Utilization

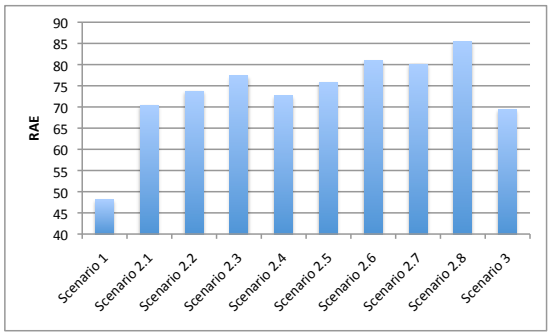


(c) Reconfiguration actions

Figure 6.10: Violations, Utilization and Reconfiguration actions for ten autonomic management scenarios using bioinformatics workflow



(a) Cost



(b) Resource Allocation Efficiency

Figure 6.11: Resource Allocation Efficiency and Cost for ten autonomic management scenarios using bioinformatics workflow

Scenario 1 naturally achieves high utilization. This is the case, because when a parameter is violated, then the resource is already fully used up, but even more of the resource would be needed to fulfill the needs. On the opposite, Scenario 3 naturally achieves low utilization, as a lot of resources are over-provisioned. Scenarios 2.* achieve a good utilization that is on average in between of the two extremes and ranges from 70.6% (Scenario 2.1) to 86.2% (Scenario 2.8). Furthermore, we observe some exceptions to this "rule" when considering individual parameters. So, e.g., for memory we achieve a utilization of 85.0 % with Scenario 2.8 or 80.0% with Scenario 2.6, which is higher than the utilization in Scenario 1 (77.4%). The same is true for CPU utilization rates of 85.5% as compared to 84.3 % for the Scenario 1 and 2.8, respectively. Only for storage the utilization of all but one of the scenarios 2.*, which is at 85.9%, is smaller than for Scenario 3 (90.1%).

A huge advantage of Scenarios 2.* is that they do not run into any crucial SLA violation (except for Scenario 2.3) , but achieve a higher utilization as compared to Scenario 3. As to the reallocation actions, of course, Scenario 1 and 3 do not execute any, but also for the autonomic management in Scenarios 2.*, the amount of executed reallocation actions for most scenarios stays below 10%. Only Scenario 2.7 executes actions in 19.8% of the cases on average of the time. Five out of eight scenarios stay below 5% on average.

When it comes to the overall costs of the scenarios (cf. Figure 6.11a), all 2.* scenarios approach the result achieved by the best case scenario 3. Scenario 1 sums up costs of 4493.6, and has therefore been omitted in the figure. Furthermore, the lowest cost is achieved using Scenario 2.6, which is even lower than the cost for Scenario 3. This is possible, because Scenario 2.6 achieves a very good utilization and SLA violation rate with a very few number of reallocation actions. Also resource allocation efficiency for Scenarios 2.* as shown in Figure 6.11b achieves unambiguously better results than for Scenario 1 (RAE of 48.2%). Furthermore, all scenarios of the second category achieve a better RAE than the RAE of Scenario 3 (69.3%).

Thus, we conclude that by using the suggested autonomic management technique, we can avoid most costly SLA violations, and thus ensure workflow execution, together with a focus on resource-efficient usage. All this can be achieved by a very low number of time- and energy consuming VM reallocation actions for many of the autonomic management scenarios.

6.6 Evaluation of the Self-adapting Rule-based Approach

Evaluation of the Self-adapting Rule-based Approach Using Synthetic Data

In this subsection we evaluate the six options A-F presented in Section 4.5 using synthetic workload. As a quality measure, we will use the cost function defined by Equation (4.11) with $\mathfrak{p}^r(p) = 100p, \mathfrak{w}^r(w) = 5w$, and $\mathfrak{a}^r(a) = a$ for all r, and for all adaptation options we set $\alpha = 4$ as used in Equations (4.14)-(4.17).

Every simulation run consists of 100 iterations. The SLA for the synthetic workloads is presented in Table 6.3. Results of the simulation runs can be seen in Figures 6.12 - 6.14. In all Subfigures 6.12-6.15(a) we present p, $100 - w$, a for every simulation run. The specifics of each run are explained below each group of three bars: At first the adaptation option is stated, or "off", if none is used. Adaptation options also show k where applicable. All autonomic

r_{min}		SLA parameter		r_{max}
1 GB	$\leq$	storage	$\leq$	1000 GB
1 Mbit/s	$\leq$	incoming bandwidth	$\leq$	20 Mbit/s
1 Mbit/s	$\leq$	outgoing bandwidth	$\leq$	50 Mbit/s
1 MIPS	$\leq$	CPU power	$\leq$	100 MIPS
8 MB	$\leq$	memory	$\leq$	512 MB

Table 6.3: SLA for synthetic workloads

TT experiments have been conducted with $TT_{low} = 50\%$ and $TT_{high} = 75\%$ initially set (we will refer to this as the *standard case*), unless stated otherwise. This was chosen based on the evaluation in [151], as this setting brought best results for a LOW_MEDIUM WV class with $iEnd = 18\%$. For compact notation a TT pair is written as $[TT_{low}, TT_{high}]$. In all Subfigures 6.12-6.15(b) we show the cost $c(p, w, c)$ with the parameters as defined above.

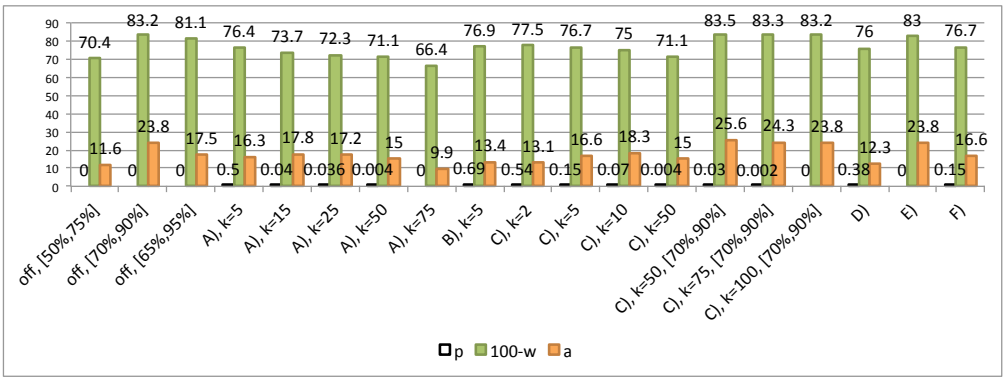
The first three (group of) bars in Figure 6.12 represent static TT configurations evaluated in [151]. The goal of the autonomic TT management is to achieve costs that are as low or lower than the costs resulting from a static TT configuration. We see that the best static result in terms of costs can be achieved setting $TTs = [70\%, 90\%]$, and the cost for the standard case is 159. This value is beaten (or attained) by evaluated options A for $k \leq 25$, C for $k = 2, 5$ with the standard TT pair, C for all evaluated k with the best (a-priori unknown) TT pair, and options E and F. The best case is attained by options C with the best TT pair, and by option E.

For the MEDIUM WV class we deduce from Figure 6.13 that options A for $k \geq 15$, E and F beat the static TT scenario. On the contrary, option C achieves the worst results by far.

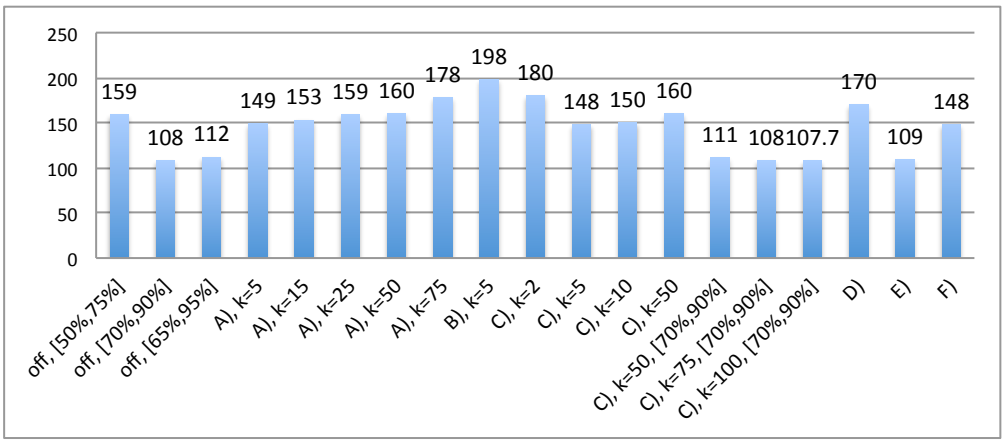
Due to space limitations, we omit the graphs of the MEDIUM_HIGH WV class, which are quite similar to those of the HIGH WV class. Evaluation shows that all options except option C beat the results from the standard case. Option E achieves the best result.

As far as the HIGH WV class is concerned (cf. Figure 6.14), all options beat the results from the "standard case". From these, again option C still achieves the worst results, and again option E results into lowest costs.

Generally, autonomic adaptation works best for workloads with higher volatility and quite acceptable for workloads with lower volatility. We also see that option C for $k = 5$ generally achieves worst results except for low WV. This is explained by the fact as stated in Section 4.5 that option C is less cautious than other options with respect to SLA violations. These violations, naturally, have a higher impact with higher WV. Option B for $k = 5$ achieves the worst result for LOW WV, and only outperforms the standard case for MEDIUM_HIGH and HIGH WV classes. Nevertheless, options E and F always outperform the standard case, and achieve best or very good results, and there is always a k for option A such that it also outperforms the standard case. The best cases for each WV class have been resembled in option F.



(a) Violations p, utilization $100 - w$, actions a



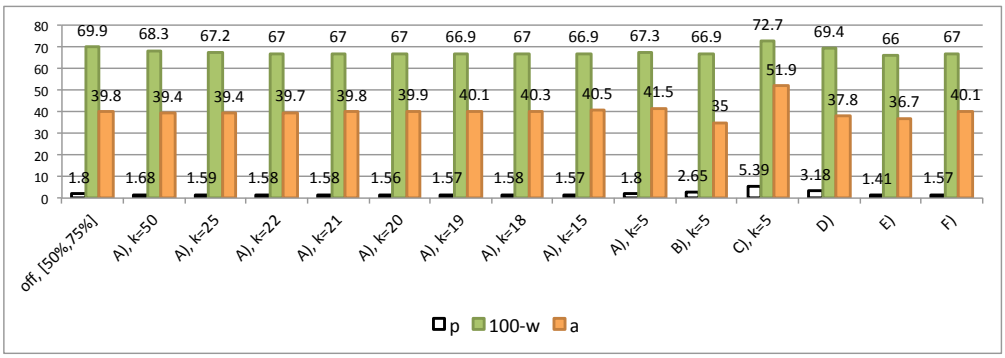
(b) Cost $c(p, w, a)$

Figure 6.12: Evaluation results for LOW workload volatility class

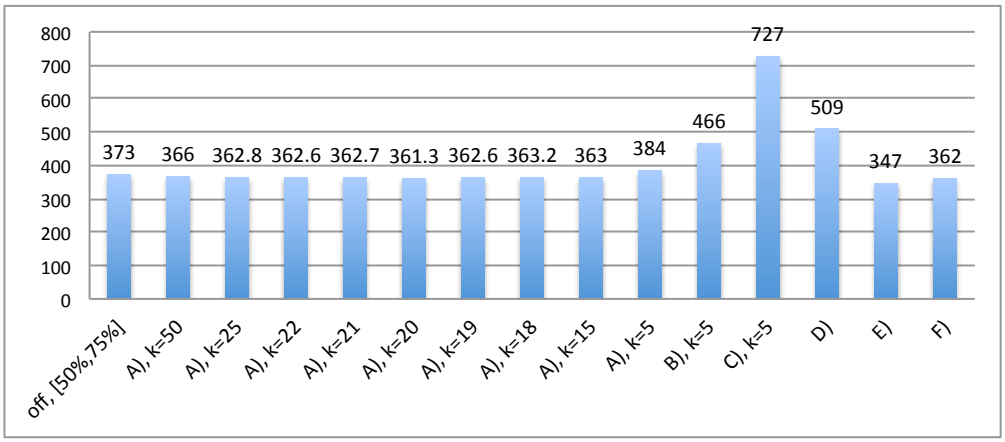
Evaluation of the Self-adapting Rule-based Approach Using Image Rendering Software Workload

This and the next section will present the evaluation of two real-world workloads categories. One important point to observe with these workloads is that they do no longer fall into the same WV class for all the resources.

The SLA for the *POV-Ray* application, an image rendering software [1], is depicted in Table 6.4. As we have seen that in the previous subsection options E and F always outperform the standard case, we chose only these options for further evaluation. As can be seen in Table 6.5 (AM describes whether the autonomic manager is turned on or off), we remark that for POV_F* options E and F always outperform the standard case with partially big cost improvements up to 48% (for POV_F9), while the better option is not clearly the one or the other. For POV_B* workloads there is one case, where neither option outperforms the standard case, whereas in the other cases either option E or option F outperform the standard case.



(a) Violations p, utilization $100 - w$, actions a



(b) Cost $c(p, w, a)$

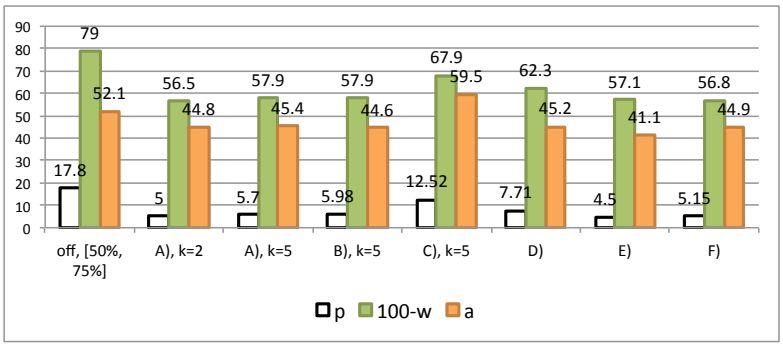
Figure 6.13: Evaluation results for MEDIUM workload volatility class

r_{min}		SLA parameter		r_{max}
1 GB	$\leq$	storage	$\leq$	1000 GB
1 Kbit/s	$\leq$	incoming bandwidth	$\leq$	80000 Kbit/s
1 Kbit/s	$\leq$	outgoing bandwidth	$\leq$	8000 Kbit/s
1 MIPS	$\leq$	CPU power	$\leq$	100000 MIPS
8 MB	$\leq$	memory	$\leq$	512 MB

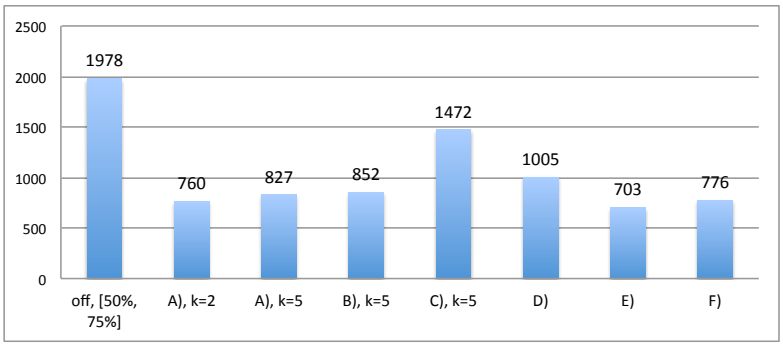
Table 6.4: PovRay SLA

p	$100 - w$	a	$c(p, w, c)$	WV	AM	Details
5.56	63.8	17.0	754	POVRAYF1	off	[50%, 75%]
2.56	50.96	11.56	512	POVRAYF1	on	A), 2step
3.0	56.34	14.2	533	POVRAYF1	on	E)
3.0	53.44	11.7	544	POVRAYF1	on	F)
1.45	72.1	12.3	297	POVRAYF2	off	[50%, 75%]
0.68	69.6	9.4	229	POVRAYF2	on	E)
1.13	70.5	15.5	275.8	POVRAYF2	off	F)
1.34	72.0	7.7	282	POVRAYF3	off	[50%, 75%]
1.12	71.7	7.8	261	POVRAYF3	on	E)
0.45	68.9	6.5	207	POVRAYF3	on	F)
1.56	71.7	9.3	306	POVRAYF4	off	[50%, 75%]
0.89	71.4	8.2	240	POVRAYF4	on	E)
0.89	66.3	5.4	263	POVRAYF4	on	F)
1.89	72.1	10.8	339	POVRAYF5	off	[50%, 75%]
0.89	69.9	9.3	249	POVRAYF5	on	E)
1.0	69.8	14.1	265	POVRAYF5	on	F)
3.02	72.4	13.2	453	POVRAYF6	off	[50%, 75%]
0.89	68.5	10.8	258	POVRAYF6	on	E)
1.56	69.9	16.5	324	POVRAYF6	on	F)
2.78	72.5	12.4	428	POVRAYF7	off	[50%, 75%]
0.89	69.0	10.3	254	POVRAYF7	on	E)
1.56	70.1	16.2	321	POVRAYF7	on	F)
3.44	72.4	14.0	496	POVRAYF8	off	[50%, 75%]
1.0?	67.4	9.9	273	POVRAYF8	on	E)
1.89	67.0	17.2	356	POVRAYF8	on	F)
3.24	72.9	15.8	475	POVRAYF9	off	[50%, 75%]
0.78	68.7	12.1	247	POVRAYF9	on	E)
1.34	70.1	18.4	302	POVRAYF9	on	F)
3.91	73.1	16.2	542	POVRAYF10	off	[50%, 75%]
1.23	68.3	12.2	293	POVRAYF10	on	E)
2.01	70.5	18.7	367	POVRAYF10	on	F)
0.45	72.2	6.1	190	POVRAY_B1	off	[50%, 75%]
0.44	73.0	6.0	186	POVRAY_B1	on	E)
0.56	72.3	9.2	204	POVRAY_B1	on	F)
0.11	71.2	10.1	161	POVRAY_B2	off	[50%, 75%]
0.11	71.8	6.9	159	POVRAY_B2	on	E)
0.22	72.5	10.8	171	POVRAY_B2	on	F)
0.22	72.5	10.3	170	POVRAY_B3	off	[50%, 75%]
0.45	71.8	8.8	194	POVRAY_B3	on	E)
0.34	69.7	6.0	191	POVRAY_B3	on	F)

Table 6.5: Measurement results for PovRay measurements



(a) Violations p, utilization $100 - w$, actions a



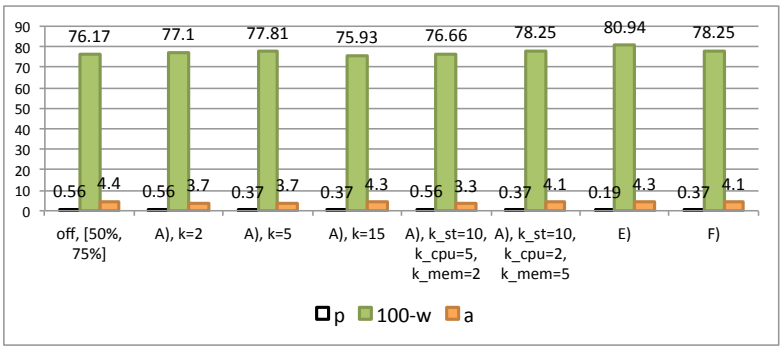
(b) Cost $c(p, w, a)$

Figure 6.14: Evaluation results for HIGH WV class

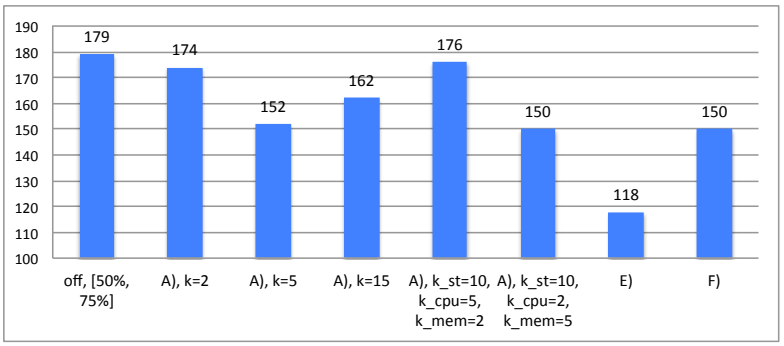
Evaluation of the Self-adapting Rule-based Approach Using a Bioinformatics Workflow Workload

The SLA of the second workload, the bionformatics workflow, is defined as follows (similarly as in Table 6.2): 1 MB $\leq$ *storage* $\leq$ 19456 MB, 1 MIPS $\leq$ *CPU Power* $\leq$ 20000 MIPS, and 768 MB $\leq$ *memory* $\leq$ 8192 MB. Figure 6.15 reveals that all evaluated autonomic options outperform the standard case with option E achieving by far the best result. For option A we have also experimented with varying k for different resources and could achieve the second best result (tied with option F) by setting $k = 10$ for storage, $k = 2$ for CPU, and $k = 5$ for memory.

Concluding we find that for 11 out of 14 real-world workloads both options E and F of the self-adaptive approach achieve better results than the static approach for at least 7% (workload POV_F2) and at most 48% (workload POV_F9). From the remaining workloads, for two of them (POV_B1 and POV_B2) only option E performs better, and for only one workload the static approach outperforms both self-adaptive ones by 11% (POV_B3).



(a) Violations p, utilization $100 - w$, actions a



(b) Cost $c(p, w, a)$

Figure 6.15: Evaluation results for the bioinformatics workflow

6.7 Energy-efficient and SLA-Aware Management of IaaS Clouds

In this section we will evaluate the more holistic framework for VM reconfiguration, VM migration and PM power management. We divide the evaluation into four experiments. In the first experiment we want to determine the energy gain VM reconfiguration brings alone. In the second experiment we focus on the four different reallocation algorithms to see which one performs best. In the third experiment we more deeply investigate some of the parameters for the two best reallocation algorithms. Finally, with the fourth experiment we evaluate the scalability of the algorithms.

We simulated 100 physical machines with 1.1GHz processors and 4GB memory, that consume 20W at idle (E_{min}) and 100W when fully loaded (E_{max}). We used several different workloads for the 100 VMs of the system, two synthetic ones and one based on real measurements of a scientific bioinformatics workflow presented in [80]. For the synthetic workloads we distinguish between LIGHT workload volatility, i.e., workload does not change a lot (up to 10% from one iteration to the other), and the opposite MEDIUM_HEAVY (up to 50% from one iteration to the other workload volatility. A more detailed description of the workload generation can be found in [151]. We will abbreviate the bioinformatics workflow with BOKU. We evaluated the algorithms with 100 iterations and the PM powering off strategy with $a = 2$, unless stated

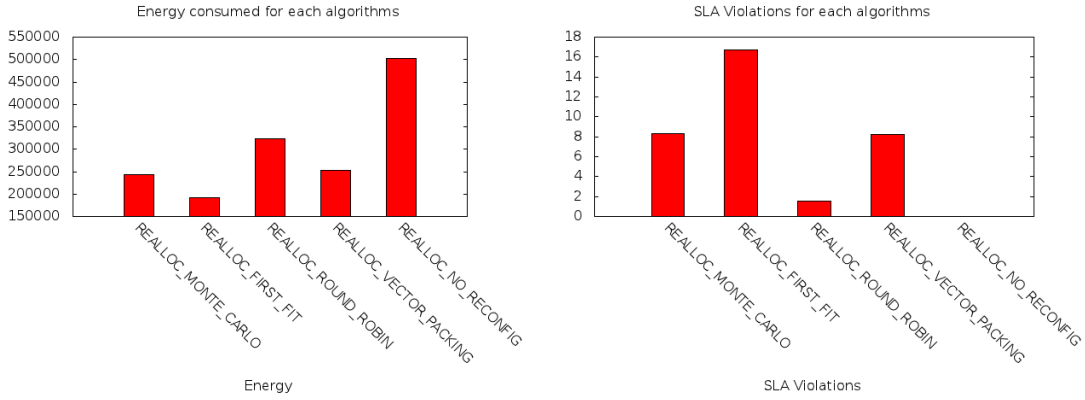
otherwise.

Impact of VM Reconfiguration over Energy Consumption

We did the first set of runs to experiment on the effect of the VM reconfiguration handled by the autonomic manager on the energy consumption of the system. In order to do so, we ran the four algorithms with a fixed workload volatility class (here MEDIUM_HEAVY), and a fixed set of TT pairs. We then compared to the same runs with the autonomic manager disabled. Evaluation parameters can be found in Table 6.6.

Parameter	Evaluated values
Tested workloads	MEDIUM_HEAVY volatility
VM reconfiguration	turned on/off
VM reconfiguration TT pairs	[20%, 40%]
VM reallocation algorithms	RoundRobin, FirstFit, Monte-Carlo, VectorPacking
tt_{cpu}	0.8
tt_{memory}	0.8

Table 6.6: Evaluation input parameters for Experiment 1



(a) Energy consumption

(b) SLA violations

Figure 6.16: Evaluation results for Experiment 1

Figure 6.16a shows the total energy consumption over the 100 time steps. We only plotted one of the no reconfiguration results since all the 4 runs had the same energy consumption. The reason was that since there was an over-provisioning at the first time step, the VM wouldn't change during the whole run. Adding the fact that VMs are provisioned for the CPU at a bit less than the half of the PM's capability, 2 VMs per PM were achieved by all algorithms since it was the optimal initial mapping. We can however add to this the fact that the initial allocation makes the PMs CPU resource loaded, and since our PMs have a low E_{min} compared to E_{max} the

energy consumption difference when disabling the VM reconfiguration is as big as it is. Thus, VM reconfiguration tremendously reduces energy consumption up to 61.6% at the price of more SLA violations as shown in Figure 6.16b.

Evaluation of VM Reallocation Algorithms

In order to evaluate the performance of the VM reallocation algorithms, we ran the simulations for the 4 algorithms, with the VM reconfiguration turned on with one set of TT pairs. The evaluation was made for three different workloads: low and medium-high volatility of the resource needs, and the bioinformatics workflow. Evaluation parameters are shown in Table 6.7.

Parameter	Evaluated values
Tested workloads	LIGHT, MEDIUM_HEAVY, BOKU
VM reconfiguration	turned on
VM reconfiguration TT pairs	[20%, 40%]
VM reallocation algorithms	RoundRobin, FirstFit, MonteCarlo, VectorPacking
tt_{cpu}	0.8
tt_{memory}	0.8

Table 6.7: Evaluation input parameters for Experiment 2

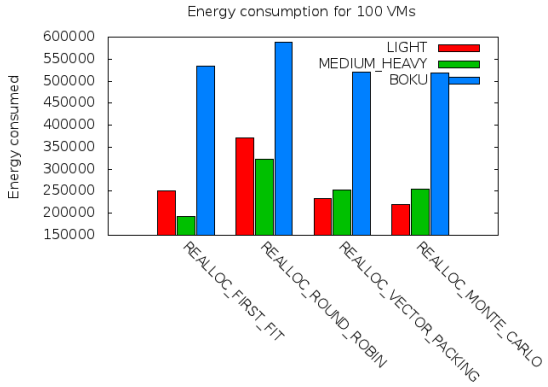
Figure 6.17a shows the total energy consumption of the PMs over the 100 time steps for the 3 workloads. As the figure shows, for LIGHT volatility workloads, the MonteCarlo algorithm performs the best, closely followed by VectorPacking and FirstFit. The RoundRobin algorithms performs badly since it is consuming twice as much energy. If we look at the MEDIUM_HEAVY volatility workload, we can see that the FirstFit algorithm outperforms all other algorithms energy-wise. Finally, for the BOKU workload, which stresses the resources more than the two other workloads, the results are the same as for the LIGHT workload, only with a generally much higher energy consumption.

The reason behind these differences is partially shown in Figure 6.17c, which shows the average number of powered on PMs during the run. As we can see, the RoundRobin will load balance the VMs on every PMs, thus preventing the autonomic manager to shut down empty PMs. The algorithm that performs the best, however, is the VectorPacking algorithm, since it is designed to consolidate heavily the virtual machines, while load balancing if possible on the hosts that remain powered on. We can also note that, except for FirstFit and RoundRobin, the number of powered on PMs increases as the system resource consumption becomes more volatile. This can be explained by the fact that the FirstFit algorithm is less proactive than the others, thus, as we will see in Figure 6.17b leads to some problems.

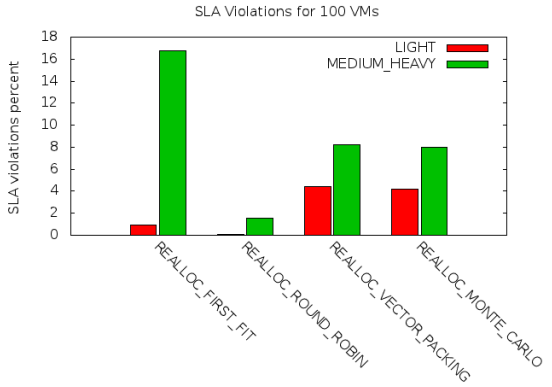
Figure 6.17b plots the SLA violation percentage of the cloud for each algorithms. Only the LIGHT and MEDIUM_HEAVY workloads are plotted, since the BOKU workload is much less volatile than the others and has an SLA violation rate of 0%. As we can see, the RoundRobin has the least violation percentage of all the algorithms, since it uses all the hosts, the small amount of violations there is generated by the VM reconfiguration. The VectorPacking and MonteCarlo algorithms are at around 4% and 8% of SLA violations. Last, the FirstFit

algorithm which is performing better for the LIGHT volatility workload, performs poorly when the volatility increases, since it goes up to over 16% SLA violations.

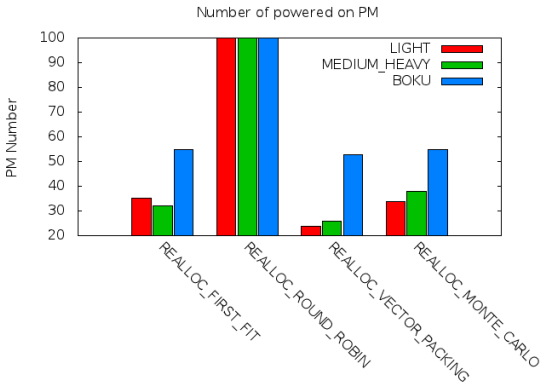
To examine the performance of the algorithms, we have to account for both the energy consumption of the cloud, and the SLA violations that the reconfiguration of the VMs and the PMs have induced. The perfect example is when looking at the FIRSTFIT algorithm for the MEDIUM_HEAVY workload. For these parameters, the algorithm performs extremely well energy-wise, outperforming *smarter* algorithms, but we can see that the setback is to have over 16% SLA violations. Looking at the global picture, we have a 60kW difference between the two algorithms for a 8 point difference of SLA violations.



(a) Energy consumption of the algorithms under different workloads



(b) SLA violation percentages of the VMs under different workloads



(c) Average number of powered on machines

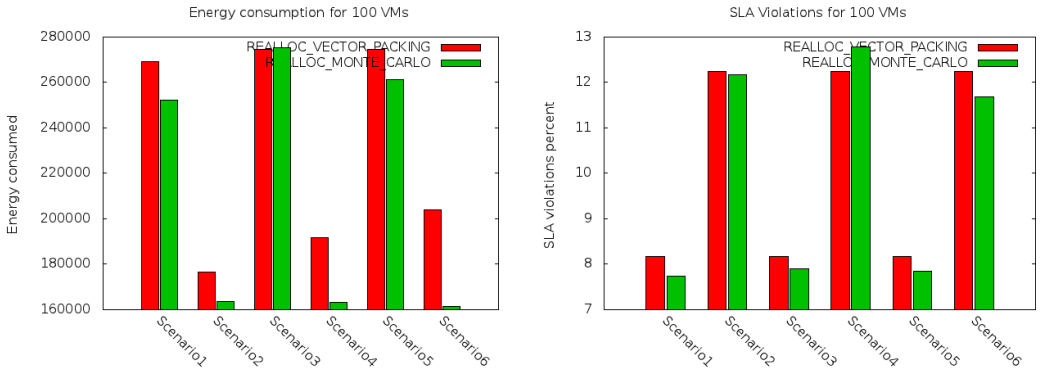
Figure 6.17: Evaluation results for Experiment 2

Evaluation of VM and PM Threat Thresholds

As the next evaluation step we decided to focus on the two threat threshold pairs we use: One for PM power management (average CPU and memory utilization of all PMs), which we call PM-TTs, and one for the VM reconfiguration (TT_{min}, TT_{max}) named VM-TTs. We evaluate them on the two VM reallocation algorithms that achieved best results in previous allocations: MONTECARLO and VECTORPACKING. We analyzed three different PM-TTs $[80\%, 80\%]$, $[60\%, 80\%]$, $[60\%, 60\%]$ in the format $[CPU, memory]$. We used a case with a more cautious TT for CPU, because this showed to be the resource which is usually fluctuating more quickly than memory. For the VM-TTs we used the standard interval $[50\%, 75\%]$ found in [151] to be a good general setting additional to the very cautious setting of $[20\%, 40\%]$ we used in the previous evaluation. All the resulting scenarios are depicted in Table 6.8.

	Scenario 1	Scenario 2	Scenario 3	Scenario 4	Scenario 5	Scenario 6
PM-TTs = [CPU,Memory]	$[80\%, 80\%]$	$[80\%, 80\%]$	$[60\%, 80\%]$	$[60\%, 80\%]$	$[60\%, 60\%]$	$[60\%, 60\%]$
VM-TTs = $[TT_{low}, TT_{high}]$	$[20\%, 40\%]$	$[50\%, 75\%]$	$[20\%, 40\%]$	$[50\%, 75\%]$	$[20\%, 40\%]$	$[50\%, 75\%]$

Table 6.8: Scenarios for Experiment 3



(a) Energy consumption for varying VM and PM thresholds

(b) SLA violations for varying VM and PM thresholds

Figure 6.18: Evaluation results for Experiment 3

As can be seen in Figures 6.18a and 6.18b the results show that MONTECARLO is almost always better in terms of energy and violations. However, it takes much longer processing time as presented in Section 6.7. For Scenarios 2, 4 and 6 the difference in favor of MONTECARLO is extremely large as far as energy consumption is concerned. Not surprisingly, the scenarios, where energy consumption is lowest has the highest number of SLA violations and vice versa. Generally speaking, the even and the odd scenarios show similar behavior meaning the VM-TTs

have a higher impact on the outcome as compared to the PM-TTs. Moreover, lowering PM-TTs increases energy consumption, but does not lower SLA violations in all cases. Finally, the better results of MONTECARLO can also be explained when looking at the number of PMs that were powered on or off. VECTORPACKING powers on at least as much (if not more) PMs as MONTECARLO, and MONTECARLO also spends less energy on powering off again PMs that were unnecessarily powered off by VECTORPACKING.

Scalability

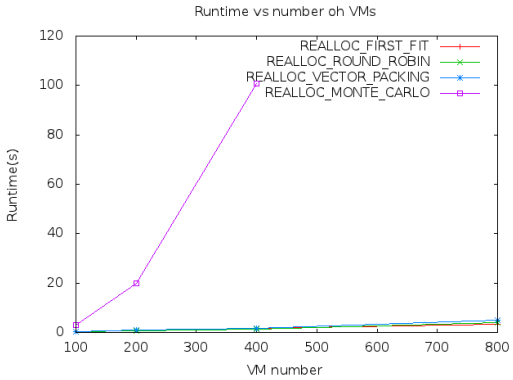


Figure 6.19: Runtime of the algorithms for 100 VMs

Figure 6.19 shows the runtime of the reallocation algorithms for 100, 200, 400 and 800 VMs. These runtimes contain both the VM reconfiguration decisions and the reallocation algorithm. As we can see, the MONTECARLO algorithm is taking six time as much time to compute a solution each time step at 100 VMs, as the others are computing in a reasonable time (around half a second for 100 VMs with the reconfiguration overhead). The MONTECARLO algorithm, even if it performs rather well, will not scale well for two reasons. The first is that it has to compute lots of time the solution (100 in our tests), thus taking more and more time to compute. The second reason is that with an increasing number of VMs and PMs, in order to achieve a near optimal solution every time step, the algorithm has to increase its iteration number to increase the chance of a good solution to emerge. If we increase the number of VMs and PMs without increasing the number of iterations of the MONTECARLO, the quality of the results will become more sporadic, and the average quality of the solution will decrease.

Figure 6.19 shows that the MONTECARLO algorithm is not scalable, unlike the other 3 for which the runtime seems to grow linearly to the VM number, and that it takes around an acceptable 5 seconds to compute a solution for 800 VMs.

CHAPTER 7

Knowledge Management for Cloud Federations

In this chapter we use an existing inter-Cloud architecture [107] and analyze a possible extension with knowledge management. We formalize elements of the Cloud federation architecture, and show the feasibility of this extension by pointing out concrete implementation samples.

7.1 Federated Cloud Management Architecture

Figure 7.1 shows the Federated Cloud Management (FCM) architecture (first introduced in [143]), and its connections to the corresponding components that together represent an interoperable solution for establishing a federated cloud environment. Using this architecture, users are able to execute services deployed on cloud infrastructures transparently, in an automated way. Virtual appliances for all services should be stored in a generic repository called FCM Repository, from which they are automatically replicated to the native repositories of the different Infrastructure as a Service cloud providers.

Users are in direct contact with the *Generic Meta-Broker Service* (GMBS – [112]) that allows requesting a service by describing the call with a WSDL, the operation to be called, and its possible input parameters. The GMBS is responsible of selecting a suitable cloud infrastructure for the call, and submitting to a CloudBroker (CB) in contact with the selected infrastructure. Selection is based on static data gathered from the *FCM Repository* (e.g. service operations, WSDL, appliance availability), and on dynamic information of special deployment metrics gathered by the CloudBrokers (see Section 7.2). The role of GMBS is to manage autonomously the interconnected cloud infrastructures with the help of the CloudBrokers by forming a cloud federation.

CloudBrokers are set up externally for each IaaS provider to process service calls and manage VMs in the particular cloud. Each *CloudBroker* [142] has its own queue for storing the incoming service calls, and it manages one virtual machine queue for each virtual appliance (VA). Virtual machine queues represent the resources that can currently serve a virtual appliance

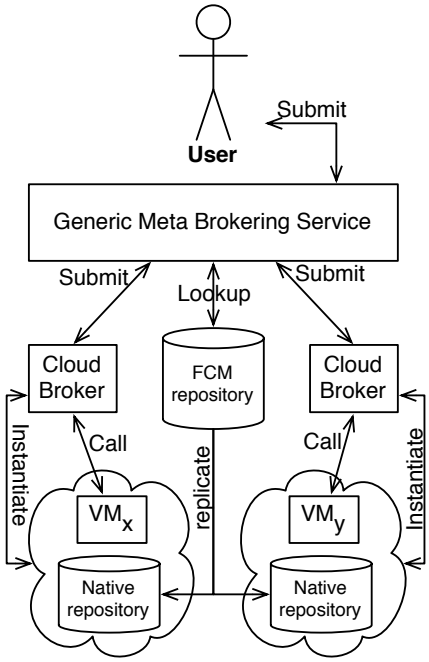


Figure 7.1: The original Federated Cloud Management architecture

specific service call. The main goal of the CloudBroker is to manage the virtual machine queues according to their respective service demand. The default virtual machine scheduling is based on the currently available requests in the queue, their historical execution times, and the number of running VMs.

Virtual Machine Handlers are assigned to each virtual machine queue and process the VM creation and destruction requests in the queue. Requests are translated and forwarded to the underlying IaaS system. VM Handlers are infrastructure-specific and built on top of the public interfaces of the underlying IaaS. Finally, the CloudBroker manages the incoming service call queue by associating and dispatching calls to VMs created by the VM Handler.

As a background process, the architecture organizes virtual appliance distribution with the *Automatic Service Deployment* (ASD) component [107]. This component minimizes pre-execution service delivery time to reduce the apparent service execution time in highly dynamic service environments. Service delivery is minimized by decomposing virtual appliances and replicating them according to demand patterns, then rebuilding them on the IaaS system that will host the future virtual machine. This chapter does not aim to further discuss the behavior of the ASD, however it relies on its features that reduce virtual appliance replication time and transfer time between the FCM and the native repositories.

7.2 Self-adaptable Inter-Cloud Management Architecture

This chapter offers two options to incorporate the concepts of knowledge management (KM) systems into the Federated Cloud Management architecture: local and global. Local integration is applied on a per deployed component basis, e.g. every CloudBroker utilizes a separate KM system for its internal purposes. In contrast, global integration is based on a single KM system that controls the autonomous behavior of the architectural components considering the available information from the entire cloud federation. In this section first we discuss which integration option is best to follow, then we introduce the extensions made to a KM system in order to perform the integration.

Knowledge Management Integration Options

When *local* integration is applied, each knowledge manager can make fine-grained changes – e.g., involving actions on non-public interfaces – on its controlled subsystem. First, the meta-broker can select a different scheduling algorithm if necessitated by SLA violation predictions. Next, the CloudBroker can apply a more aggressive VM termination strategy, if the greenness of the architecture is more prioritized. Finally, if the storage requirements of the user are not valid any more, the FCM repository removes unnecessarily decomposed packages (e.g. when the used storage space approaches its SLA boundaries, the repository automatically reduces the occupied storage). However, the locally made reactions to predicted SLA violations might conflict with other system components not aware of the applied changes. These conflicts could cause new SLA violation predictions in other subsystems, where new actions are required to maintain the stability of the system. Consequently, local reactions could cause an *autonomic chain reaction*, where a single SLA violation prediction might lead to an unstable system.

To avoid these chain reactions, we investigated *global* integration (presented in Figure 7.2) that makes architecture-wide decisions from an external viewpoint. High-level integration is supported by a monitoring solution – deployed next to each subcomponent in the system (GMBS, the various CloudBrokers and repositories) – that determines system behavior in relation to the settled SLA terms. Global KM integration aggregates the metrics received from the different monitoring solutions, thus operates on the overall architecture and makes decisions considering the state of the entire system before changing one of its subsystems. However, adaptation actions are restricted to use the public operations of the FCM architecture (e.g., new cloud selection requests, new VM and call associations or repository rearrangements). Consequently, the global integration *exhausts adaptation actions* earlier than the local one, because of metrics aggregation and restricted interface use. For instance, if aggregated data hides the cause of a possible future SLA violation, then global KM cannot act without user involvement.

In this chapter, we propose to use a *hybrid* KM system (revealed in Figure 7.3) combining both global and local integration options. The hybrid system avoids the disadvantages of the previous solutions by enabling global control over local decisions. In our system, local actions can be preempted by the global KM system by propagating predicted changes in aggregated metrics. Based on predicted changes, the global KM could stop the application of a locally optimal action and prevent the autonomic chain reaction that would follow the local action. On the other hand, if the global system does not stop the locally optimal action, then it enables the

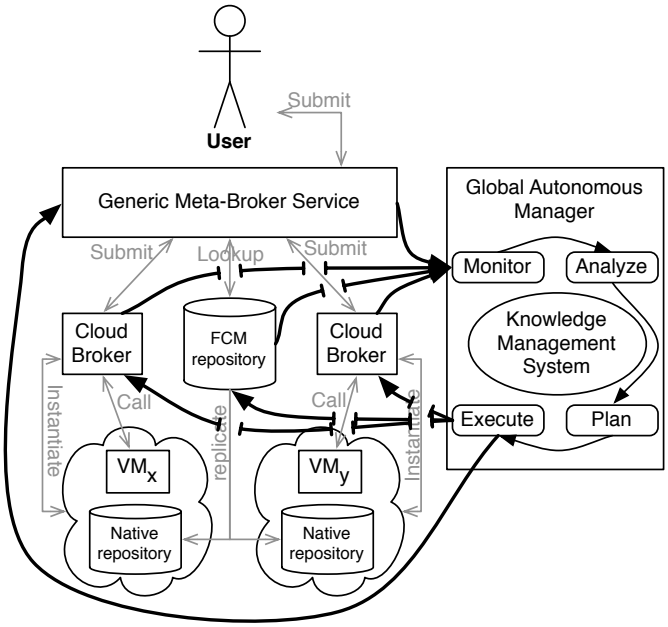


Figure 7.2: Global integration of the knowledge management system

execution of more fine-grained actions postponing adaptation action exhaustion.

Knowledge Management System Extensions

This subsection first lists the possible autonomic actions in our KM system, then it analyzes the collected monitoring data that can indicate the need for autonomous behavior. Finally, based on these indicators, we conclude with the rules triggering the adaptation in our FCM components.

Actions

Based on the affected components, the architecture applies four basic types of actions on unacceptable behavior. First, at the meta-brokering level, the system can organize a *rescheduling of several service calls*. E.g., the autonomous manager could decide to reschedule a specific amount of queued calls – $c \in Q_x$, where c refers to the call, and Q_x specifies the queue of the CloudBroker for IaaS provider x. Consequently, to initiate rescheduling, the knowledge manager specifies the amount of calls (N_{cr}) to be rescheduled and the source cloud (C_s) from which the calls need to be removed. Afterwards, the meta-broker evaluates the new situation for the removed calls, and schedules them to a different cloud, if possible.

Second, at the level of cloud brokering, the system could decide either to *rearrange the VM queues* of different CloudBrokers, or alternatively to *extend or shrink the VM queue* of a specific CloudBroker. *VM queue rearrangement* requires global KM integration in the system so it can determine the effects of the queue rearrangement on multiple infrastructures. The autonomous manager accomplishes rearrangement by destructing VMs of a particular virtual appliance in a

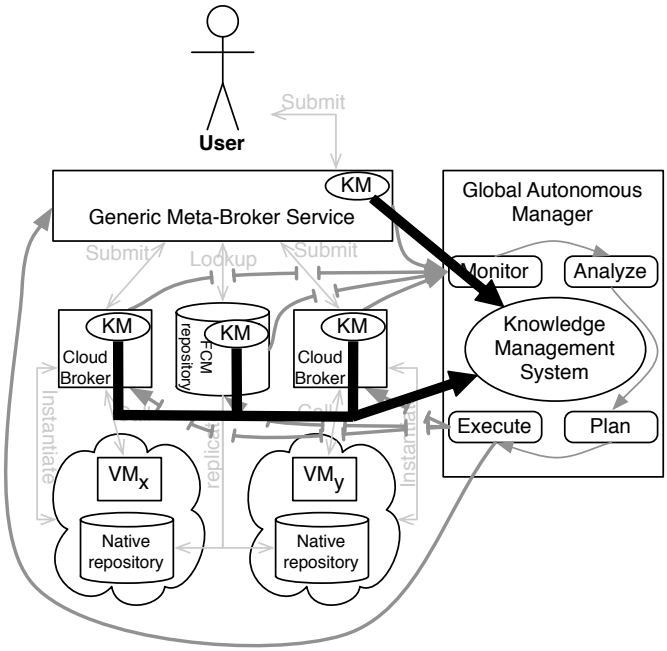


Figure 7.3: Hybrid integration of the knowledge management system

Action	Involved component	Integration
Reschedule calls	Meta-Broker	Global
Rearrange VM queues	CloudBroker	Global
Extend/Shrink VM Queue	CloudBroker	Local
Rearrange VA storage	FCM repository	Global
Self-Instantiated Deployment	Service instances	Local

Table 7.1: Summary of Autonomous Actions

specific cloud and requesting new VMs in another one. Consequently, the autonomous manager selects the virtual appliance (VA_{arr}) that has the most affected VMs. Then it identifies the amount of virtual machines (N_{vmtr}) to be removed from the source cloud (C_s) and instantiated in a more suitable one (C_d).

The queue rearrangement operations have their counterparts also in case of local KM integration. The VM *queue extension and shrinking* operations are local decisions that are supported by energy efficiency related decisions. In case of queue shrinking, some of the virtual machines controlled by local CloudBroker are destructed. However, under bigger loads, virtual machines could be in the process of performing service calls. Therefore, the autonomous manager can choose between the three VM destruction behaviors embedded into the CloudBrokers: (i) destroy after call completed, (ii) destroy right after request and put the call back to the local service call queue and finally, (iii) destroy right after request and notify the user about call abortion. As

a result, the autonomous manager specifies the number of VMs to extend (N_{ex}) or shrink (N_{shr}) the queue with and the destruction strategy (S_{dest}) to be used.

Third, on the level of the FCM repository, the autonomous manager can make the decision to *rearrange virtual appliance storage* between native repositories. This decision requires the FCM repository either to remove appliances from the native repositories, or to replicate its contents to a new repository. Appliance removal is only feasible, if one of the following cases are met: (i) the hosting cloud will no longer execute the VA, (ii) the hosting cloud can download the VA from third party repositories or finally, (iii) the appliance itself was based on an extensible appliance that is still present in the native repository of the hosting cloud. The objective of the rearrangement is to reduce the storage costs in the federation at the expense of increased virtual machine instantiation time for VMs of the removed appliances. Conclusively, the rearrangement decision should involve the decision on the percentage (N_{repr}) of the reduced or replicated appliances that should participate in the rearrangement process.

Finally, when virtual appliances are built with embedded autonomous capabilities (internal monitoring, KM system etc.), then virtual machines based on them are capable of *self-initiated deployment*. If a service instance gets either overloaded or dysfunctional according to its internal monitoring metrics, then the instance contacts the local CloudBroker to instantiate a new virtual machine just like the instance is running in. In case of overloading, the new instance will also be considered for new *Call→VM* associations. In case of dysfunctional instances, the system creates a proxy service inside the original VM replacing the original service instance. This proxy is then used to forward the requests towards the newly created instance until the current VM is destructed.

Monitored Metrics

After analyzing the various autonomous actions that the KM system can exercise, we investigated the monitoring system and the possible metrics to be collected for the identification of those cases when the architecture encounters unsatisfactory behavior. Currently, we monitor and analyze the behavior of CloudBrokers, the FCM repository and individual service instances.

Since CloudBrokers represent the behavior of specific IaaS systems, most of the measurements and decisions are made based on their behavior. All measurements are related to the queues of the CloudBroker; therefore we summarize their queuing behavior. CloudBrokers offer two types of queues: the call queue (Q_x, where x identifies the specific CloudBroker that handles the queue) and the VM queues ($VMQ_{x,y}$, where y identifies the specific service – or appliance VA_y – the queued VMs are offering). The members of the call queue represent the service calls that a CloudBroker needs to handle in the future (the queue is filled by the meta-broker and emptied by the CloudBroker through associating a call with a specific VM). On the other hand, VM queues are handled on a more complex way: they list the currently handled VMs offering a specific service instance. Consequently, the CloudBrokers maintain VM queues for all service instances separately. Entries in the VM queues are used to determine the state of the VMs:

$$State : VM \rightarrow \begin{cases} WAITING \\ INIT \\ RUNNING.AVAILABLE \\ RUNNING.ACQUIRED \\ CANCEL \end{cases} \tag{7.1}$$

- **Waiting:** the underlying cloud infrastructure does not have resources to fulfill this VM request yet.
- **Init:** the VM handler started to create the VM but it has not started up completely yet.
- **Running and available:** the VM is available for use, the CloudBroker can associate calls to these VMs only.
- **Running and acquired:** the VM is associated with a call and is processing it currently.
- **For cancellation:** the CloudBroker decided to remove the VM and stop hosting it in the underlying infrastructure.

Based on these two queues the monitor collects the metrics listed in the following paragraphs.

To support decisions for *service call rescheduling*, the system monitors the call queue length for all available CloudBrokers for a specific service call s:

$$q(x, s) := \{c \in Q_x : (type(c) = s)\}, \tag{7.2}$$

where $type(c)$ defines the kind of the service call c is targeting.

Call throughput measurement of available CloudBrokers is also designed to assist *call rescheduling*:

$$throughput(x) := \frac{1}{max_{c \in Q_x}(waitingtime(c))}, \tag{7.3}$$

where $waitingtime(c)$ expresses the time in sec a service call has been waiting in the specific Q.

We define the average waiting time of a service s by

$$awt(s, Q_x) := \frac{\sum_{c \in q(x,s)} waitingtime(c)}{|q(x, s)|}, \tag{7.4}$$

and the average waiting time of a queue by

$$awt(Q_x) := \frac{\sum_{c \in Q_x} waitingtime(c)}{|Q_x|}. \tag{7.5}$$

To distinguish the CloudBrokers, where *VM queue rearrangements* could occur, we measure the number of service instances that are offered by a particular infrastructure:

$$vms(x,s) := \{vm \in VMQ_{x,s} : State(vm) = RUNNING.AVAILABLE \vee State(vm) = RUNNING.ACQUIRED\} \quad (7.6)$$

The call/VM ratio for a specific service managed by a specific CloudBroker:

$$cvmratio(x,s) := \frac{|q(x,s)|}{|vms(x,s)|} \quad (7.7)$$

This ratio allows the global autonomous manager to plan *VM queue rearrangements* and equalize the service call workload on the federated infrastructures. When applied with the local KM system, this ratio allows the system to decide on *extending and shrinking the VM queues* of particular services and balance the service instances managed by the local CloudBroker.

The load of the infrastructure managed by a specific CloudBroker:

$$load(x) := \frac{\sum_{\forall s} |vms(x,s)|}{\sum_{\forall s} |VMQ_{x,s}|} \quad (7.8)$$

The load analysis is used for *VM queue rearrangements* in order to reduce the number of waiting VMs in the federation. When applied locally, along with the call/vm ratio the load analysis is utilized to determine when to *extend or shrink the VM queues* of various services. As a result, CloudBrokers could locally reorganize their VM structures that better fit the current call patterns.

To support the remaining autonomous actions, the FCM repository and individual service instances are also monitored. First, the system monitors the accumulated storage costs of a virtual appliance in all the repositories ($r \in R$) in the system (expressed in US dollars/day):

$$stcost(VA_s) := \sum_{\forall r} locstcost(r, VA_s), \quad (7.9)$$

where $locstcost(r, VA_s)$ signifies the local storage cost at repository r for appliance VA_s (representing a specific service referred as s). To better identify the possible appliance storage rearrangements the system also analyzes the usage rate of appliances in the different repositories expressed in the number of times the VMs based on the appliance have changed status from $INIT$ to $RUNNING.AVAILABLE$ in a single day ($deployfreq(r, VA_s)$).

Finally, individual services are monitored to support self-instantiated deployment. Here we analyze the service availability (expressed as the % of time that the instance is available for external service calls) of the specific service instance deployed in the same VM where the monitoring system is running.

Basic Rules for Applying Actions

We decided to formulate the knowledge base (KB) as a rule-based system. Rules are of the form "WHEN condition THEN action" and can be implemented e.g. using the Java rule engine

1 rule “Reschedule calls”
2 **WHEN**
3 $C_s : Cloudbroker()$
4 $throughput(x) < mean(throughput(.)) + \delta \cdot std(throughput(.))$
5 **THEN**
6 $C_d := \arg\max throughput(.)$
7 $N_{cr} := equalizeQs(C_s, C_d)$
8 $calls := remove(N_{cr}, C_s)$; //removes last N_{cr} entries in Q_{C_s}.
9 $add(calls, C_d)$;

Figure 7.4: Rule for rescheduling calls

Drools [15]. We define several rules based on the previously defined measurements and actions, and present them in Drools-related pseudo code. The working memory of the KM system, which is the main class for using the rule engine at runtime, does not only consist of the specified rules, but also of the objects whose knowledge has to be modeled, and that are currently active in the Cloud federation (like a CloudBroker, the native repository, different queues, etc.). These objects are typically modeled as Java classes, and thus referred to as CloudBroker(), NativeRepository(), etc.

Figure 7.4 shows the rule for *rescheduling service calls*. Line 1 states the unique name the rule can be identified with in the KB. This way, rules can be dynamically altered or replaced if different global behavior due to changing high-level policies (i.e., changing from energy efficient to SLA performant) is required. Lines 3-4 state the conditions that have to be fulfilled to trigger the actions in lines 6-9. At first, we look for a CloudBroker C_s (line 3), whose throughput falls below the average of all the queues' throughputs ($mean()$) plus a multiple of their standard deviation ($std()$, line 4). If such C_s is found, the rule is executed. We have to decide to which Cloud C_d (line 6) to move N_{cr} service calls (line 7), and finally invoke the appropriate public interface methods of the Cloud brokers at stake (lines 8-9). As C_d we choose the Cloud with maximum throughput. The equalizeQs() method (line 7) tries to equal out the average waiting times of the queues of C_s and C_d. It takes the last service call $\hat{s}$ out of Q_s, retrieves its average waiting time $awt(\hat{s}, Q_s)$ and calculates the new estimated average waiting time for Q_s and Q_d by $awt(Q_s) := awt(Q_s) - awt(\hat{s}, Q_s)$ and $awt(Q_d) := awt(Q_d) + awt(\hat{s}, Q_d)$, respectively. Then it adds $\hat{s}$ to Q_d. It continues this procedure as long as $awt(Q_s) \geq awt(Q_d)$, and returns the number of service calls that have been hypothetically added to Q_d. The rule could then either really add the chosen calls to C_d as presented in line 9, or return them to the meta-broker

Figures 7.5 and 7.6 show possible rules for removing VAs from a Cloud's native repository due to high local or global costs, respectively. Both rules try to find a repository r and a VA VA_x that have been inserted into the working memory of the rules engine (lines 3-4), and remove the specified VA from the repository (line 8), when certain conditions hold. In Figure 7.5 the removal action is executed when two conditions hold: First, the local storage cost of the VA at the specified resource exceeds a certain threshold. The threshold is calculated as the average local storage costs at all repositories for the same VA plus a multiple of its standard deviation. Second, the deployment frequency of the VA at this repository falls below a certain threshold,

1 rule “Remove VA from native repository due to high local costs”
2 **WHEN**
3 r : NativeRepository()
4 VA_x : VirtualAppliance()
5 $locstcost(r, VA_x) > mean(locstcost(., VA_x)) + \delta \cdot std(locstcost(., VA_x))$
6 $deployfreq(r, VA_x) < mean(deployfreq(., VA_x))$
7 **THEN**
8 $remove(VA_x, r)$ //removes VA_x from native repository r

Figure 7.5: Rule for removing VA from native repository of a specific Cloud due to high local costs

1 rule “Remove VA from native repository due to high global costs”
2 **WHEN**
3 r : NativeRepository()
4 VA_x : VirtualAppliance()
5 $stcost(VA_x) > mean(stcost(.)) + \delta \cdot std(stcost(.))$
6 $r_{min} : \arg\min deployfreq(., VA_x)$
7 **THEN**
8 $remove(VA_x, r_{min})$ //removes VA_x from native repository r_{min}

Figure 7.6: Rule for removing VA from native repository of a specific Cloud due to high global costs

which is the mean deployment frequency of the VA at all repositories. In short, the VA is instantiated less often than other VAs, but its cost is higher than for other VAs, so the VA should be removed. Figure 7.6 takes a global perspective and checks whether the overall storage cost for the VA exceeds a certain threshold (defined similarly as with Figure 7.5, line 5). Then, the VA is removed from the repository that has the lowest deployment frequency (line 6).

The remaining rules can be specified according to the actions and measurements as explained before. However, their specific parameters may have heavy impact on the overall performance of the system. These parameters are to be learned by the KM system. In our future work, we plan to evaluate the system performance with the extension of the simulation engine presented in [150].

CHAPTER 8

State of the Art

This chapter follows in principal the organization of this thesis. In Section 8.1 we describe work related to adaptive SLA mapping from Chapter 3. Sections 8.2, 8.3, and 8.4 describe related work on resource-efficiency, knowledge management and self-adaptive algorithms, respectively (cf. Chapter 4). Section 8.5 covers related work on energy efficiency from Chapter 5, and Section 8.6 focuses on Cloud federations from Chapter 7. Finally, Section 8.7 concludes with other holistic Cloud management projects.

8.1 SLA Generation and Adaptive SLA Mapping

For putting our work on adaptive SLA mapping in context of the state of the art, we describe Cloud resource management, Cloud marketplaces, and the existing work on SLA matching.

Cloud Resource Management

There is a large body of work about managing resource provisions, negotiations, and federation of Cloud and Grid resources. An example is [67]. They designed an agent technology to address the federation problems in Grids, i.e., resource selection and policy reconciliation. [186] propose a new abstraction layer for managing the life cycle of services. It allows automatic service deployment and escalation depending on the service status. This abstraction layer can be positioned on top of different Cloud provider infrastructures, hence mitigating the potential lock-in problem and allowing the transparent federation of Clouds for the execution of services. [87] investigate three novel heuristics for scheduling parallel applications on utility Grids, optimizing the trade-off between time and cost constraints.

However, most of the related work on resource management considers resource provision from the provider's point of view and does not consider Cloud computing infrastructures in the context of a marketplace.

Cloud Market

Currently, a large number of commercial Cloud providers have entered the utility computing market, offering a number of different types of services. These services can be grouped into three types: computing infrastructure services, which are pure computing resources on a pay-per-use basis [183, 12, 5]; software services, which are computing resources in combination with a software solution [7, 10]; and platform services, which allow customers to create their own services in combination with the help of supporting services of the platform provider. The first type of services, which is also called Infrastructure-as-a-Service (IaaS) consists of a virtual machine, as in the case of Amazon's EC2 service, or in the form of a computing cluster, as done by Tsunamic Technologies. The number of different types of virtual machines offered by a provider is low. For example, Amazon and EMC introduced only three derivations of their basic resource type [3]. Examples for the second type of services, which are called Software-as-a-Service (SaaS) are services offered by Google (Google Apps [7]) and Salesforce.com [10]. These companies provide access to software on pay-per-use basis. These SaaS solutions can hardly be integrated with other solutions, because of their complexity. Examples for the third kind of Cloud services, which are called Platform-as-a-Service (PaaS), are Sun N1 Grid [11], force.com [10], and Microsoft Azure [9]. In this category, the focus lies on provisioning essential basic services that are needed by a large number of applications. These basic services can be ordered on a pay-per-use basis. Although the goal of the PaaS service offerings is a seamless integration with the users' applications, standardization of interfaces is largely absent. Furthermore, big Cloud providers as the mentioned Azure or EC2 do not even provide their SLAs in a standardized format, e.g., XML. If they want to participate in markets with higher liquidity, as leveraged by our approach, they have to comply to the market rules and formalize their SLA templates in a machine-readable way. Nevertheless, the implementation of system resource markets has been discussed in several projects [57, 165, 167]. [203] give an overview over information systems for traded resources in Grid markets and [93] deal with economic models of Grid computing markets. All in all, however, mentioned works either do not define the tradable goods, work with very simplified definitions, or do not take market liquidity into account.

Service Level Agreement Matching

The main SLA matching mechanisms are based on OWL, DAML-S, or similar semantic technologies. [168] describe a framework for semantic matching of SLAs based on WSDL-S and OWL. [91] describes another onotology-based approach based on OWL and SWRL. [75] present a unified QoS ontology applicable to specific scenarios such as QoS-based Web services selection, QoS monitoring, and QoS adaptation. [34] present an autonomic Grid architecture with mechanisms for dynamically reconfiguring service center infrastructures. It is exploited to fulfill varying QoS requirements. Besides those ontology-based mechanisms, [118] discuss autonomous QoS management, using a proxy-like approach for defining QoS parameters that a service has to maintain during its interaction with a specific customer. The implementation is based on WS-Agreement, using predefined SLA templates. However, these templates cannot consider changes in user needs, which is essential for creating successful markets, as shown in our earlier work [184]. Additionally, several works on SLA management have been presented

in [58]. Besides, regardless of the type of the used approach, these approaches do not evaluate and explain the benefit and costs through the introduction of SLA matching mechanisms.

In [209] Yarmolenko et al. make a case for increasing the expressiveness of SLAs. Doing this, they can possibly also increase market liquidity, when it comes to matching asks and bids, where a same understanding of the parameters has already been established. Our approach could be seen as complimentary in the sense that it makes sure that their pre-condition holds.

8.2 Resource-Efficient SLA Enactment

Apart from adaptive SLA mappings and SLA generation, we have determined six different ways to compare our work with other achievements in this area. Whereas this section compares our work with other works dealing with SLA enactment and resource efficiency, Section 8.3 considers the area of knowledge management, Section 8.4 highlights self-adaptive approaches, Section 8.5 focuses on energy efficiency, Section 8.6 on Cloud federations, and Section 8.7 more generally relates the FoSII project to other projects in this field.

As to resource-efficient SLA enactment, most works aim at optimizing resource usage while keeping QoS goals. However, we can identify six categories that present shortcomings of related work in this area. In the following list we give examples of work falling into these categories. A more detailed description of related work can be found thereafter.

(i) Work with no proactive SLA enactment [206]

(ii) Work related to Grid computing or SOA in general [170, 188, 208]

(iii) Work just tied to specific SLA parameters or use cases [115, 175, 46, 113, 179, 66]

(iv) Work without holistic view [213, 210, 160, 157]

(v) Work neglecting the overhead of reallocation actions [179]

(vi) Work without VM reconfiguration only considering static workloads [200, 47]

Several papers concentrate on specific subsystems of large-scale distributed systems, as [115] on the performance of memory systems, or only deal with one or two specific SLA parameters. Petrucci [175] or Bichler [46] investigate one general resource constraint and Khanna [113] only focuses on response time and throughput. [66] describe in detail the process of how to fulfill an SLA, which is limited to only one SLO and the analysis of this resource provisioning is closely tied to a special resource, i.e., CPU utilization. A lot of work under this aspect [170, 188, 208] has been carried out on Grids, which, however, have a different architecture than Clouds. Related work in Grid computing uses job finishing and start times for scheduling. This is not applicable in Cloud Computing, since Cloud applications do not necessarily have start or finishing times, but run for an unspecified amount of time as web or database servers.

A quite similar approach to our concept is provided by the Sandpiper framework [206], which offers black-box and gray-box resource management for VMs. Contrary to our approach, though, it plans reactions just after violations have occurred. Also the VCONF model by Rao et al. [179] has similar goals as presented in Section 1.1, but depends on specific parameters, can

only execute one action per iteration and neglects the energy consumption of executed actions. Hoyer et al. [99] also undertake a speculative approach as in our work by overbooking PM resources. They assign VMs to PMs that would exceed their maximum resource capacities, because VMs hardly ever use all their assigned resources. Computing this allocation they also take into consideration workload correlation of different VMs. Zhang et al. [213] optimize revenue for a single Cloud provider by adapting the number of specific VM types that should be available for auctioning as practiced on Amazon EC2 [3]. They also experiment with the price for VM types and use model predictive control to find solutions [84]. However, none of the presented papers use a KB for recording past actions and learning. Those, which do, are presented in Section 8.3.

Other papers neglect VM reconfiguration or the dynamic nature of Cloud workloads. [210, 160] solely focus on VM migration and [157] on turning on and off physical machines, whereas we also focus on VM re-configuration. Borgetto et al. [47] tackle the trade-off between consolidating VMs on PMs and turning off PMs on the one hand, and attaining SLOs for CPU and memory on the other. However, the authors assume a static setting and do not consider dynamically changing workloads. So, e.g., they do not take the number of migrations into account. Stillwell et al. [195] in a similar setting define the resource allocation problem for static workloads, present the optimal solution for small instances and evaluate heuristics by simulations. Nathani et al. [164], e.g., also deal with VM placement on PMs using scheduling techniques. [101] react to changing workload demands by starting new VM instances; taking into account VM startup time, they use prediction models to have VMs available already before the peak occurs. Rego et al. [180] allocate VMs to PMs based on the CPU capacity. They take into account a variety of CPU types in a heterogeneous Cloud setting, and achieve the allocation by introducing a novel representation of the processing capacity. Sugiki et al. [196] follow a resource allocation approach for virtualization based on common resource allocation techniques used for operating systems. Watson et al. [205] relate CPU allocation for a VM to the response time of an application and create a probabilistic model to predict response time. Kephart et al. [109] argue for avoiding a system based on action or goal policies, and opt for a utility-driven approach, where they give a detailed view on how to derive utility functions. However, it would be interesting to develop an automatic mapping of general SLAs to these utility functions, because as in [66], the authors only deal with one SLA parameter, response time, and relate it to the number of servers they use for satisfying a certain consumer load. Thus, the only actions to execute are *shut down server* and *start server*. Muthusamy et al.'s vision [163] is quite similar to our goals, but set in the more general field of Service Oriented Architectures (SOA). They present a methodology to optimize workflow execution by reducing communication effort in terms of exchanged messages between different servers. Their optimization routines are based on the declarative specification of parameters in SLAs, and they also aim at attaining the SLAs by efficiently utilizing resources.

Complementary work to ours has been carried out by Verma et al. [201], who study the impact VM reconfiguration and VM live migration have on application performance. They focus on VM migration, where they predict its duration. A state of the art survey has been conducted by [199], who compare application scalability that has been achieved by different approaches. As an application may work with several VMs as a database server for instance, application scalability also involves the relationship between several VMs. [186] propose a single controller for the

whole application that exploits user defined rules to add or remove certain VMs to achieve so called *elasticity*. An *elasticity controller* has sensors that gather information of the infrastructure and application performance, and actuators that use the API of an IaaS provider to change VM configuration. Baladine et al. [38] deal with network scalability based on applications, and Lim et al. [137] on elastic storage.

Resource allocation has also been observed in different Cloud settings. Distefano et al. [74] bring together volunteer and Cloud computing and deal with the resource management thereof. [144] built a resource manager based on the Nimbus toolkit [21]. This resource manager extends a cluster by using public Cloud resources when necessary. Also [72] evaluated the costs and benefits of such an approach. [145] also builds on Nimbus, and tries to increase utilization of an IaaS Cloud in certain cases. The authors exploit the fact that private Cloud providers have to keep the utilization of their infrastructure low such that they can provide computing power on-demand and do not have to reject spontaneously incoming requests due to the lack of available resources at a given moment. They propose to use the therefore unutilized resources by scientific applications such as SETI@Home [31] or Folding@Home [128] that do not rely on on-demand access, but are designed to opportunistically exploit available resources, whose usage can be terminated at any time. [77] also target the underutilization of Clouds and focus on the response time and latency of a service using load balancing in EC2 [3]. Sridharan et al. [194] focus on virtual desktop clouds [116], but also use an allocation strategy that is based on cost-awareness and utility. Pan et al. [172] present a toolkit, which allows users to build their own private Cloud out of a cluster of PMs. Via a web interface, the user submits jobs, and the middleware allocates VMs to execute the jobs. Our design of a Cloud can also handle applications that have no specific end time, but run continuously on a Cloud infrastructure.

Summarizing we can say that there has been a great deal of work on the different escalation levels, whereas VM configuration has not been observed yet, nor its combination with other escalation levels.

8.3 Knowledge Management and Autonomic Computing in Clouds and Related Fields

We devise this section into four areas: First, we present state-of-the-art KM techniques used in Cloud computing, SLA management, and related fields. Second, we take a broader perspective and have a look on KM techniques in general. Third, we point out advances in autonomic computing in Clouds and related fields; and fourth, we discuss other simulation engines that evaluate KM and autonomic computing techniques.

First, there has been work on KM of SLAs, especially rule-based systems. Paschke [174] et al. look into a rule-based approach in combination with the logical formalism ContractLog [173]. This approach specifies rules to trigger after a violation has occurred, but it does not deal with avoidance of SLA violations. A similar methodology has been taken by Kyas et al. [125], who monitor SLAs and enforce penalties in case of violations. The SLAs have to be written in a specific action-based formal language called CL, which allows to write conditional obligations, permissions and prohibitions over actions. Hasselmeyer et al. [95] introduce a Conversion Factory, which on a design level combines the SLA, the system status, and the Business Level

Objectives to create Operational Level Agreements (OLAs), which govern system configuration. Whereas the idea seems promising, there are no details on how to achieve these mappings to OLAs. Others inspected the use of ontologies as KBs only at a conceptual level. [123, 122] viewed the system in four layers (i.e., business, system, network and device) and broke down the SLA into relevant information for each layer, which had the responsibility of allocating required resources. Again, no details on how to achieve this have been given.

More similar to our approach presented in Section 4.4 Bahati et al. [36] also use policies, i.e., rules, to achieve autonomic management. They provide a system architecture including a KB and a learning component, and divide all possible states of the system into so called regions, which they assign a certain benefit for being in this region. A bad region would be, e.g., response time > 500 (too slow), a fair region would be response time < 100 (too fast, consuming unnecessary resources), and a good region would be $100 \leq$ response time ≤ 500. With reward signals from the given metrics, the system learns whether different actions for one state were good or not. Yet, the actions are not structured, but are mixed together into a single rule, which makes the rules very hard to manage and to determine a salience concept behind them. Nevertheless, we share the idea of defining "over-utilized", "neutral" and "under-utilized" regions. As in some previously mentioned papers, that work deals with only one SLA parameter and a quite limited set of actions, and with violations and not with the avoidance thereof. Our KM system allows to choose any arbitrary number of resource parameters that can be adjusted on a VM. Moreover, our approach is more wholesome than related work and integrates the different action levels that work has been carried out on.

In several papers Yousif et al. [114, 207] present autonomic resource management as far as power consumption is concerned by using fuzzy logic containing IF-THEN rules, for instance. In [68] Choi et al. use a learning module also based on CBR, but for VM migration decisions. These decisions are based on CPU utilization and the standard deviation thereof after and before the migration takes place. Cases are assumed to be the same when the current standard deviation and the CPU utilization of the PM are the same. A migration is supposed to be useful if the standard deviation of the CPU utilization after migration of the previous case is less than the current CPU utilization. The authors store the mentioned data into data vectors that form a so-called history matrix. Additionally to the CBR approach, there is also the approach of Case-based planning (CBP) [69], which transforms an initial state into a goal state by applying actions. CBP is very similar to Situation Calculus in its initial description – and due to the same reasons as described in Section 4.1 we decided not to apply CBP for Cloud computing –, but it searches for the actions to be applied in a different way. CBP uses past experience to see if a specific action was helpful to advance the state towards the goal state or not. Berral et al. [44] use machine learning techniques to schedule jobs on clusters in an energy-efficient way. With a training data set they create a model which they use to predict the future performance of the jobs and the energy consumption in the resulting allocations.

Second, [40,71] give a good overview of many different (semantic) knowledge management methods and their applications. Presented methods include rules, default logic, case based reasoning, situation calculus, truth maintenance systems, logic programming, answer set planing (e.g., SMODELS [166], DLV [134]), and agent-based systems. Those considered for usage in the area of governing Cloud computing infrastructures have been presented in Section 4.1.

Eichner et al. [78] describe a KM approach in the broader field of software development in the framework of the BREIN project [86], which aims at developing an intelligent grid infrastructure. Saripalli et al. [189] follow the path of predicting loads for SaaS platforms by cubic spline interpolation. Bhoj et al. [45] present early work on the monitoring and management of SLAs (not enactment, though) in distributed systems. They present tools and languages for formalizing SLAs. As SLA parameters they consider typical parameters like availability, response time, throughput or utilization. Their work can be seen as preceding service-oriented architectures. Dan et al. [70] use the WSLA standard [108] to describe a high level view and an architecture for SaaS.

Third, Lee et al. [131] present workflow adaptation as an autonomic computing problem. They separate the four phases of the MAPE cycle neatly and devise concrete actions for the monitoring, analysis, and planning phases. For monitoring they use the progress of a service, its data consumption rate, or the load on an execution node as performance indicators. These indicators are then used in the analysis phase to determine potential problems as load imbalances, or bottlenecks, as well as opportunities as free capacities, or underutilized execution nodes. In the planning phase, actions are planned to be executed that mitigate the found problems or opportunities in the previous phase. These actions include making a workflow complete more quickly by increasing service parallelism or by rescheduling a service to a different execution node, or to resort to faster data sources. However, problems in the analysis phase can trigger not only one, but several different actions in the planning phase, and it is not clarified in what manner to make the necessary decision-making for which action should be triggered. Similar to this, [130] include a utility measurement into the autonomic cycle of the workflow adaptation to trigger concrete actions. The utility is based on response time, or profit, where also execution costs are taken into account. In the planning phase, the action is triggered that maximizes the utility function. These calculations are based on a model that can estimate the response and queue times that the examined actions would cause. More generally, [110] give an overview of policies for autonomic computing. These policies are based on actions, goals, and utility functions.

Fourth, CloudSim [63] is another toolkit for modeling a Cloud infrastructure to evaluate resource management strategies. The main difference to our simulation engine is that CloudSim uses the concept of a *Cloudlet*, which is based on the former *Gridlet* used in GridSim [60]. A Cloudlet assumes a certain start and finishing time of a job, which is very typical in a Grid environment. However, applications in Cloud computing do not necessarily have a limited execution time, but run steadily as web or database servers. This is reflected by our simulation engine.

8.4 Self-Adaptive Algorithms for Cloud Computing Infrastructures

In this section we present state of the art particularly relevant to our work in Section 4.5.

Dutreilh et al. [76] investigate horizontal scaling, e.g., adding or removing VMs running an application server by using a load balancer, using a threshold-based and a reinforcement learning technique. However, the authors do not consider adapting the thresholds themselves via learning. Moreover, the authors determine problems with static thresholds as well as with determining

good tuning for the reinforcement algorithms. The authors also state the importance of understanding the workload variation, but do not present a method how to deal with it. Kalyvianaki et al. [104] use Kalman filters for CPU resource provisioning for virtualized servers. They self-adapt their approach using variances and covariances. Bu et al. [56] use a reinforcement learning approach combined with the simplex algorithm to auto-configure virtual machines and applications in a coordinated way. Padala et al. [171] develop self-tuning controllers for multi-tier applications using control theory. Song et al. [193] use self-adaptation in the field of Cloud federations. Their algorithm selects tasks and allocates them by finding a trade-off between SLA adherence and resource utilization. This trade-off is represented by a parameter, which is optimized using a similar principle as the bisection method. For the optimization, the benefit of a specific threshold is estimated by simulation. This estimation is executed several times until an adequate value is found.

[178] apply genetic algorithms for decision making and self-reconfiguration, but on the network topology of remote data mirrors. Heinis et al. [97] experiment with self-configuring thresholds, but tied to a workflow execution engine. Ghanbari et al. [89] also dynamically classify workload, but they use clustering techniques based on parameters such as response time or throughput. Their model is intended rather for web servers than for general applications. Almost the same authors in [88] investigate and compare control-theoretic and rule-based approaches to achieve elasticity. For them elasticity means to add or remove resources such as application server instances on the PaaS layer.

Summarizing, there are quite few works on self-adaptive algorithms for managing Cloud computing infrastructures, and none of them self-adapts an approach for VM reconfiguration, nor deals with the volatility of Cloud workloads.

8.5 Energy-Efficient Cloud Computing Infrastructures

In this section we describe work explicitly dealing with energy efficiency in Cloud computing and compare it with our work in Chapter 5. There has been considerable work on energy efficiency in ICT systems. Their common goal is to attain certain performance criteria for reducing energy consumption. Many of these works, though, focus on different escalation levels (cf. Section 2.3) alone, and do not combine them as in our approach.

E.g., [210, 160, 140] only focus on VM migration and [157] on turning on and off physical machines. Our approach achieves a more holistic approach taking all these mentioned level plus VM reconfiguration into account. Meng et al. [160] try to increase efficient resource usage by provisioning multiple specific VMs together on a physical machine. [138, 98] reduce power consumption by PM consolidation using several heuristics. Shi et al. [191] aim at attaining SLAs with a given energy budget. They maximize profit by efficient virtual machine placement. Wang et al. [204] inspect other aspects virtualization has on workloads performance besides VM migration, namely the number of virtual CPUs per VM and their memory share of the PM. Goiri et al. [90] present energy-aware scheduling of VMs on PMs. However, for migration they assume just some arbitrary value, and they only consider jobs with a deadline instead of permanently running applications. Voorsluys et al. [202] tackle the cost of live migration of virtual machines regarding the response time of the services inside the VMs in order to match the response time

with the SLA requirements of the services. Liu et al. [139] also have studied live migration of virtual machines in order to model the performance and energy consumption of the migration. They show that migration is an I/O intensive application, and that it consumes energy on both ends. The architectural framework proposed in [41] to achieve green clouds also achieves VM reconfiguration, allocation and reallocation. They use a CPU power model to monitor the energy consumption of the cloud. The algorithm they propose to achieve dynamic consolidation of the VMs significantly reduces the global power consumption of their infrastructure. Their work, however, differs from our approach in several points, the main points being the use of a different VM migration model, the use of a reactive VM reconfiguration instead of reactive and proactive one, and not taking into account time taken to power on and off hosts. Our research provides a more wholesome approach than related work and integrates most of the different possible escalation levels seen in the literature.

Some authors as Kalyvianaki [104] focus on optimizing a specific resource type such as CPU usage, or only deal with homogeneous resources [115]. While most authors assume a theoretical energy model behind their approaches, Yu [212] targets the more basic question of how to effectively measure energy consumption in Cloud computing environments in a scalable way. Additionally, Klingert et al. [117] take energy efficiency into account already when defining SLAs. E.g., they specify that a job must not run during nighttime, since it could not be powered by solar energy then.

Some works consider energy savings for very specific settings as [120] for parallel applications, [119] for multicore architectures, or [121] shared memory architectures. On the contrary, in [141] the authors take a more holistic approach by also considering energy savings stemming from more efficient cooling systems.

As to our formulation of the IaaS management problem as a binary integer programming problem in Section 5.2, there are some works which also formulate similar problems as integer programming problems. [177] uses it to formalize an allocation problem of scheduling periodic tasks to a fixed number of processors. [136] uses it for VM migration (without PM power management) in Cloud environments. [64] uses stochastic integer programming for VM placement for a fixed number of resources. To the best of our knowledge there is no formulation of the IaaS management problem accounting for arbitrary resource types, VM migrations and their costs, as well as PM power management and its cost in a heterogeneous computing environment into a binary integer programming problem.

8.6 Cloud Federations

In this section we describe work related to our work presented in Chapter 7.

Bernstein et al. [43] define two use case scenarios that exemplify the problems faced by users of multi-cloud systems. They define the case of VM mobility, where they identify networking, specific cloud VM management interfaces and the lack of mobility interfaces as the three major obstacles. They also discuss a storage interoperability and federation scenario, in which storage provider replication policies are subject to change when a cloud provider initiates subcontracting. However, they offer interoperability solutions only for low-level functionality of clouds that are not focused on recent user demands, but on solutions for IaaS system operators.

Buyya et al. in [61] suggest a services provisioning environment called InterCloud, which is Cloud federation oriented, just-in-time, opportunistic, and scalable. They envision utility-oriented federated IaaS systems that are able to predict application service behavior for intelligently down- and up-scaling infrastructures. They also present a market-oriented approach to offer InterClouds including cloud exchanges and brokers that bring together producers and consumers. Producers are offering domain specific enterprise Clouds that are connected and managed within the federation with their Cloud Coordinator component. Finally, they have implemented a CloudSim-based simulation that evaluates the performance of the federations created using InterCloud technologies. Unfortunately, users face most federation-related issues before the execution of their services, therefore the concept of InterClouds cannot be applied in user scenarios our work is targeting.

Frincu et al. [85] study placing applications on nodes in a multi-cloud setting. They take all nodes of federated Clouds into account for scheduling decisions, and they may place different parts of the same application on different Clouds. In their approach the authors violate the integrity of one Cloud, since their scheduler does not only decide to put an application into a specific Cloud, but it also selects a VM for it to run on. Thus, this scenario is rather applicable for only one Cloud or Clouds having the same owners or profit maximizing strategy.

Besides, none of the presented approaches in Section 8.3 investigated knowledge management or the MAPE-K autonomic loop in Cloud federations.

8.7 Holistic Cloud Management Projects

Finally, we will relate the FoSII project to other Cloud management projects. As compared to, e.g., SLA@SOI [26], the FoSII project in general is more specific on Cloud Computing aspects like deployment, monitoring of resources and their translation into high level SLAs instead of just working on high-level SLAs in general service-oriented architectures.

We will describe other related Cloud management projects in the following. The Reservoir model [185] is a framework for Cloud computing with the conceptual addition of SLA management. It states the need of dynamically adjusting resources (in addition to federating resources from peer providers) in order to meet SLAs, but does not specify a way to do that. The ConPaaS project [176] aims at providing scalable open-source software for providing PaaS. The project provides a testbed to create VMs, on which web servers run that host Java servlets [162] or PHP documents [22], or databases like Scalarix [25] or MapReduce [73]. The focus of the BREIN project [86] has been laid on grids, and the GridEcon project [30] focuses on providing a marketplace for grid or cloud resources. The Consequence project [169] focuses on confidentiality and privacy aspects of data exchange in distributed systems. They use decision-making by risk assessment [124]. The SORMA project [167] developed a self-organizing resource management system for efficient market allocation. StreamCloud [92] present a Cloud computing platform which is specialized on processing large data streams in a scalable and resource-efficient way. They use horizontal scaling and a parallelization technique that splits queries into subqueries that can be allocated to different computing nodes.

Furthermore, there are other works that take a more general and holistic view on Cloud computing. Sedaghat et al. [190] aim at unifying Cloud management. They present an archi-

tecture involving important management components that could be developed independently of each other. These components include an admission controller, a VM placement engine, a data placement engine, an elasticity engine, a fault tolerance controller, and an SLA management engine. With this architecture the authors want to achieve a business level objective, which is expressed as a utility function to maximize profit. Rimal et al. [181] present an early overview on different Cloud deployment platforms and the utilized technologies. They evaluate the computing architecture, load balancing and fault tolerance strategies, storage and security systems of platforms like Flexiscale [16], Mosso, which has been rebranded to Rackspace Cloud [23], Google App Engine [7], RightScale [24], or Azure [9]. Youseff et al. [211] work towards an ontology of Cloud computing. They present five layers of Cloud, namely hardware, the software kernel, the cloud software infrastructure, the cloud software environment and cloud applications, and discuss their relations to each other. Lenk et al. [133] use a similar ontology and present a corresponding stack architecture.

On the contrary, the FoSII project, and more specifically the work in this thesis are concerned about governing Cloud computing infrastructures in terms of VM configuration, VM migration, PM power management, and Cloud federations under the aspects of adaptive SLA generation and autonomic SLA enactment, as well as resource and energy efficiency.

CHAPTER 9

Conclusion

In this thesis we have devised strategies for two important problem fields of Cloud computing related to Service Level Agreements (SLAs): SLA management and energy-efficient and resource-efficient SLA enactment. In the following, we will summarize the achievements, limitations, and future work of both areas separately.

We will start with SLA management. We have investigated cost, utility, and net utility of the adaptive SLA mapping approach, in which market participants may define SLA mappings for translating their private SLA templates to public SLA templates. Contrary to all other available SLA matching approaches, the adaptive SLA mapping approach facilitates the continuous adaptation of public SLA templates based on market trends. However, the adaptation of SLA mappings comes with a cost for users in the form of effort for generating new SLA mappings to the adapted public SLA template. To calculate the cost and benefits of the SLA mapping approach, we utilized the SLA management framework VieSLAF and simulated different market situations. Our findings show that the cost for SLA mappings can be reduced by introducing heuristics into the adaptation methods for generating adapted public SLA templates. The methods show cost reduction and an increase in average overall net utility. The best-performing adaptation method is the maximum-percentage-change method.

In recent work, Breskovic et al. [54, 55] have already carried out work that is based on this one. They inspected whether intelligently determining different groups of users coming from different domains can increase the overall net utility. Furthermore, they did not only take the SLA parameter names into account, but also the different metrics they are measured with (e.g., storage measured in MB or GB, or response time measured completely differently for different types of applications (also cf. [147])) and the values or intervals of the desired Service Level Objectives.

For future work, we want to investigate other metrics (besides the quantity based mapping count) for the adaptation methods. This could be the measured market liquidity after a new SLA template gets introduced into the market. Additionally, it would be interesting to identify the optimal number of different SLA templates to maximize overall net utility.

A limitation of this work is that it only considers SLA parameters, and not its values as

achieved in later work [55]. Furthermore, the simulation is based on a random group of users drawn from a specific distribution for every iteration. It would have been interesting to effectuate real-world case studies that test how many users would give up their proprietary SLA templates to follow pre-selected ones. This could then affect the outcome of the various adaptation techniques. Finally, the assumption of low market liquidity due to a plethora of different definitions and namings of SLA parameters could be challenged, once a standardization of these parameters has taken place.

As to SLA enactment, the first goal is to enact SLAs in a resource-efficient way. Autonomically governing Cloud Computing infrastructures is the investigated method, whose goal is to reduce SLA violations, increase resource utilization and achieve both by a low number of reconfiguration actions.

In this thesis we have hierarchically structured all possible reallocation actions, and conducted a study over several knowledge management techniques. We have then designed, implemented, and evaluated the two most promising knowledge management techniques, Case Based Reasoning and a rule-based approach to achieve the aforementioned goals for one reallocation level, i.e., VM reconfiguration. After a comparison, we determined the rule-based approach to outperform CBR with respect to violations and utilization, but also to time performance. Consequently, we continued investigation of the rule-based approach with different synthetic workload volatility classes. Furthermore, we applied the rule-based approach to a real-world use case evaluating a scientific workflow from the area of bioinformatics. We showed by simulation that the rule-based approach can effectively guarantee the execution of a workload with unpredictably large resource consumptions.

However, the presented methods still involve some user-interaction for parameter tuning. Thus, we have devised several methodologies for autonomically adapting parameters of a Cloud resource management framework on the level of VM reconfiguration. We presented two groups of strategies: the first one is based on a cost function that reflects the goal of the approach. The second strategy is based on classifying the workload into workload volatility classes. This second strategy acts according to this classification by either applying the substrategy of preconfigured parameters or the substrategy of applying the most appropriate strategy from the first group. In most cases we have seen that strategies from the second group achieve better results for both substrategies, and outperform the strategies not taking workload volatility into account. Thus, we can deduce that workload volatility is an important aspect for governing Cloud computing infrastructures. Corresponding research is still at its beginning.

For future work we want to prove the benefit regarding the energy consumption for the self-adapting approach. We will be able to not only capture the improvement in costs of the self-adaption, but also the reduction in energy consumption as compared to a non-self-adapting approach. Furthermore, we plan to investigate if we can generalize the findings for autonomically adapting approaches for other levels of governing Cloud computing infrastructures, e.g., VM migration or PM power management.

Furthermore, we have analyzed a possible extension of our approach to the last escalation level, i.e., Cloud federations. Using the presented FCM architecture as the basis of our further investigations, we analyzed different approaches to integrate the knowledge management system within this architecture, and found a hybrid approach that incorporates fine-grained lo-

cal adaptation operations with options for high-level override. Then this research pinpointed the adaptation actions and their possible effects on cloud federations. Finally, we established metrics that could indicate possible SLA violations in federations, and defined rules that could trigger adaptation actions in the case of predicted violations. Regarding future work, we plan to investigate more the green aspects in the autonomous behavior of cloud federations. We also aim at defining new rules for advanced action triggering and evaluating the applicability of case based reasoning. Finally, we also plan to investigate the effects of the autonomous behavior on the overall performance of the cloud federation on an experimental system.

Going now ultimately from resource efficiency to energy efficiency, we have presented a management framework for governing Cloud Computing infrastructures to achieve two goals: reduce energy consumption while keeping pre-defined Service Level Agreements. We have a devised a multi-level action approach that breaks down the NP-hard resource allocation problem for Clouds. We have specialized on several views of the Cloud computing infrastructure, i.e., VM reconfiguration, VM migration, and PM power management, in order to reduce the problem's complexity. In each of these views we have defined a subproblem and solved it using a wide variety of heuristics ranging from rules over random methods, i.e., Monte Carlo, to vector packing algorithms. We have evaluated the sequential execution of these views. We showed for the first time that the VM reconfiguration algorithm alone, which already succeeded to minimize SLA violations and decrease resource wastage, also effectively saves up to 61.6% of energy. Considering scalable algorithms, these energy savings can still be increased by up to 37% in the best case and 11% in the worst case while keeping SLA violations at 0% for the workload of a bioinformatics scientific workflow, below 4% for synthetic workloads with low volatility for all VM migration algorithms, and below 8% for synthetic workloads with higher volatility for the smarter VM migration algorithms.

For future work we plan to focus more on a possible heterogeneity of the systems, re-fining the migration model, and integrating the framework into a real-world Cloud computing environment. Also other ongoing projects as HALEY that work on realizing a holistic energy-efficient approach for the management of hybrid clouds [20] will be of interest for this work. Possibly in the framework of the that project, a next step would be to move from simulation to a real Cloud testbed, where real energy measurements could be made. Also the timeliness of the iteration-based simulation would have to be investigated more deeply. Basing this work on simulation only is probably the greatest limitation of this work. However, the implementation of the SLA enactor on a real-world system requires a lot of computing nodes that are under full control of the researchers. Nevertheless, a smaller prototype Cloud that shows the principal validation would already be beneficial. Furthermore, this thesis does not cover escalation level 2, i.e., application migration as it assumes that an application resides on exactly one VM, and one VM hosts exactly one application. This assumption could be challenged for efficiency reasons. Allowing more applications to reside on the same VM would reduce the overhead caused by the VMs and their operating systems. However, security issues still have to be addressed. An application is better isolated, of course, when it resides on a VM alone.

Another related field for future work is the autonomic generation of an IaaS SLA out of SaaS or PaaS SLAs. Theoretically, SaaS or PaaS applications can be perfectly set up on top of IaaS platforms. The crucial point is to extract an SLA for the IaaS parameters like bandwidth, storage,

CPU power and memory that fit to SaaS/PaaS parameters like response time. It is obvious that response time directly relates to the mentioned IaaS parameters and user interaction. It is not that obvious, however, how this translation should take place. E.g., does the SLO "response time < 2 s" translate into "memory > 512 MB" and "CPU power > 8000 MIPS" or rather "memory > 4096 MB" and "CPU power > 1000 MIPS"? Once the autonomic governance of IaaS infrastructures is up and running, the autonomic translation of these SLAs will probably leverage the usage and usability of IaaS even more.

Bibliography

[1] POV-Ray. http://www.povray.org/, 2012.

[2] Gartner estimates ICT industry accounts for 2 percent of global CO2 emissions. http://www.gartner.com/it/page.jsp?id=503867, 2007.

[3] Amazon elastic compute cloud (Amazon EC2). http://aws.amazon.com/ec2/, 2010.

[4] Brein - business objective driven reliable and intelligent grids for real business. http://www.eu-brein.com/, 2010.

[5] Emc atmos online. https://mgmt.atmosonline.com/, 2010.

[6] FreeCBR. http://freecbr.sourceforge.net/, 2010.

[7] Google app engine. http://code.google.com/appengine/, 2010.

[8] Jess. http://www.jess.org, 2010.

[9] Microsoft azure. http://www.microsoft.com/windowsazure/, 2010.

[10] Salesforce.com. http://www.salesforce.com, 2010.

[11] Sun grid. http://www.sun.com/service/sungrid/index.jsp, 2010.

[12] Tsunamic tech. inc. http://www.clusterondemand.com/, 2010.

[13] Documentation for matlab function bintprog. http://www.mathworks.de/help/toolbox/optim/ug/bintprog.html, March 2012.

[14] Documentation for matlab function sparse. http://www.mathworks.de/help/techdoc/ref/sparse.html, March 2012.

[15] Drools. http://www.drools.org, 2012.

[16] Flexiscale. http://www.flexiscale.com/, 2012.

[17] (FOSII) - Foundations of Self-governing ICT Infrastructures. http://www.infosys.tuwien.ac.at/linksites/FOSII, March 2012.

[18] Google docs. https://docs.google.com/, 2012.

[19] Google mail. http://mail.google.com/, 2012.

[20] Hollistic energy-efficient approach for the management of hybrid clouds (HALEY). http://www.infosys.tuwien.ac.at/linksites/haley/, March 2012.

[21] Nimbus. http://www.nimbusproject.org/, 2012.

[22] PHP. http://www.php.net/, 2012.

[23] Rackspace cloud. http://www.rackspace.com/cloud/, March 2012.

[24] Rightscale. http://www.rightscale.com/, 2012.

[25] Scalarix. http://www.onscale.de/scalarix.html, 2012.

[26] SLA@SOI. http://sla-at-soi.eu/, March 2012.

[27] Agnar Aamodt and Enric Plaza. Case-based reasoning: Foundational issues, methodological variations, and system approaches. *AI Communications*, 7:39–59, 1994.

[28] Sheikh Iqbal Ahamed et al., editors. *Proceedings of the 33rd Annual IEEE International Computer Software and Applications Conference, COMPSAC 2009, Seattle, Washington, USA, 20-24 July 2009*. IEEE Computer Society, 2009.

[29] Rainer Alt and Stefan Klein. Twenty years of electronic markets research—looking backwards towards the future. *Electronic Markets*, 21(1):41–51, 2011.

[30] Jörn Altmann, Costas Courcoubetis, John Darlington, and Jeremy Cohen. Gridecon - the economic-enhanced next-generation internet. In Jörn Altmann and Daniel Veit, editors, *GECON*, volume 4685 of *Lecture Notes in Computer Science*, pages 188–193. Springer, 2007.

[31] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. SETI@home: an experiment in public-resource computing. *Commun. ACM*, 45:56–61, November 2002.

[32] A Andrieux, K Czajkowski, A Dan, K Keahey, H Ludwig, J Pruyne, J Rofrano, S Tuecke, and M Xu. Web services agreement specification (WS-agreement). *Global Grid Forum*, 31(GFD.107):1–47, 2007.

[33] Grigoris Antoniou. A tutorial on default logics. *ACM Comput. Surv.*, 31(4):337–359, 1999.

[34] D. Ardagna, G. Giunta, N. Ingraa, R. Mirandola, and B. Pernici. Qos-driven web services selection in autonomic grid environments. In *International Conference on Grid Computing, High Performance and Distributed Applications (GADA)*, Montpellier, France, November 2006.

[35] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, February 2009.

[36] Raphael M. Bahati and Michael A. Bauer. Adapting to run-time changes in policies driving autonomic management. In *ICAS '08: Proceedings of the 4th Int. Conf. on Autonomic and Autonomous Systems*, Washington, DC, USA, 2008. IEEE Computer Society.

[37] Mark Baker and Garry Smith. Gridrm: A resource monitoring architecture for the grid. In *Proceedings of the Third International Workshop on Grid Computing*, GRID '02, pages 268–273, 2002.

[38] Ilia Baldine, Yufeng Xin, Daniel Evans, Chris Heerman, Jeff Chase, Varun Marupadi, and Aydan Yumerefendi. The missing link: Putting the network in networked cloud computing. In *ICVCI09: International Conference on the Virtual Computing Initiative*, 2009.

[39] 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 Oper. Syst. Rev.*, 37(5):164–177, October 2003.

[40] Christoph Beierle and Gabriele Kern-Isberner. *Methoden wissensbasierter Systeme*. Vieweg, 2006.

[41] 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*, 28(5):755 – 768, 2012.

[42] Fran Berman, Anthony Hey, and Geoffrey Fox. *Grid Computing: Making The Global Infrastructure a Reality*. John Wiley & Sons, April 2003.

[43] D. Bernstein, E. Ludvigson, K. Sankar, S. Diamond, and M. Morrow. Blueprint for the intercloud - protocols and formats for cloud computing interoperability. In *Internet and Web Applications and Services, 2009. ICIW '09. Fourth International Conference on*, pages 328 –336, may 2009.

[44] Josep Ll. Berral, Íñigo Goiri, Ramón Nou, Ferran Julià, Jordi Guitart, Ricard Gavaldà, and Jordi Torres. Towards energy-aware scheduling in data centers using machine learning. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, e-Energy '10, pages 215–224, New York, NY, USA, 2010. ACM.

[45] P Bhoj, S Singhal, and S Chutani. SLA management in federated environments. *Computer Networks*, 35(1):5 – 24, 2001. Selected Topics in Network and Systems Management.

[46] Martin Bichler, Thomas Setzer, and Benjamin Speitkamp. Capacity Planning for Virtualized Servers. *Presented at Workshop on Information Technologies and Systems (WITS), Milwaukee, Wisconsin, USA, 2006*, 2006.

[47] Damien Borgetto, Henri Casanova, Georges Da Costa, and Jean-Marc Pierson. Energy-aware service allocation. *Future Generation Computer Systems*, 28(5):769 – 779, 2012.

[48] Damien Borgetto, Georges Da Costa, Jean-Marc Pierson, and Amal Sayah. Energy-Aware Resource Allocation. In *Proc. of the Energy Efficient Grids, Clouds and Clusters Workshop (E2GC2)*, page (electronic medium). IEEE, October 2009.

[49] Damien Borgetto, Michael Maurer, Georges Da Costa, Jean-Marc Pierson, and Ivona Brandic. Energy-efficient and SLA-aware managament of IaaS clouds. In *Third international conference on future energy systems (e-Energy 2012)*, Madrid, Spain, May 2012.

[50] I. Brandic, S. Benkner, G. Engelbrecht, and R. Schmidt. Qos support for time-critical grid workflow applications. In *1st IEEE International Conference on e-Science and Grid Computing*, Melbourne, Australia, December 2005.

[51] I. Brandic, D. Music, P. Leitner, and S. Dustdar. VieSLAF framework: Enabling adaptive and versatile SLA-management. In *GECON2009. In conjunction with Euro-Par 2009*, Delft, The Netherlands, August 2009.

[52] Ivona Brandic. Towards self-manageable cloud services. In Ahamed et al. [28], pages 128–133.

[53] Ivona Brandic, Tobias Anstett, David Schumm, Frank Leymann, Schahram Dustdar, and Ralf Konrad. Compliant cloud computing (C3): Architecture and language support for user-driven compliance management in clouds. In *The 3rd International Conference on Cloud Computing (IEEE Cloud 2010)*, Miami, FL, USA, July 2010.

[54] Ivan Breskovic, Michael Maurer, Vincent C. Emeakaroha, Ivona Brandic, and Jörn Altmann. Towards autonomic market management in cloud computing infrastructures. In *International Conference on Cloud Computing and Services Science - CLOSER 2011*, Noordwijkerhout, the Netherlands, May 2011.

[55] Ivan Breskovic, Michael Maurer, Vincent C. Emeakaroha, Ivona Brandic, and Schahram Dustdar. Cost-efficient utilization of public sla templates in autonomic cloud markets. In *4th IEEE International Conference on Utility and Cloud Computing (UCC 2011)*, Melbourne, Australia, December 2011.

[56] Xiangping Bu, Jia Rao, and Cheng-Zhong Xu. A model-free learning approach for coordinated configuration of virtual machines and appliances. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*, pages 12 –21, July 2011.

[57] R. Buyya, D. Abramson, and J. Giddy. A case for economy grid architecture for service oriented grid computing. In *Parallel and Distributed Processing Symposium*, 2001.

[58] R. Buyya and K. Bubendorfer. *Market Oriented Grid and Utility Computing*. John Wiley & Sons, Inc, New Jersey, USA, 2008.

[59] Rajkumar Buyya. *High Performance Cluster Computing: Architectures and Systems*, volume 1. Prentice Hall, Upper Saddle River, NJ, 1999.

[60] Rajkumar Buyya and Manzur Murshed. Gridsim: a toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220, 2002.

[61] Rajkumar Buyya, Rajiv Ranjan, and Rodrigo Calheiros. Intercloud: Utility-oriented federation of cloud computing environments for scaling of application services. In Ching-Hsien Hsu, Laurence Yang, Jong Park, and Sang-Soo Yeo, editors, *Algorithms and Architectures for Parallel Processing*, volume 6081 of *Lecture Notes in Computer Science*, pages 13–31. Springer Berlin / Heidelberg, 2010.

[62] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599 – 616, 2009.

[63] Rodrigo N. Calheiros, Rajiv Ranjan, Anton Beloglazov, César A. F. De Rose, and Rajkumar Buyya. 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.

[64] S. Chaisiri, Bu-Sung Lee, and D. Niyato. Optimal virtual machine placement across multiple cloud providers. In *Services Computing Conference, 2009. APSCC 2009. IEEE Asia-Pacific*, pages 103 –110, December 2009.

[65] J. Chen and B. Lu. An universal flexible utility function in grid economy. In *IEEE Pacific-Asia Workshop on Computational Intelligence and Industrial Application*, 2008.

[66] Yuan Chen, Subu Iyer, Xue Liu, Dejan Milojicic, and Akhil Sahai. Translating service level objectives to lower level policies for multi-tier services. *Cluster Computing*, 2008.

[67] Wai-Khuen Cheng, Boon-Yaik Ooi, and Huah-Yong Chan. Resource federation in grid using automated intelligent agent negotiation. *Future Generation Computer Systems*, 26(8):1116–1126, October 2010.

[68] Hyung Won Choi, Hukeun Kwak, Andrew Sohn, and Kyusik Chung. Autonomous learning for efficient resource utilization of dynamic VM migration. In *Proceedings of the 22nd annual international conference on Supercomputing*, ICS '08, pages 185–194, New York, NY, USA, 2008. ACM.

[69] Michael T. Cox, Héctor Muñoz-Avila, and Ralph Bergmann. Case-based planning. *The Knowledge Engineering Review*, 20(03):283–287, 2005.

[70] A. Dan, D. Davis, R. Kearney, A. Keller, R. King, D. Kuebler, H. Ludwig, M. Polan, M. Spreitzer, and A. Youssef. Web services on demand: WSLA-driven automated management. *IBM Systems Journal*, 43(1):136 –158, 2004.

[71] John Davies, editor. *Semantic knowledge management*. Springer, 2009.

[72] Marcos de Assunção, Alexandre di Costanzo, and Rajkumar Buyya. A cost-benefit analysis of using cloud computing to extend the capacity of clusters. *Cluster Computing*, 13:335–347, 2010. 10.1007/s10586-010-0131-x.

[73] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51:107–113, January 2008.

[74] Salvatore Distefano, Maria Fazio, and Antonio Puliafito. The cloud@home resource management system. *Utility and Cloud Computing, IEEE Internatonal Conference on*, 0:122–129, 2011.

[75] G. Dobson and A. Sanchez-Macian. Towards unified QoS/SLA ontologies. In *IEEE Services Computing Workshops (SCW)*, pages 18–22, Chicago, Illinois, USA, 2006.

[76] X. Dutreilh, N. Rivierre, A. Moreau, J. Malenfant, and I. Truck. From data center resource allocation to control theory and back. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 410 –417, July 2010.

[77] Dmytro Dyachuk and Ralph Deters. A solution to resource underutilization for web services hosted in the cloud. In Robert Meersman, Tharam Dillon, and Pilar Herrero, editors, *On the Move to Meaningful Internet Systems: OTM 2009*, volume 5870 of *Lecture Notes in Computer Science*, pages 567–584. Springer Berlin / Heidelberg, 2009. 10.1007/978-3-642-05148-7_42.

[78] Hannes Eichner, András Micsik, Máté Pataki, and Robert Woitsch. A use case of service-based knowledge management for software development. In *IFIP International Conference on Research and Practical Issues of Enterprise Information Systems (Confenis)*, October 2009.

[79] V. C. Emeakaroha, I. Brandic, M. Maurer, and S. Dustdar. Low level metrics to high level SLAs - LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments. In *The 2010 High Performance Computing and Simulation Conference in conjunction with IWCMC 2010*, Caen, France, 2010.

[80] Vincent C. Emeakaroha, Pawel Labaj, Michael Maurer, Ivona Brandic, and David P. Kreil. Optimizing bioinformatics workflows for data analysis using cloud management techniques. In *The 6th Workshop on Workflows in Support of Large-Scale Science (WORKS11)*, 2011.

[81] Vincent C. Emeakaroha, Marco A. S. Netto, Rodrigo N. Calheiros, Ivona Brandic, and César A. F. De Rose. Desvi: An architecture for detecting SLA violations in cloud computing infrastructures. In *CloudComp 2010*, Barcelona, Spain, October 2010.

[82] Xiaobo Fan, Wolf dietrich Weber, and Luiz André Barroso. Power provisioning for a warehouse-sized computer. In *Proceedings of ISCA*, 2007.

[83] Gerhard Fettweis and Ernesto Zimmermann. ICT energy consumption – trends and challenges. In *The 11th International Symposium on Wireless Personal Multimedia Communications (WPMC 2008)*, 2008.

[84] Rolf Findeisen, Frank Allgöwer, and Lorenz T. Biegler. *Assessment and Future Directions of Nonlinear Model Predictive Control*. Springer-Verlag Berlin Heidelberg, 2007.

[85] Marc E. Frincu and Ciprian Craciun. Multi-objective meta-heuristics for scheduling applications with high availability requirements and cost constraints in multi-cloud environments. In *Utility and Cloud Computing, IEEE Internatonal Conference on*, pages 267–274, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

[86] Henar Muñoz Frutos and Ioannis Kotsiopoulos. Brein: Business objective driven reliable and intelligent grids for real business. *International Journal of Interoperability in Business Information Systems*, 3(1), 2009.

[87] Saurabh Kumar Garg, Rajkumar Buyya, and Howard Jay Siegel. Time and cost tradeoff management for scheduling parallel applications on utility grids. *Future Generation Computer Systems*, 26(8):1344–1355, October 2010.

[88] H. Ghanbari, B. Simmons, M. Litoiu, and G. Iszlai. Exploring alternative approaches to implement an elasticity policy. In *Cloud Computing (CLOUD), 2011 IEEE International Conference on*, pages 716 –723, July 2011.

[89] Hamoun Ghanbari, Cornel Barna, Marin Litoiu, Murray Woodside, Tao Zheng, Johnny Wong, and Gabriel Iszlai. Tracking adaptive performance models using dynamic clustering of user classes. *SIGSOFT Softw. Eng. Notes*, 36(5):179–188, September 2011.

[90] I. Goiri, F. Julià and, R. Nou, J.L. Berral, J. Guitart, and J. Torres. Energy-aware scheduling in virtualized datacenters. In *Cluster Computing (CLUSTER), 2010 IEEE International Conference on*, pages 58 –67, September 2010.

[91] Les Green. Service level agreements: an ontological approach. In *8th international conference on Electronic commerce: The new e-commerce: innovations for conquering current barriers, obstacles and limitations to conducting successful business on the internet, ICEC '06*, New York, NY, USA, 2006.

[92] V. Gulisano, R. Jimenez-Peris, M. Patino-Martinez, and P. Valduriez. Streamcloud: A large scale data streaming system. In *Distributed Computing Systems (ICDCS), 2010 IEEE 30th International Conference on*, pages 126 –137, June 2010.

[93] Aminul Haque, Saadat M. Alhashmi, and Rajendran Parthiban. A survey of economic models in grid computing. *Future Generation Computer Systems*, 2011.

[94] J. A. Hartigan and M. A. Wong. Algorithm as 136: A k-means clustering algorithm. *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, 28(1):100–108, 1979.

[95] Peer Hasselmeyer, Bastian Koller, Lutz Schubert, and Philipp Wieder. Towards SLA-supported resource management. In *High Performance Computing and Communications*, pages 743–752. Springer, Berlin / Heidelberg, 2006.

[96] Mark Hefke. A framework for the successful introduction of KM using CBR and semantic web technologies. *Journal of Universal Computer Science*, 10(6), 2004.

[97] Thomas Heinis and Cesare Pautasso. Automatic configuration of an autonomic controller: An experimental study with zero-configuration policies. In Ahamed et al. [28], pages 67–76.

[98] Yufan Ho, Pangfeng Liu, and Jan-Jan Wu. Server consolidation algorithms with bounded migration cost and performance guarantees in cloud computing. In *Utility and Cloud Computing, IEEE Internatonal Conference on*, pages 154–161, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

[99] Marko Hoyer, Kiril Schröder, and Wolfgang Nebel. Statistical static capacity management in virtualized data centers supporting fine grained QoS specification. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, e-Energy '10, pages 51–60, New York, NY, USA, 2010. ACM.

[100] Markus C. Huebscher and Julie A. McCann. A survey of autonomic computing—degrees, models, and applications. *ACM Comput. Surv.*, 40(3):1–28, 2008.

[101] Sadeka Islam, Jacky Keung, Kevin Lee, and Anna Liu. Empirical prediction models for adaptive resource provisioning in the cloud. *Future Generation Computer Systems*, 28(1):155 – 162, 2012.

[102] B. Jacob, R. Lanyon-Hogg, D. K. Nadgir, and A. F. Yassin. A practical guide to the IBM Autonomic Computing toolkit. *IBM Redbooks*, 2004.

[103] Peter Johnson and Tony Marker. Data center energy efficiency product profile. Technical report, Equipment Energy Efficiency Program (E3) Energy Rating (A joint initiative of Australian, State and Territory and New Zealand Governments), 2009.

[104] Evangelia Kalyvianaki, Themistoklis Charalambous, and Steven Hand. Self-adaptive and self-configured CPU resource provisioning for virtualized servers using Kalman filters. In *Proceedings of the 6th international conference on Autonomic computing*, ICAC '09, pages 117–126, New York, NY, USA, 2009. ACM.

[105] Richard M. Karp. Reducibility among combinatorial problems. In Raymond E. Miller and James W. Thatcher, editors, *Complexity of Computer Computations: Proc. of a Symp. on the Complexity of Computer Computations*, pages 85–103. Plenum Press, 1972.

[106] Gabor Kecskemeti, Michael Maurer, Ivona Brandic, Attila Kertesz, Zsolt Nemeth, and Schahram Dustdar. Facilitating self-adaptable inter-cloud management. In *20th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP 2012)*, Munich, Germany, February 2012.

[107] Gabor Kecskemeti, Gabor Terstyanszky, Peter Kacsuk, and Zsolt Neméth. An approach for virtual appliance distribution for service deployment. *Future Generation Computer Systems*, 27(3):280 – 289, 2011.

[108] Alexander Keller and Heiko Ludwig. The WSLA framework: Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management*, 11:57–81, 2003. 10.1023/A:1022445108617.

[109] Jeffrey O. Kephart and Rajarshi Das. Achieving self-management via utility functions. *IEEE Internet Computing*, 2007.

[110] Jeffrey O. Kephart and William E. Walsh. An artificial intelligence perspective on autonomic computing policies. In *Fifth IEEE International Workshop on Policies for Distributed Systems and Networks, POLICY 2004*, 2004.

[111] J.O. Kephart and D.M. Chess. The vision of autonomic computing. *Computer*, 36(1):41 – 50, January 2003.

[112] Attila Kertész and Péter Kacsuk. Gmbs: A new middleware service for making grids interoperable. *Future Generation Computer Systems*, 26(4):542 – 553, 2010.

[113] G. Khanna, K. Beaty, G. Kar, and A. Kochut. Application performance management in virtualized server environments. In *Network Operations and Management Symposium, 2006. NOMS 2006. 10th IEEE/IFIP*, pages 373–381, 2006.

[114] Bithika Khargharia, Salim Hariri, Ferenc Szidarovszky, Manal Houri, Hesham El-Rewini, Samee Ullah Khan, Ishfaq Ahmad, and Mazin S. Yousif. Autonomic power & performance management for large-scale data centers. In *IPDPS*, pages 1–8. IEEE, 2007.

[115] Bithika Khargharia, Salim Hariri, and Mazin S. Yousif. Autonomic power and performance management for computing systems. *Cluster Computing*, 11(2):167–181, 2008.

[116] Won Kim. Cloud computing: Status and prognosis. *Journal of Object Technology*, 8(1):65–72, 2009.

[117] Sonja Klingert, Thomas Schulze, and Christian Bunse. Managing energy-efficiency by utilizing GreenSLAs. In *2nd International Conference on Energy-Efficient Computing and Networking 2011 (e-energy 2011)*, New York, NY, USA, 2011.

[118] B. Koller and L. Schubert. Towards autonomous SLA management using a proxy-like approach. *Multiagent Grid Systems*, 3(3), 2007.

[119] V.A. Korthikanti and G. Agha. Analysis of parallel algorithms for energy conservation in scalable multicore architectures. In *Parallel Processing, 2009. ICPP '09. International Conference on*, pages 212 –219, September 2009.

[120] V.A. Korthikanti and G. Agha. Avoiding energy wastage in parallel applications. In *Green Computing Conference, 2010 International*, pages 149 –163, August 2010.

[121] Vijay Anand Korthikanti and Gul Agha. Towards optimizing energy costs of algorithms for shared memory architectures. In *Proceedings of the 22nd ACM symposium on Parallelism in algorithms and architectures*, SPAA '10, pages 157–165, New York, NY, USA, 2010. ACM.

[122] Giannis Koumoutsos, Spyros Denazis, and Kleanthis Thramboulidis. SLA e-negotiations, enforcement and management in an autonomic environment. *Modelling Autonomic Communications Environments*, pages 120–125, 2008.

[123] Giannis Koumoutsos and Kleanthis Thramboulidis. Towards a knowledge-base for building complex, proactive and service-oriented e-negotiation systems. In *MCETECH '08: Proceedings of the 2008 International MCETECH Conference on e-Technologies*, pages 178–189, Washington, DC, USA, 2008. IEEE Computer Society.

[124] Leanid Krautsevich, Aliaksandr Lazouski, Fabio Martinelli, and Artsiom Yautsiukhin. Risk-aware usage decision making in highly dynamic systems. In *International Conference on Internet Monitoring and Protection*, pages 29–34, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

[125] Marcel Kyas, Cristian Prisacariu, and Gerardo Schneider. Run-time monitoring of electronic contracts. In Sungdeok Cha, Jin-Young Choi, Moonzoo Kim, Insup Lee, and Mahesh Viswanathan, editors, *Automated Technology for Verification and Analysis*, volume 5311 of *Lecture Notes in Computer Science*, pages 397–407. Springer Berlin / Heidelberg, 2008. 10.1007/978-3-540-88387-6_34.

[126] Paweł P. Łabaj, German G. Leparc, Bryan E. Linggi, Lye Meng Markillie, H. Steven Wiley, and David P. Kreil. Characterization and improvement of RNA-Seq precision in quantitative transcript expression profiling. *Bioinformatics*, 27(13):i383–i391, 2011.

[127] Ben Langmead, Cole Trapnell, Mihai Pop, and Steven Salzberg. Ultrafast and memory-efficient alignment of short DNA sequences to the human genome. *Genome Biology*, 10(3):R25, 2009.

[128] Stefan M. Larson, Christopher D. Snow, Michael R. Shirts, and Vijay S. Pande. Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology. *Computational Genomics*, 2002.

[129] Leon S. Lasdon. *Optimization Theory for Large Systems*. Dover Books on Mathematics. Dover Publications, 2011.

[130] Kevin Lee, Norman W. Paton, Rizos Sakellariou, and Alvaro A. A. Fernandes. Utility driven adaptive workflow execution. In *Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, CCGRID '09, pages 220–227, Washington, DC, USA, 2009. IEEE Computer Society.

[131] Kevin Lee, Rizos Sakellariou, Norman W. Paton, and Alvaro A. A. Fernandes. Workflow adaptation as an autonomic computing problem. In *Proceedings of the 2nd workshop on Workflows in support of large-scale science*, WORKS '07, pages 29–34, New York, NY, USA, 2007. ACM.

[132] Young Choon Lee, Chen Wang, Albert Y. Zomaya, and Bing Bing Zhou. Profit-driven service request scheduling in clouds. In *Cluster, Cloud and Grid Computing (CCGrid), 2010 10th IEEE/ACM International Conference on*, pages 15 –24, May 2010.

[133] A. Lenk, M. Klems, J. Nimis, S. Tai, and T. Sandholm. What's inside the cloud? an architectural map of the cloud landscape. In *Software Engineering Challenges of Cloud Computing, 2009. CLOUD '09. ICSE Workshop on*, pages 23 –31, May 2009.

[134] Nicola Leone, Gerald Pfeifer, Wolfgang Faber, Thomas Eiter, Georg Gottlob, Simona Perri, and Francesco Scarcello. The dlv system for knowledge representation and reasoning. *ACM Trans. Comput. Logic*, 7:499–562, July 2006.

[135] Hector Levesque, Fiora Pirri, and Ray Reiter. Foundations for the situation calculus. *Electronic Transactions on Artificial Intelligence*, 2:159–178, 1998.

[136] Wubin Li, J. Tordsson, and E. Elmroth. Modeling for dynamic cloud scheduling via migration of virtual machines. In *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pages 163 –171, December 2011.

[137] Harold C. Lim, Shivnath Babu, and Jeffrey S. Chase. Automated control for elastic storage. In *Proceedings of the 7th international conference on Autonomic computing*, ICAC '10, pages 1–10, New York, NY, USA, 2010. ACM.

[138] Ching-Chi Lin, Pangfeng Liu, and Jan-Jan Wu. Energy-efficient virtual machine provision algorithms for cloud systems. In *Utility and Cloud Computing, IEEE Internatonal Conference on*, pages 81–88, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

[139] Haikun Liu, Cheng-Zhong Xu, Hai Jin, Jiayu Gong, and Xiaofei Liao. Performance and energy modeling for live migration of virtual machines. In *Proceedings of the 20th international symposium on High performance distributed computing*, HPDC '11, pages 171–182, New York, NY, USA, 2011. ACM.

[140] Liang Liu, Hao Wang, Xue Liu, Xing Jin, Wen Bo He, Qing Bo Wang, and Ying Chen. Greencloud: a new architecture for green data center. In *Proceedings of the 6th international conference industry session on Autonomic computing and communications industry session*, ICAC-INDST '09, pages 29–38, New York, NY, USA, 2009. ACM.

[141] Lu Liu, O. Masfary, and Jianxin Li. Evaluation of server virtualization technologies for green IT. In *Service Oriented System Engineering (SOSE), 2011 IEEE 6th International Symposium on*, pages 79 –84, December 2011.

[142] A.C. Marosi and P. Kacsuk. Workers in the clouds. In *Parallel, Distributed and Network-Based Processing (PDP), 2011 19th Euromicro International Conference on*, pages 519 –526, February 2011.

[143] Attila Csaba Marosi, Gabor Kecskemeti, Attila Kertesz, and Peter Kacsuk. Fcm: an architecture for integrating iaas cloud systems. In *Proceedings of The Second International Conference on Cloud Computing, GRIDs, and Virtualization*, Rome, Italy, September 2011.

[144] Paul Marshall, Kate Keahey, and Tim Freeman. Elastic site: Using clouds to elastically extend site resources. In *Proceedings of the 2010 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing*, CCGRID '10, pages 43–52, Washington, DC, USA, 2010. IEEE Computer Society.

[145] Paul Marshall, Kate Keahey, and Tim Freeman. Improving utilization of infrastructure clouds. In *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, CCGRID '11, pages 205–214, Washington, DC, USA, 2011. IEEE Computer Society.

[146] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia distributed monitoring system: Design, implementation and experience. *Parallel Computing*, 30(7):817–840, 2004.

[147] Toni Mastelic, Vincent Emeakaroha, Michael Maurer, and Ivona Brandic. M4cloud - generic application level monitoring for resource-shared cloud environments. In *CLOSER 2012, 2nd International Conference on Cloud Computing and Services Science*, Porto, Portugal, April 2012.

[148] M. Maurer, I. Breskovic, V.C. Emeakaroha, and I. Brandic. Revealing the MAPE loop for the autonomic management of cloud infrastructures. In *Computers and Communications (ISCC), 2011 IEEE Symposium on*, pages 147 –152, July 2011.

[149] Michael Maurer, Ivona Brandic, Vincent C. Emeakaroha, and Schahram Dustdar. Towards knowledge management in self-adaptable clouds. In *IEEE 2010 Fourth International Workshop of Software Engineering for Adaptive Service-Oriented Systems*, Miami, USA, 2010.

[150] Michael Maurer, Ivona Brandic, and Rizos Sakellariou. Simulating autonomic SLA enactment in clouds using case based reasoning. In *ServiceWave 2010*, Ghent, Belgium, 2010.

[151] Michael Maurer, Ivona Brandic, and Rizos Sakellariou. Enacting SLAs in clouds using rules. In *Euro-Par 2011*, Bordeaux, France, 2011.

[152] Michael Maurer, Ivona Brandic, and Rizos Sakellariou. Enacting SLAs in clouds using knowledge management. *Future Generation Computer Systems (submitted)*, 2012.

[153] Michael Maurer, Ivona Brandic, and Rizos Sakellariou. Self-adaptive and resource-efficient SLA enactment for cloud computing infrastructures. In *5th International Conference on Cloud Computing (IEEE Cloud 2012) (submitted)*, Honolulu, HI, USA, June 2012.

[154] Michael Maurer, Vincent C. Emeakaroha, and Ivona Brandic. Economic analysis of the SLA mapping approach for cloud computing goods. In *Achieving Federated and Self-Manageable Cloud Infrastructures: Theory and Practice*. IGI Global, 2012.

[155] Michael Maurer, Vincent C. Emeakaroha, Ivona Brandic, and Jörn Altmann. Cost and benefit of the SLA mapping approach for defining standardized goods in cloud computing markets. In *International Conference on Utility and Cloud Computing (UCC 2010) in conjunction with the International Conference on Advanced Computing (ICoAC 2010)*, Chennai, India, December 2010.

[156] Michael Maurer, Vincent C. Emeakaroha, Ivona Brandic, and Jörn Altmann. Cost–benefit analysis of an SLA mapping approach for defining standardized cloud computing goods. *Future Generation Computer Systems*, 28(1):39 – 47, 2012.

[157] M. Mazzucco, D. Dyachuk, and R. Deters. Maximizing cloud providers' revenues via energy aware allocation policies. In *CLOUD 2010*, pages 131 –138, July 2010.

[158] R. Mehrotra, A. Dubey, S. Abdelwahed, and W. Monceaux. Large scale monitoring and online analysis in a distributed virtualized environment. In *Engineering of Autonomic and Autonomous Systems (EASe), 2011 8th IEEE International Conference and Workshops on*, pages 1 –9, April 2011.

[159] Peter Mell and Timothy Grance. The NIST definition of cloud computing. *Recommendations of the National Institue of Standards and Technology*, (Special Publication 800-145), September 2011.

[160] Xiaoqiao Meng, Canturk Isci, Jeffrey Kephart, Li Zhang, Eric Bouillet, and Dimitrios Pendarakis. Efficient resource provisioning in compute clouds via VM multiplexing. In *Proceeding of the 7th international conference on Autonomic computing*, ICAC '10, pages 11–20, New York, NY, USA, 2010. ACM.

[161] Nirav Merchant, John Hartman, Sonya Lowry, Andrew Lenards, David Lowenthal, and Edwin Skidmore. Leveraging cloud infrastructure for life science research laboratories: A generalized view. In *International Workshop on Cloud Computing at OOPSLA09*, Orlando, USA, 2009.

[162] Karl Moss. *Java Servlets*. McGraw-Hill, Inc., New York, NY, USA, 2nd edition, 1999.

[163] Vinod Muthusamy and Hans-Arno Jacobsen. SLA-driven distributed application development. In Ahamed et al. [28], pages 31–36.

[164] Amit Nathani, Sanjay Chaudhary, and Gaurav Somani. Policy based resource allocation in iaas cloud. *Future Generation Computer Systems*, 28(1):94 – 103, 2012.

[165] D. Neumann, J. Stößer, and C. Weinhardt. Bridging the adoption gap – developing a roadmap for trading in grids. *Electronic Markets*, 18(1):65–74, 2008.

[166] Ilkka Niemelä, Patrik Simons, and Tommi Syrjänen. Smodels: A system for answer set programming. *CoRR*, cs.AI/0003033, 2000.

[167] Jens Nimis, Arun Anandasivam, Nikolay Borissov, Garry Smith, Dirk Neumann, Niklas Wirström, Erel Rosenberg, and Matteo Villa. SORMA - business cases for an open grid market: Concept and implementation. In Springer, editor, *5th international workshop on Grid Economics and Business Models (GECON '08)*, pages 173 – 184, 2008.

[168] N. Oldham, K. Verma, A. P. Sheth, and F. Hakimpour. Semantic ws-agreement partner selection. In *15th International Conference on World Wide Web, WWW 2006*, Edinburgh, Scotland, UK, May 2006.

[169] A. Orlov. Project consequence. *Science and Technology Magazine*, 1:62–63, 2008.

[170] D. Ouelhadj, J. Garibaldi, J. MacLaren, R. Sakellariou, and K. Krishnakumar. A multi-agent infrastructure and a service level agreement negotiation protocol for robust scheduling in grid computing. In Peter Sloot, Alfons Hoekstra, Thierry Priol, Alexander Reinefeld, and Marian Bubak, editors, *Advances in Grid Computing - EGC 2005*, volume 3470 of *Lecture Notes in Computer Science*, pages 651–660. Springer Berlin / Heidelberg, 2005. 10.1007/11508380_66.

[171] 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 *Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007*, EuroSys '07, pages 289–302, New York, NY, USA, 2007. ACM.

[172] Yi-Lun Pan, Chang-Hsing Wu, Hsi-En Yu, Hui-Shan Chen, and Weicheng Huang. Ezilla toolkit - one click to build private cloud easily. In *Utility and Cloud Computing, IEEE Internatonal Conference on*, pages 332–333, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

[173] A. Paschke, M. Bichler, and J Dietrich. Contractlog: An approach to rule based monitoring and execution of service level agreements. In *International Conference on Rules and Rule Markup Languages for the Semantic Web*, Galway, Ireland, 2005.

[174] Adrian Paschke and Martin Bichler. Knowledge representation concepts for automated SLA management. *Decision Support Systems*, 46(1):187–205, 2008.

[175] Vinicius Petrucci, Orlando Loques, and Daniel Mossé. A dynamic optimization model for power and performance management of virtualized clusters. In *e-Energy '10*, pages 225–233, New York, NY, USA, 2010. ACM.

[176] Guillaume Pierre, Ismail El Helw, Corina Stratan, Ana Oprescu, Thilo Kielmann, Thorsten Schütt, Matej Artač, and Aleş Černivec. Conpaas: an integrated runtime environment for elastic cloud applications. In *Proceedings of the Middleware conference*, December 2011.

[177] L. Puente-Maury, P. Mejia-Alvarez, and L.E. Leyva-del Foyo. A binary integer linear programming-based approach for solving the allocation problem in multiprocessor partitioned scheduling. In *Electrical Engineering Computing Science and Automatic Control (CCE), 2011 8th International Conference on*, pages 1 –6, oct. 2011.

[178] Andres J. Ramirez, David B. Knoester, Betty H.C. Cheng, and Philip K. McKinley. Applying genetic algorithms to decision making in autonomic computing systems. In *Proceedings of the 6th international conference on Autonomic computing*, ICAC '09, pages 97–106, New York, NY, USA, 2009. ACM.

[179] Jia Rao, Xiangping Bu, Cheng-Zhong Xu, Leyi Wang, and George Yin. Vconf: a reinforcement learning approach to virtual machines auto-configuration. In *ICAC '09*, pages 137–146, New York, NY, USA, 2009. ACM.

[180] Paulo Antonio Leal Rego, Emanuel Ferreira Coutinho, Danielo Goncalves Gomes, and Jose Neuman de Souza. FairCPU: Architecture for allocation of virtual machines using processing features. In *Utility and Cloud Computing, IEEE Internatonal Conference on*, pages 371–376, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

[181] B.P. Rimal, Eunmi Choi, and I. Lumb. A taxonomy and survey of cloud computing systems. In *INC, IMS and IDC, 2009. NCM '09. Fifth International Joint Conference on*, pages 44 –51, August 2009.

[182] M. Risch and J. Altmann. Enabling open cloud markets through WS-agreement extensions. In *Service Level Agreements in Grids Workshop, in conjunction with GRID 2009*, CoreGRID Springer Series, Banff, Canada 2009.

[183] M. Risch, J. Altmann, L. Guo, A. Fleming, and C. Courcoubetis. The gridecon platform: A business scenario testbed for commercial cloud services. In *6th international Workshop on Grid Economics and Business Models*, Delft, The Netherlands, August 2009.

[184] Marcel Risch, Ivona Brandic, and Jörn Altmann. Using SLA mapping to increase market liquidity. In *NFPSLAM-SOC 2009 in conjunction with The 7th International Joint Conference on Service Oriented Computing*, Stockholm, Sweden, November 2009.

[185] Benny Rochwerger et al. The RESERVOIR model and architecture for open federated cloud computing. *IBM Journal of Research and Development*, 53(4), 2009.

[186] Luis Rodero-Merino, Luis M. Vaquero, Victor Gil, Fermin Galan, Javier Fontan, Ruben S. Montero, and Ignacio M. Llorente. From infrastructure delivery to service management in clouds. *Future Generation Computer Systems*, 26(8):1226–1240, October 2010.

[187] Paolo Romano. Automation of in-silico data analysis processes through workflow management systems. *Briefings in Bioinformatics*, 9(1):57–68, October 2007.

[188] Rizos Sakellariou and Viktor Yarmolenko. Job scheduling on the grid: Towards SLA-based scheduling. In Lucio Grandinetti, editor, *High Performance Computing and Grids in Action*, volume 16 of *Advances in Parallel Computing*, pages 207–222. IOS Press, 2008.

[189] Prasad Saripalli, G.V.R. Kiran, R. Ravi Shankar, Harish Narware, and Nitin Bindal. Load prediction and hot spot detection models for autonomic cloud computing. In *Utility and Cloud Computing, IEEE Internatonal Conference on*, pages 397–402, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

[190] M. Sedaghat, F. Hernandez, and E. Elmroth. Unifying cloud management: Towards overall governance of business level objectives. In *Cluster, Cloud and Grid Computing (CCGrid), 2011 11th IEEE/ACM International Symposium on*, pages 591 –597, May 2011.

[191] Weiming Shi and Bo Hong. Towards profitable virtual machine placement in the data center. In *Utility and Cloud Computing, IEEE Internatonal Conference on*, pages 138–145, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

[192] Damian Smedley, Morris A. Swertz, Katy Wolstencroft, Glenn Proctor, Michael Zouberakis, Jonathan Bard, John M. Hancock, and Paul Schofield. Solutions for data integration in functional genomics: a critical assessment and case study. *Briefings in Bioinformatics*, 9(6):532–544, September 2008.

[193] Biao Song, M.M. Hassan, and Eui nam Huh. A novel heuristic-based task selection and allocation framework in dynamic collaborative cloud service platform. In *Cloud Computing Technology and Science (CloudCom), 2010 IEEE Second International Conference on*, pages 360 –367, December 2010.

[194] Mukundan Sridharan, Prasad Calyam, Aishwarya Venkataraman, and Alex Berryman. Defragmentation of resources in virtual desktop clouds for cost-aware utility-optimal allocation. In *Utility and Cloud Computing, IEEE International Conference on*, pages 253–260, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

[195] Mark Stillwell, David Schanzenbach, Frederic Vivien, and Henri Casanova. Resource allocation algorithms for virtualized service hosting platforms. *Journal of Parallel and Distributed Computing*, 70(9):962 – 974, 2010.

[196] Akiyoshi Sugiki and Kazuhiko Kato. An extensible cloud platform inspired by operating systems. In *Utility and Cloud Computing, IEEE International Conference on*, pages 306–311, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

[197] Ashish Thusoo, Joydeep Sen Sarma, Namit Jain, et al. Hive - a warehousing solution over a map-reduce framework. In *VLDB*, 2009.

[198] Cole Trapnell, Lior Pachter, and Steven L. Salzberg. Tophat: discovering splice junctions with RNA-Seq. *Bioinformatics*, 25(9):1105–1111, 2009.

[199] Luis M. Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. Dynamically scaling applications in the cloud. *SIGCOMM Comput. Commun. Rev.*, 41:45–52, 2011.

[200] Suresh Venugopal, Sravan Desikan, and Karthikeyan Ganesan. Effective migration of enterprise applications in multicore cloud. In *Utility and Cloud Computing, IEEE Internatonal Conference on*, pages 463–468, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

[201] A. Verma, G. Kumar, R. Koller, and A. Sen. CosMig: Modeling the impact of reconfiguration in a cloud. In *Modeling, Analysis Simulation of Computer and Telecommunication Systems (MASCOTS), 2011 IEEE 19th International Symposium on*, pages 3 –11, July 2011.

[202] William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Cost of virtual machine live migration in clouds: A performance evaluation. In *Proceedings of the 1st International Conference on Cloud Computing*, CloudCom '09, pages 254–265, Berlin, Heidelberg, 2009. Springer-Verlag.

[203] George A. Vouros, Andreas Papasalouros, Konstantinos Tzonas, Alexandros Valarakos, Konstantinos Kotis, Jorge-Arnulfo Quiane-Ruiz, Philippe Lamarre, and Patrick Valduriez. A semantic information system for services and traded resources in grid e-markets. *Future Generation Computer Systems*, 26(7):916–933, July 2010.

[204] Qingling Wang and Carlos A. Varela. Impact of cloud computing virtualization strategies on workloads' performance. In *Utility and Cloud Computing, IEEE Internatonal Conference on*, pages 130–137, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

[205] Brian J. Watson, Manish Marwah, Daniel Gmach, Yuan Chen, Martin Arlitt, and Zhikui Wang. Probabilistic performance modeling of virtualized resource allocation. In *Proceedings of the 7th international conference on Autonomic computing*, ICAC '10, pages 99–108, New York, NY, USA, 2010. ACM.

[206] Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Sandpiper: Black-box and gray-box resource management for virtual machines. *Computer Networks*, 53(17):2923 – 2938, 2009.

[207] Jing Xu, Ming Zhao, José Fortes, Robert Carpenter, and Mazin S. Yousif. Autonomic resource management in virtualized data centers using fuzzy logic-based approaches. *Cluster Computing*, 11(3):213–227, 2008.

[208] V. Yarmolenko and R. Sakellariou. An evaluation of heuristics for SLA based parallel job scheduling. In *Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International*, page 8 pp., April 2006.

[209] Victor Yarmolenko and Rizos Sakellariou. Towards increased expressiveness in service level agreements. *Concurrency and Computation: Practice and Experience*, 19:1975–1990, 2007.

[210] Y.O. Yazir, C. Matthews, R. Farahbod, S. Neville, A. Guitouni, S. Ganti, and Y. Coady. Dynamic resource allocation in computing clouds using distributed multiple criteria decision analysis. In *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on*, pages 91–98, July 2010.

[211] L. Youseff, M. Butrico, and D. Da Silva. Toward a unified ontology of cloud computing. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1 –10, November 2008.

[212] Yi Yu and Saleem Bhatti. Energy measurement for the cloud. In *Parallel and Distributed Processing with Applications, International Symposium on*, pages 619–624, Los Alamitos, CA, USA, 2010. IEEE Computer Society.

[213] Qi Zhang, Quanyan Zhu, and Raouf Boutaba. Dynamic resource allocation for spot markets in cloud computing environments. In *IEEE Internatonal Conference on Utility and Cloud Computing*, pages 178–185, Los Alamitos, CA, USA, 2011. IEEE Computer Society.

APPENDIX A

Curriculum Vitae

Michael Maurer

Personal Information

Date of birth	May 26th, 1983
Place of birth	Eisenstadt, Austria
Citizenship	Austria
Phone	+43-1-58801-18457
E-mail	maurer@infosys.tuwien.ac.at
Web	http://www.infosys.tuwien.ac.at/staff/maurer
Affiliation	Vienna University of Technology Distributed Systems Group Argentinierstraße 8 A-1040 Wien, Austria

Education

08/2009 - ongoing	PhD program in Computer Science at TU Vienna, Austria.
12/2007 - 04/ 2009	Master's program Computational Intelligence at TU Vienna, Austria - graduation with distinction.
08/2008 - 12/2008	Exchange semester for master's program Computational Intelligence at City College of New York in New York, NY, USA.
02/2008 - 06/2008	Exchange semester for master's program Computational Intelligence at the Pavol Jozef Šafárik University in Košice, Slovakia, with Slovak language intensive course.

12/2007	European Business Competence* Licence (EBC*L).
10/2001 - 11/2007 (w/o 2002-2003)	Studies of Applied Mathematics (equivalent to MSc, with specialization in Computer Science) at TU Vienna, Austria - graduation with distinction.
07/2002 - 09/2003	Social year (called Gedenkdienst) at the *Fondation Auschwitz* in Brussels, Belgium.
06/2001	School leaving certificate (Matura) at grammar school BG&BRG Bruck/Leitha with a paper in Mathematics dealing with game theory.
08/1999 - 12/1999	Exchange semester at the St. Johnsbury Academy in St. Johnsbury, VT, USA.
1993 - 2001	Grammar school BG&BRG Bruck/Leitha (emphasis on modern languages).
1991 - 2001	Music school (keyboard, piano) in Bruck/Leitha, Austria.

Professional Experience

08/2009 - ongoing	Project assistant at TU Vienna, Distributed Systems Group.
03/2010	Short Term Scientific Mission (STSM) at the University of Manchester within the COST Action IC 0804 "Energy efficiency in large scale distributed systems".
02 - 03/2009	Advancing a database system to record and manage scouting reports within Austrian professional soccer.
10/2007 - 02/2008	Team-development of a web-based administration tool at the grammar school *GRG 17 Geblergasse* together with enhancements of the web application at the *GRG 21 Bertha von Suttner*.
10/2005 - 01/2008	Tutor at the Institute of Information Systems at the TU Vienna for the courses Data Modeling and Database Systems.
10/2006 - 02/2007	Team-development of a web-based management system for the grammar school *GRG 21 Bertha von Suttner* in Vienna.
07/2002 - 09/2003	Administration of the computer network, web design, translation of letters between English, French and German, assistance in the library, writing reviews of German and English books at the *Fondation Auschwitz* in Brussels, Belgium.
08/2001	Summer job at Denzel (Austrian car company).
1999 - 2007	Tutoring high-school students in English, mathematics, German and French.

Scholarships and Awards

08 - 12/2008	Joint-Study scholarship granted by TU Vienna for exchange semester at CCNY in New York, USA.
02 - 06/2008	CEEPUS II Free Mover scholarship granted by ÖAD (Österr. Austauschdienst) and SAIA (Slovak Academic Information Agency) for exchange semester at UPJŠ in Košice, Slovakia.
10/2007 - 06/2008	TUtheTOP, the High Potential Program at the TU Vienna.
03/2007	ATHENS program in Warsaw, Poland. Course: Numerical Methods.
08/2005	Three-week Summer language course of Slovak at the SAS (Studia Academica Slovaca) in Bratislava, Slovakia.

06/1999 Summer School of Lower Austria for Highly Talented Students in Physics and Mathematics.

Scientific Activities

Projects

- FoSII - Foundations of Self-governing ICT Infrastructures funded by Vienna Science and Technology Fund (WWTF), ICT call 2008.
- HALEY - Holistic Energy Efficient Management of Hybrid Clouds, TU Vienna Science Award 2011.

Research Visits

- Short Term Scientific Mission (STSM), Increasing Energy Efficiency by Incorporating VM Resource Allocation and VM Placement on Clouds, COST Action IC0804 on Energy Efficient Large Scale Distributed Systems, carried out at the Insitut de Recherche en Informatique de Toulouse (IRIT), Université Paul Sabatier, Toulouse, France, from 14-March-2011 to 01-April-2011, in cooperation with Damien Borgetto, Georges Da Costa and Jean-Marc Pierson.
- Short Term Scientific Mission (STSM), A step towards the incorporation of energy efficiency in autonomic SLA management, COST Action IC0804 on Energy Efficient Large Scale Distributed Systems, carried out at the University of Manchester, School of Computer Science, UK, from 08-March-2010 to 26-March-2010, in cooperation with Rizos Sakellariou.

Scientific Talks

- "Towards Energy-efficient Cloud Computing". Vienna Scientific Cluster (VSC) workshop, February 27-28 2012, Hotel Wende, Neusiedl am See, Austria.
- "Achieving SLA-aware and energy-efficient management of IaaS Cloud Computing infrastructures". Cost Action Meeting (IC0804 on Energy Efficient Large Scale Distributed Systems), November 7-8 2011, International Hellenic University, Thessaloniki, Greece (invited).
- "Enacting SLAs in Clouds Using Rules". Euro-Par 2011, Bordeaux, France, August 29 - September 2, 2011.
- "Energy Efficient Autonomic Management of Clouds". Research visit at the Insitut de Recherche en Informatique de Toulouse (IRIT), Université Paul Sabatier, Toulouse, France, March 18, 2011 (invited).
- "Simulating Autonomic SLA Enactment in Clouds using Case Based Reasoning". ServiceWave 2010, Ghent, Belgium, December 13-15, 2010.
- "Towards Knowledge Management in Self-adaptable Clouds". IEEE 2010 Fourth International Workshop of Software Engineering for Adaptive Service-Oriented Systems (SEASS '10), in conjunction with ICWS 2010 and SCC 2010, Miami, Florida, USA, July 5-10, 2010.
- "Towards Knowledge Management in Clouds - Prevention of SLA Violations vs. Minimization of Energy Consumption". Cost Action Meeting (IC0804 on Energy Efficient Large Scale Distributed Systems, Focus Group Green wired networks), June 10-11 2010, University of Lyon, Lyon, France.

Program Committee Member

- CLOUD COMPUTING 2012 - The Third International Conference on Cloud Computing, GRIDs, and Virtualization, July 22-27, 2012, Nice, France.

Reviewer for Journals

- Business and Information Systems Engineering / Wirtschaftsinformatik (Gabler)
- Computing (Springer)
- Concurrency and Computation: Practice and Experience (Wiley)
- Future Generation Computer Systems (Elsevier)
- IEEE Transactions on Services Computing (IEEE Computer Society)
- Information Sciences (Elsevier)
- Journal of Systems and Software (Elsevier)
- Scientific Programming (IOS Press)

Reviewer for Conferences and Workshops

- HPCC 2012 - The 14th IEEE International Conference on High Performance Computing and Communications
- IEEE ICWS 2012 - 19th International Conference on Web Services
- Euro-Par 2012 - International European Conference on Parallel and Distributed Computing
- SEAMS 2012 - 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems
- WWW 2012 - World Wide Web Conference
- WORKS 2011 - The 6th Workshop on Workflows in Support of Large-Scale Science in conjunction with SC 2011
- HPCC 2011 - 13th IEEE International Conference on High Performance Computing and Communications
- WoSS - 2nd Workshop on Software Services: Cloud Computing and Applications based on Software Services
- GreenCom 2011 - The 2011 IEEE/ACM International Conference on Green Computing and Communications
- ICSE 2011 - 33rd International Conference on Software Engineering
- CCGRID 2011 - The 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing
- UCC 2010 - 3rd International Conference on Utility and Cloud Computing
- CSE 2010 - The 13th IEEE International Conference on Computational Science and Engineering
- SEE 2010 - First International Workshop on Services, Energy and Ecosystem
- ICSOC 2010 - International Conference on Service Oriented Computing
- ICEBE 2010 - 7th IEEE International Conference on e-Business Engineering

- PDGC 2010 - 1st International Conference on Parallel, Distributed and Grid Computing
- IEEE ICSM 2010 - 26th IEEE International Conference on Software Maintenance
- SEFM 2010 - 8th IEEE International Conference on Software Engineering and Formal Methods
- IC3 - 3rd International Conference on Contemporary Computing
- IEEE ICWS 2010 - The 8th International Conference on Web Services
- IADIS International Conference WWW/INTERNET 2009
- SOCA '09 - IEEE International Conference on Service-Oriented Computing and Applications
- WORKS 2009 - The 4th Workshop on Workflows in Support of Large-Scale Science In conjunction with SC 2009

Publications

Refereed Publications in Conference Proceedings

1. Damien Borgetto*, **Michael Maurer***, Georges Da Costa, Jean-Marc Pierson, and Ivona Brandic. Energy-efficient and SLA-aware managament of iaas clouds. In Third international conference on future energy systems (e-Energy 2012), Madrid, Spain, May 2012. (accepted). (* contributed equally)
2. Drazen Lucanin, **Michael Maurer**, Toni Mastelic, and Ivona Brandic. Energy Efficient Service Delivery in Clouds in Compliance with the Kyoto Protocol. E2DC - 1st International Workshop on Energy-Efficient Data Centers held in conjunction with e-Energy 2012 - Third International Conference on Future Energy Systems, May 9-11 2012, Madrid, Spain. (accepted).
3. Toni Mastelic, Vincent Emeakaroha, **Michael Maurer**, Ivona Brandic. M4Cloud - Generic Application Level Monitoring for Resource-Shared Cloud Environments. CLOSER 2012, 2nd International Conference on Cloud Computing and Services Science, April 18-21, 2012, Porto, Portugal.
4. Gabor Kecskemeti, **Michael Maurer**, Ivona Brandic, Attila Kertesz, Zsolt Nemeth and Schahram Dustdar. Facilitating self-adaptable Inter-Cloud management. 20th Euromicro International Conference on Parallel, Distributed, and Network-Based Processing PDP 2012. Munich, Germany, 15-17 February, 2012.
5. Ivan Breskovic, **Michael Maurer**, Vincent C. Emeakaroha, Ivona Brandic, Schahram Dustdar. Cost-Efficient Utilization of Public SLA Templates in Autonomic Cloud Markets. 4th IEEE International Conference on Utility and Cloud Computing (UCC 2011), December 5-8, 2011, Melbourne, Australia.
6. Vincent Chimaobi Emeakaroha*, Pawel Labaj*, **Michael Maurer***, Ivona Brandic and David P. Kreil. Optimizing Bioinformatics Workflows for Data Analysis Using Cloud Management Techniques. The 6th Workshop on Workflows in Support of Large-Scale Science (WORKS11), in conjunction with Supercomputing 2011, Seattle, November 12-18, 2011. (* contributed equally)
7. **Michael Maurer**, Ivona Brandic, Rizos Sakellariou. Enacting SLAs in Clouds Using Rules. Euro-Par 2011, Bordeaux, France, August 29 - September 2, 2011.

8. **Michael Maurer**, Ivan Breskovic, Vincent C. Emeakaroha, Ivona Brandic. Revealing the MAPE Loop for the Autonomic Management of Cloud Infrastructures. Workshop on Management of Cloud Systems (MoCS 2011), in association with the IEEE Symposium on Computers and Communications (ISCC 2011), 28 June 2011, Kerkyra (Corfu) Greece.

9. Ivan Breskovic, **Michael Maurer**, Vincent C. Emeakaroha, Ivona Brandic, Jörn Altmann. Towards Autonomic Market Management in Cloud Computing Infrastructures, International Conference on Cloud Computing and Services Science - CLOSER 2011, 7-9 May, 2011 Noordwijkerhout, the Netherlands.

10. **Michael Maurer**, Vincent C. Emeakaroha, Ivona Brandic, Joern Altmann. Cost and Benefit of the SLA Mapping Approach for Defining Standardized Goods in Cloud Computing Markets. International Conference on Utility and Cloud Computing (UCC 2010) in conjunction with the International Conference on Advanced Computing (ICoAC 2010), December 14-16, 2010, Chennai, India.

11. **Michael Maurer**, Ivona Brandic and Rizos Sakellariou. Simulating Autonomic SLA Enactment in Clouds using Case Based Reasoning. ServiceWave 2010, Ghent, Belgium, December 13-15 2010.

12. Vincent C. Emeakaroha, **Michael Maurer**, Ivona Brandic, Schahram Dustdar: FoSII - Foundations of Self-Governing ICT Infrastructures. ERCIM NEWS, Number 83 (2010), October 2010, p. 40 - 41.

13. Ivona Brandic, Vincent C. Emeakaroha, **Michael Maurer**, Sandor Acs, Attila Kertész, Gábor Kecskeméti, Schahram Dustdar. LAYSI: A Layered Approach for SLA-Violation Propagation in Self-manageable Cloud Infrastructures. The First IEEE International Workshop on Emerging Applications for Cloud Computing (CloudApp 2010), In conjunction with the 34th Annual IEEE International Computer Software and Applications Conference Seoul, Korea, July 19-23 2010.

14. **Michael Maurer**, Ivona Brandic, Vincent C. Emeakaroha, Schahram Dustdar. Towards Knowledge Management in Self-adaptable Clouds. IEEE 2010 Fourth International Workshop of Software Engineering for Adaptive Service-Oriented Systems (SEASS '10), in conjunction with ICWS 2010 and SCC 2010, Miami, Florida, USA, July 5-10, 2010.

15. Vincent C. Emeakaroha, Ivona Brandic, **Michael Maurer**, Schahram Dustdar. Low Level Metrics to High Level SLAs - LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in Cloud environments. The 2010 High Performance Computing and Simulation Conference (HPCS 2010), in conjunction with The 6th International Wireless Communications and Mobile Computing Conference (IWCMC 2010), June 28 - July 2, 2010, Caen, France.

Refereed Publications in Journals

1. **Michael Maurer**, Vincent C. Emeakaroha, Ivona Brandic, Joern Altmann. Cost-Benefit Analysis of an SLA Mapping Approach for Defining Standardized Cloud Computing Goods. Future Generation Computer Systems, 2011, doi:10.1016/j.future.2011.05.023.

2. Vincent C. Emeakaroha, Ivona Brandic, **Michael Maurer**, Schahram Dustdar. SOA and QoS Management for Cloud Computing. In: Cloud computing: methodology, system, and applications. Editors: Lizhe Wang, Rajiv Ranjan, Jinjun Chen, Boualem Benatallah, CRC, Taylor & Francis group, 2011.

3. Vincent C. Emeakaroha, Ivona Brandic, **Michael Maurer**, Schahram Dustdar. Cloud Resource Provisioning and SLA Enforcement Via LoM2HiS Framework, Concurrency and Computation: Practice and Experience, 2011.

4. Vincent C. Emeakaroha, **Michael Maurer**, Ivona Brandic, Schahram Dustdar. FoSII - Foundations of Self-Governing ICT Infrastructures. Special Theme: "Cloud Computing Platforms, Software, and Applications". ERCIM News No. 83 (October 2010).

Other Publications

1. Ivona Brandic, Vincent C. Emeakaroha, **Michael Maurer**, Schahram Dustdar. Including Energy Efficiency into Self-adaptable Cloud Services. Proceedings of the COST Action IC0804 on Energy Efficiency in Large Scale Distributed Systems 1st Year, J. Pierson, H. Hlavacs (Ed.), COST Office, 2010, (invited), ISBN: 978-2-917490-10-5, p. 84 - 87.
2. Ivona Brandic, **Michael Maurer**, Rizos Sakellariou. Simulating Autonomic SLA Enactment in Clouds using Case Based Reasoning, Proceedings of the COST Action IC0804 - 2nd Year, 2011, p. 36 - 40.
3. Vincent C. Emeakaroha, **Michael Maurer**, Ivan Breskovic, Ivona Brandic. Time Shared VMs and Monitoring of Time Shared VMs. Proceedings of the COST Action IC0804 on Energy Efficiency in Large Scale Distributed Systems, 2nd Year, J. Pierson, H. Hlavacs (Ed.), COST Office, 2011, p. 47-51.
4. **Michael Maurer**, Ivona Brandic, Rizos Sakellariou. Enacting SLAs in Clouds Using Rules, Proceedings of the COST Action IC0804 on Energy Efficiency in Large Scale Distributed Systems, 2nd Year, J. Pierson, H. Hlavacs (Ed.), COST Office, 2011, p. 132-136.
5. **Michael Maurer**. Increasing Energy Efficiency by Incorporating VM Resource Allocation. In: COST Action 804 Newsletter, Vol. 3, June 2011.
6. **Michael Maurer**. A step towards the incorporation of energy efficiency in autonomic SLA management. In: COST Action 804 Newsletter, Vol. 2, October 2010.

Books

1. **Michael Maurer**, Approval Voting - A characterization and compilation of advantages and drawbacks in respect of other voting procedures, VDM Verlag Dr. Mueller, Saarbrücken, 2008.

Extra-curricular Activities

2004 - ongoing	Team leader of junior scouts (children from 7-10 years) at the scouts group Bruck/Leitha. Organization of weekly meetings, excursions and summer camps.
07/2010	Leadership and Soft Skills training (Wood Badge course, PPÖ).
12/2006 - 09/2007	Lead responsible for the organization and realization of a one-day international event (100 years of Scouting – We celebrate) with participating Slovak, Hungarian and Austrian Scouts in Bruck/Leitha.
2001 - 2005	Gedenkdienst. Study trips to Auschwitz and Theresienstadt; meeting survivors of the Holocaust; civil service at the *Fondation Auschwitz* in Brussels; maintaining and enhancing the club's library in Vienna.

Hobbies and Special Interests

Scouts, Skiing, Snowboarding, Star Trek, Gedenkdienst, Traveling (by means of Interrail through Europe, as well as USA, Japan, Libya), physics, astronomy, geocaching, history, foreign languages, playing the piano.