Planning and Optimization During the Life-Cycle of Service Level Agreements for Cloud Computing

Dissertation
zur Erlangung des mathematisch-naturwissenschaftlichen Doktorgrades
"Doctor rerum naturalium"
der Georg-August-Universität Göttingen

im Promotionsprogramm Computer Science (PCS)
der Georg-August University School of Science (GAUSS)

vorgelegt von

Kuan Lu
aus Shanxi China

Göttingen, 2015

Betreuungsausschuss

Prof. Dr. Ramin Yahyapour,
Gesellschaft für wissenschaftliche Datenverarbeitung Göttingen mbH (GWDG),
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Jens Grabowski,
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Xiaoming Fu,
Institut für Informatik, Georg-August-Universität Göttingen

Mitglieder der Prüfungskommission

Referent: Prof. Dr. Ramin Yahyapour,
Gesellschaft für wissenschaftliche Datenverarbeitung Göttingen mbH (GWDG),
Institut für Informatik, Georg-August-Universität Göttingen

Korreferent: Prof. Dr. Jens Grabowski,
Institut für Informatik, Georg-August-Universität Göttingen

Weitere Mitglieder der Prüfungskommission

Prof. Dr. Carsten Damm,
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Dieter Hogrefe,
Institut für Informatik, Georg-August-Universität Göttingen

Prof. Dr. Wolfgang May,
Institut für Informatik, Georg-August-Universität Göttingen

Tag der mündlichen Prüfung: 16. Februar 2015

Acknowledgments

There could be countless pages of thanks to all the people for your support over the years as I moved from an idea to a completed study. This would have never been possible without your unwavering encouragement and help. It is true that "Life is what happens" when I am completing the dissertation. In these five years, I gradually realized that actually the life is to take what you should get not what you want to get. Then feel it, enjoy it and love it.

Firstly, special gratitude I would like to express to my doctoral supervisor Prof. Dr. Ramin Yahyapour, who made it possible for me to conduct my research with his experience and knowledge. I would also like to thank Prof. Dr. Jens Grabowski for supervising my work and providing advice and directions. I am very grateful for the effort that Constantinos Kotsokalis, Philipp Wieder, Xiong Wang and John Kennedy put into the proof-reading of this thesis. Especially, I would like to thank Edwin Yaqub, Thomas Röblitz, Tibor Kálmán, Ali Imran Jehangiri, Peter Chronz, Bernd Schlör and Monir Abdullah for the interesting scientific discussions, cooperation and their constant encouragement. Moreover, I want to thank all my current and prior colleagues at GWDG and Dortmund University of Technology. Last but not least, to my wife Lin, daughter Qingcheng and my parents, all what I have done and will achieve is only for you.

Abstract

A Service Level Agreement (SLA) is an electronic contract between the consumer and the provider of a service. It governs their business relationship by clarifying expectations and obligations of participating entities, with regard to the service and its quality. SLAs are already the prime paradigm for the description of cloud computing services. Once an SLA is established, the provider has to ensure that service quality remains within certain acceptable levels; and comply with the customer's demands until the end of the service life time. However, managing the SLAs is still a technical challenge that requires significant effort to achieve autonomy, economy and efficiency. Current state-of-the-art in SLA management faces challenges such as SLA representation for cloud services; business-related SLA optimizations; service outsourcing and resource management. These areas constitute, as one would expect, major contemporary research topics. Hence, a structured methodology engineered for the management of the different phases of SLAs during its lifespan is of paramount importance, which indeed facilitates the realization of the cloud SLA management.

To this aim, I present diversified models and approaches in SLA lifecycle management that address the aforementioned challenges and enable automatic service modeling, negotiation, provisioning and monitoring. During the SLA creation phase, I outline how to improve and simplify the structures that model SLAs. Furthermore, another objective of my approach is to minimize implementation and outsourcing costs for reasons of competitiveness, while respecting business policies for profit and risk. During the SLA monitoring phase, I develop the strategies for virtual cloud resources selection and allocation during live migrations. Then, I apply an appropriate theoretical model for fine-grained yet simplified and practical monitoring of massive sets of SLAs, that separates the agreement's fault-tolerance concerns into multiple autonomous layers.

The work at hand contributes a blueprint for the GWDG and its scientific communities. The research that lead to this thesis was conducted as part of the SLA@SOI EU/FP7 Integrated Project (contract No. 216556).

Zusammenfassung

Ein Service Level Agreement (SLA) ist ein elektronischer Vertrag zwischen dem Kunden und dem Anbieter eines Services. Die beteiligten Partner klären ihre Erwartungen und Verpflichtungen in Bezug auf den Dienst und dessen Qualität. SLAs werden bereits für die Beschreibung von Cloud-Computing-Diensten eingesetzt. Der Diensteanbieter stellt sicher, dass die Dienstqualität erfüllt wird und mit den Anforderungen des Kunden bis zum Ende der vereinbarten Laufzeit übereinstimmt. Die Durchführung der SLAs erfordert einen erheblichen Aufwand, um Autonomie, Wirtschaftlichkeit und Effizienz zu erreichen. Der gegenwärtige Stand der Technik im SLA-Management begegnet Herausforderungen wie SLA-Darstellung für Cloud-Dienste, geschäftsbezogene SLA-Optimierungen, Dienste-Outsourcing und Ressourcenmanagement. Diese Gebiete schaffen zentrale und aktuelle Forschungsthemen. Das Management von SLAs in unterschiedlichen Phasen während ihrer Laufzeit erfordert eine dafür entwickelte Methodik. Dadurch wird die Realisierung von Cloud SLA-Management vereinfacht.

Ich präsentiere ein breit gefächertes Modell im SLA-Laufzeitmanagement, das die genannten Herausforderungen adressiert. Diese Herangehensweise ermöglicht eine automatische Dienstemodellierung, sowie Aushandlung, Bereitstellung und Monitoring von SLAs. Während der Erstellungsphase skizziere ich, wie die Modellierungsstrukturen verbessert und vereinfacht werden können. Ein weiteres Ziel von meinem Ansatz ist die Minimierung von Implementierungs- und Outsourcingkosten zugunsten von Wettbewerbsfähigkeit. In der SLA-Monitoringphase entwickle ich Strategien für die Auswahl und Zuweisung von virtuellen Cloud Ressourcen in Migrationsphasen. Anschließend prüfe ich mittels Monitoring eine größere Zusammenstellung von SLAs, ob die vereinbarten Fehlertoleranzen eingehalten werden.

Die vorliegende Arbeit leistet einen Beitrag zu einem Entwurf der GWDG und deren wissenschaftlichen Communities. Die Forschung, die zu dieser Doktorarbeit geführt hat, wurde als Teil von dem SLA@SOI EU/FP7 integriertem Projekt durchgeführt (contract No. 216556).

Contents

I Introduction **1**

1 Introduction **3**

1.1 Background to the Research . . . 3
1.1.1 Utility Computing . . . 3
1.1.2 Cloud Computing . . . 3
1.1.3 Service Level Agreement . . . 5
1.1.4 Service Lifecycle Management . . . 6
1.1.5 SLA Lifecycle Management . . . 6
1.1.5.1 Discover Service Provider . . . 7
1.1.5.2 SLA Definition . . . 7
1.1.5.3 Agreement Establishment . . . 8
1.1.5.4 SLA Monitoring . . . 8
1.1.5.5 SLA Removal Phase . . . 8
1.2 Motivation . . . 9
1.3 Contribution . . . 11
1.4 Impact . . . 12
1.5 Dissertation Organization . . . 14

II Requirements **15**

2 Requirements and Problem Statements **17**

2.1 Requirements . . . 17
2.2 Derivation and Selection of SLA Model . . . 18
2.3 Structural Suboptimal Problem of SLA in Negotiation . . . 19
2.4 Business Policies and Resource Planning in SLA Negotiation . . . 20
2.5 Resource Management for Multi-Domain Cloud in SLA Operation . . . 22
2.6 Fault Tolerance Management in SLA Operation . . . 23
2.7 Discussion . . . 24

III Models, Implementation and Evaluation 25

3 SLA and SLA Template Modeling During SLA Definition 27
3.1 Derivation and Selection of SLA Model Specification 27
3.2 Syntax of SLA and SLA Template 28
3.3 Discussion 29

4 Structural Optimization of SLA During Establishment 30
4.1 Scenario Introduction 30
4.2 Modeling SLA with ROBDD 32
4.3 ROBDD Structural Optimization 33
4.3.1 Term Rewriting System with Mutual Exclusiveness in ROBDD 33
4.3.2 ROBDD Variable Swap and Sifting Algorithm 36
4.3.3 Node Optimization 37
4.3.4 Path Optimization 37
4.3.5 Multicriteria Optimization Problem 38
4.4 Implementation and Experimental Verification 39
4.5 Discussion 44

5 Business Policies and Resource Planning During SLA Negotiation 45
5.1 Business Modeling of IaaS Cloud 46
5.1.1 Internal and External Implementation Cost 47
5.1.2 Profit 48
5.1.3 Failure Probability 49
5.1.4 Return on Investment 50
5.1.5 Complete Problem Definition 51
5.2 Architecture of Subcontracting 53
5.2.1 Subcontracting 53
5.2.2 Heuristic Selection Criteria of Service Providers 55
5.3 Local Resource Configuration 56
5.4 Resource Advance Reservation 57
5.4.1 Computational Geometry Representation 58
5.4.2 Service and Resource Availability 60
5.4.3 Virtual Fragments of Server 63
5.5 Implementation and Experimental Verification 64
5.5.1 Architecture and Implementation 64
5.5.2 Experimental Verification 65

5.5.2.1 Experimental Environment . . . 65
5.5.2.2 Generation of Workloads for Advance Reservation . . 67
5.5.2.3 Simulation Results . . . 69
5.6 Discussion . . . 71

6 Multi-Domain Resource Management During SLA Operation 73
6.1 Introduction . . . 73
6.2 Modeling of SLA Pricing and Penalty . . . 74
6.3 Resource Rearrangement . . . 77
6.4 Resource Selection and Allocation . . . 78
6.4.1 VM Selection . . . 79
6.4.2 VM Allocation . . . 80
6.4.3 VM Live Migration Downtime Estimator . . . 81
6.5 OpenStack SLA Management Framework . . . 81
6.6 Experimental Results . . . 84
6.6.1 Additional Resource Investment . . . 85
6.6.2 VM Migration . . . 85
6.7 Discussion . . . 89

7 Fault Tolerant SLA Management 92
7.1 Introduction . . . 92
7.2 Related Works . . . 92
7.3 Decentralization of SLA Management Using an Actor System . . . 93
7.3.1 Decentralization of IaaS SLA Management . . . 94
7.3.2 Modeling of Actors . . . 95
7.3.3 QoS Actor Finite State Machine . . . 96
7.3.4 SLA Actor Finite State Machine . . . 97
7.3.5 SLA Manager Actor Finite State Machine . . . 98
7.3.6 Customer Actor Finite State Machine . . . 98
7.3.7 Autonomous SLA Violation Monitoring and Filtering . . . 99
7.4 Modeling of Negotiation Scenario . . . 101
7.5 Modeling of Service Monitoring and Fault Tolerance Scenario . . . 102
7.5.1 Violation Handling in QoS Module . . . 102
7.5.2 Violation Handling in SLA Module . . . 103
7.5.3 Violation Handling in SLA Manager Module . . . 104
7.5.4 Violation Handling in Customer Module . . . 105
7.6 Discussion . . . 106

8 Conclusions and Future Directions **107**
8.1 Summary 107
8.2 Outlook 108

Bibliography **119**

Acronyms **119**

List of Tables **123**

List of Figures **124**

IV Appendices **126**

Appendix A SLA Template XML Example **127**
A.1 SLA Template Party 127
A.2 SLA Template Variable Declaration 128
A.3 SLA Template Agreement Declaration 128

Appendix B UML Diagrams **130**
B.1 UML Diagram of SLA (Template) Model 130
B.2 UML Diagram of BDD Implementation 131
B.3 UML Diagram of Infrastructure Planning and Optimization Component Implementation 132
B.4 UML Diagram of Advanced Reservation Implementation 133

Part I

Introduction

Chapter 1

Introduction

1.1 Background to the Research

1.1.1 Utility Computing

Utility computing is rapidly moving to a paradigm, in which it is provided as services that are delivered in a manner similar to traditional utilities such as water, electricity and gas [1]. In such a paradigm, users customize and access their services in the form of pay-as-you-go charging mechanism without necessarily having to concern themselves about where the services are hosted and how they are delivered. The unique advantages of this paradigm compared to in-house services have engendered a tremendous escalation in customer interest and entrepreneurs' investments. Generally, several computing architectures have evolved to implement this utility computing vision, including grid computing, Service Oriented Architecture (SOA) and cloud computing [2].

1.1.2 Cloud Computing

Especially in recent years, cloud computing has attracted substantial attention in many realms of industry and academia, for instance, including business, government, health care, intelligent transportation networks, life sciences amongst others [3]. In general, according to the introductory definition of cloud computing provided by the National Institute of Standards and Technology (NIST) [4] is as follows:

"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."

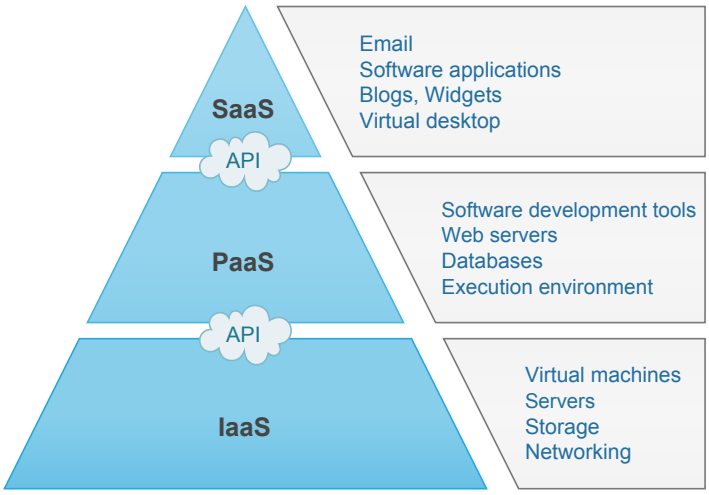


Figure 1.1: The cloud computing service categories and their characteristics

In terms of service type, three service delivery models are commonly employed in clouds, namely, Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS) and Infrastructure-as-a-Service (IaaS) [4]. As illustrated in Figure 1.1, IaaS involves outsourcing the equipments used to support consumers, including storage, servers, networking components and other fundamental computing resources on which the consumers are able to deploy and run arbitrary software. On top of IaaS, PaaS is a paradigm for delivering operating systems and associated services (e.g., acquired applications created using programming languages, libraries, tools, etc.) over the Internet onto the cloud infrastructure on demand - without downloads or manual installation. In addition, SaaS is a software distribution model, in which end-user applications are hosted by vendors or service providers over the Internet instead of executing on individual hardware. The applications are accessible from various client devices through either a thin client interface, such as a web browser (e.g., web-based email), or a program interface. In many deployments, these three approaches are closely interconnected with each other. They collaborated via their corresponding Application Program Interfaces (APIs) and this topology is known as SaaS, PaaS, IaaS (SPI) mode.

More recently, the adoption of cloud computing has shifted squarely to mainstream enterprises. This trend will continuously accelerate as more vendors escalate their focus and investments on cloud, and a tripling in the number of developers driving 10 times growth in SaaS offerings over the next four years is expected [5]. In parallel, International Data Corporation (IDC) predicts a shift from pure IaaS toward IaaS with PaaS capabilities, including development, testing, staging and deployment [5].

The IDC also foresees that the cloud expenditures will surge by 25%, reaching over $100 billion in the coming years, which is already prominently promising looking at the growing number of cloud services and users. Many Information Technology (IT) companies have been shifting their productions globally from on-premise to cloud on-demand deployments. Amazon, for example, well known for its Amazon Web Services (AWS), is offering the Amazon Elastic Compute Cloud (EC2) and the Amazon Simple Storage Server (S3) at the IaaS level [6]. On the platform level, Google has its product App Engine [7]. Last but not least, in relation to SaaS, it is estimated that SAP partners alone around the world may earn $33.6 billion in revenue in the next five years from SAP cloud and managed services, such as Customer Relationship Management (CRM) and Enterprise Resource Planning (ERP) through SAP Hana Enterprise Cloud (HEC) [5] [8].

1.1.3 Service Level Agreement

Contemporary IT Service Management (ITSM) expects service levels to be managed alongside functional service properties. More recently, the rapid evolution of the cloud market is giving rise to the emergence of new services, new ways of service provisioning and new interaction and collaboration models. Across the entire SPI mode, clouds are typically delivered based with the promises of on-demand pricing, less IT overhead, Quality of Service (QoS) amongst others.

However, for the sake of turning the promises into realized benefits, services must be accompanied by accurate and unambiguous definitions regarding the conditions of their usage [9]. This naturally entails that arrangements are made in accordance with internal IT support providers and external suppliers in the form of Operational Level Agreements (OLAs) and Underpinning Contracts (UCs), respectively [10]. These conditions can be accomplished and facilitated by Service Level Agreements (SLAs), electronic contracts between the service consumer and service provider that governs the aforementioned relationships by clarifying the terms of engagement and the consensus in expectations and obligations for the participating entities.

SLAs are important components of cloud computing governance and represent measurable elements needed to assure an agreed upon QoS between a cloud service customer and a cloud service provider [11]. Generally, according to the introductory definition of cloud computing service level agreement provided by the the International Organization for Standardization (ISO) and the International Electrotechnical Commission (IEC) [11] is as follows:

"The cloud computing service level agreement (cloud SLA) is a service level agreement between a cloud service provider and a cloud service customer based on a taxonomy of cloud computing specific terms to set the quality of the cloud services delivered. It characterizes quality of the cloud services delivered in terms of: a set of measurable properties specific to cloud computing (business and technical), and a given set of cloud computing roles (cloud service customer and cloud service provider and related sub-roles)."

Besides describing the expectations by dictating the quality and the category of the service, SLAs are also increasingly considered by the providers as a key ingredient towards achieving competitive advantages. Once the customer and service provider establish an SLA, all the terms are maintained fixed until the end of the service lifetime. In addition, the service provider has to ensure that the service quality reaches certain acceptable levels [12] and comply with the customer's demands and circumstances that may be subject to variation over time [13].

Consequently, an SLA lifecycle management architecture has been emerged to address the aforementioned concerns [14].

1.1.4 Service Lifecycle Management

SLAs relate by their very nature to different phases of a service lifecycle. Therefore, it is of fundamental importance to agree on a common reference model for such a service lifecycle which details the various phases, their stakeholders and their expected outputs in a common way without compromising the general design space for SLA management architecture.

The overall lifecycle model has been influenced to a dramatic degree and is broadly in-line with the ITIL framework [15]. In general, the service lifecycle management considers six main phases, which are: design and development, service offering, service negotiation, service provisioning, service operations and service decommissioning.

1.1.5 SLA Lifecycle Management

SLAs also play a central role in the service lifecycle, because by capturing service expectations and entity responsibilities they drive both engineering decisions at conception level (during for example service design) and operational decisions (during service usage and delivery) [16].

Although there is no standard definition, in recent years, various projects, research activities and companies provide the foundation for the state-of-the-art in

SLA lifecycle management. For instance, Ron et al. define the SLA lifecycle in three phases [17]. Later on, more detailed classifications are proposed and outlined individually by SLA@SOI project [18], by Sun Microsystems [14] and by the TeleManagement Forum [19]. In general, SLA lifecycle management consists of three phases, namely creation, operation and removal phases (with corresponding color on top of the Figure 1.2), each of which can be further expanded to sub-phases with the same color. The SLA creation includes three sub-steps, i.e. discover service provider, SLA definition and SLA establishment. Once service providers are discovered, customers have to be aware of the detailed capacity of the service providers. Therefore, the service providers describe and define their services properly and deliver the definition of their services to the customers. Then, the customers further establish the agreement(s) with one or more service providers based on the service definition through a process of SLA negotiation. In the following sections, we elaborate on the different phases in details.

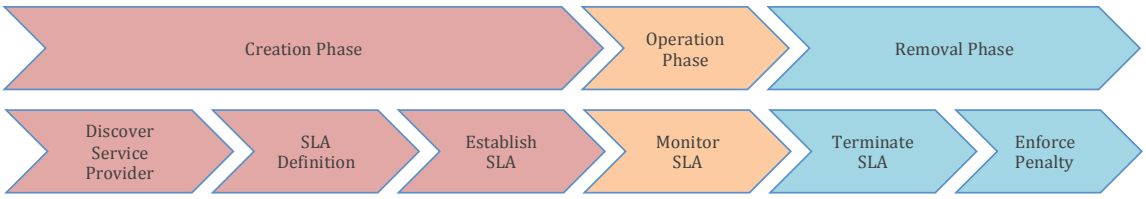


Figure 1.2: Evolution of SLA lifecycle management from three steps to six steps

1.1.5.1 Discover Service Provider

In this phase, the potential service providers are discovered and located according to customers' requirements. For instance, SLA@SOI project proposed an advertisement system by allowing the sharing of service information, i.e. SLA templates, in a seamless, interoperable and asynchronous manner between the service providers and customers. Whenever a change occurs, the system automatically propagates this update to all the subscribed parties [18].

1.1.5.2 SLA Definition

As a customer, it is necessary to identify the various elements, i.e. service terms, within an SLA, which include for instance, functional property such as configuration of resources, non-functional properties such as the bounds of guaranteed availability and performance, the parties, the cost, the data set for renegotiation, the penalty

terms for SLA violation amongst others. According to [20], those aspects should be included by an SLA template. In Chapter 3, we elaborate on an SLA template in IaaS scenario, which sets up a foundation of SLA documentation for the rest of the thesis.

1.1.5.3 Agreement Establishment

In this stage, measurement metrics and definition of each of these elements is established by a contract negotiation process between both parties. Although most of the works recognize SLA negotiation as a pivotal aspect of the SLA management [21] [22] [23] [24] [25], recent works only cast little insight on how negotiation (especially automated negotiation) can be realized. In addition, it is challenging to reflect the quality aspects of SLA components in a template [14].

1.1.5.4 SLA Monitoring

The service provider has to deliver the service that has been contractually approved during the SLA negotiation. Essentially, monitoring, measuring and recording the performance of the service in the SLA operation phase play a critical role in determining whether Service Level Objectives (SLOs) are achieved or violated.

The monitor service activity monitors the delivered service quality with respect to service levels as defined in the SLA between cloud service customer and cloud service provider [26]. It starts once an agreement has been established, in which the provider's delivery performance is measured and assessed. SLA violation implies un-fulfillment of the agreement. However, both parties want their objectives to be met. In order to avoid the violation, the service provider should be responsible for monitoring activities, adjusting the service accordingly and even triggering an SLA renegotiation process in some cases where, e.g., the QoS cannot be provisioned as promised in the template, or where the revenue policies alter from provider side arise. However, the more important the violated SLO, the more difficulties it involves in renegotiating the SLA, because no party wants to lose their competitive advantages in the market [14]. Therefore, a proper methodology for monitoring and adjustment in the SLA operation phase is of importance.

1.1.5.5 SLA Removal Phase

Once the SLA is successfully fulfilled, the SLA is naturally terminated and all the associated configuration information is removed from the service systems. On the other hand, a termination may also occur in case of an SLA violation, specifically, penalty

clauses are needed to be defined in order to enforce penalties for the SLA violation [14]. Therefore, the selection of these optimal models for the SLA violation penalty clauses enforcement is still an open problem in SLA decommission phase [27] [28] [29].

1.2 Motivation

A structured methodology engineered for the management of the different phases of an individual SLA during its lifespan is of importance. It facilitates the realization of the SLA management architecture and is typically designed to be generic and domain independent throughout the entire SPI mode. As such, the common challenges of the SLA lifecycle management motivate the work conducted for this thesis.

Motivation-1 - Regarding the phase of SLA creation, without standard documentation, SLAs could be verbose and service consumers would have to manually examine the details of the SLA that each potential provider offers. Such interaction between the consumer and service provider could be simplified by SLA automated negotiation. In other words, SLA automated negotiation can be used to accommodate a customer's diversified requirements against a service provider's capabilities and acceptable usage provisions. Over and above that, it determines the cardinality of parties involved, their roles, the visibility of the offers exchanged amongst others [25]. All this information is shared between the two parties via a representation of the SLA. During the entire SLA negotiation procedure, the SLA representation can be specified as a customer's requirement, service provider's offer or service provider's counter-offer. However, an SLA negotiation itself can be a time-consuming process, principally owing to the unpredictable number of rounds of negotiation as well as the complicated dependency relationships that may exist among SLAs. In some cases, the (counter-)offer that is provided by the service provider could be suboptimal and sometimes even be processed incorrectly. The suboptimal (counter-)offer structure further intensifies the complexity of the SLA negotiation, and the negotiation could be therefore delayed. Besides, the redundant elements within the suboptimal representation occupy extra system resource spaces. Accordingly, in this thesis, the aforementioned challenge is categorized as a structural suboptimal problem of SLA representation during SLA negotiation.

Motivation-2 - Having an available optimal SLA representation structure, the SLA management architecture is able to perform a corresponding analysis procedure for each potential offer regarding its feasibility. The feasibility in this thesis refers to two aspects, namely the feasibility of business policies and the feasibility of resource

availability. During SLA negotiation, the cloud service provider seeks to maximize profits and achieve business sustainability. These could be further translated into a multi-objective optimization problem of offering attractive and competitive pricing while allowing for an acceptable profit margin, whilst simultaneously building a reputation that will harvest longer contracts and returning customers, and avoiding penalties. For most cloud service providers, cloud resources are commonly finite. It might be the case that service providers must turn to external entities and competitors if they temporarily run out of resources, in order to avoid losing their customers. Furthermore, service providers need to offer enhanced resource management capabilities to potential returning customers who need resources at some point in the future, i.e. reservation of resources in advance. The above challenge is categorized as a utility architecture for business policies optimization and resource planning problem.

Motivation-3 - Moreover, once the IaaS service is deployed and in operation, service providers often consider Virtual Machine (VM) consolidation as another objective in order to trade-off energy consumption against service performance. An aggressive consolidation of VMs, however, may cause performance degradation when the service experiences increasing demand, resulting in unexpected rise of resource utilization. Some virtualization technologies enable the live migration of running VMs to achieve load balancing, fault-tolerance and hardware consolidation in data centers. However, in some cases the downtime / service unavailability caused by live migration may be substantial with relevance to the customers' expectations on responsiveness, as the latter are declared in established SLAs. Additionally, they may cause significant (potentially exponential) SLA violation penalties in the associated higher-level domains (i.e. PaaS and SaaS). Therefore, VM live migration should be managed in an attentive manner. The above challenge is categorized as a resource management for multi-domain SLA problem in this thesis.

Motivation-4 - Various approaches and applications are designed for reporting a historical records of outages or failures. The failures however have to be resolved manually. In any business involving computer-based work, creating an infrastructure with clear procedures and preventive maintenance can minimize the likelihood and impact of failures. In ITSM, such infrastructure is named as the escalation management that is responsible for defining and deriving action plans to return customers back to regular operations from system failures, triggering communication with internal and external stakeholders while at the same time managing customer communication and safeguarding customer's satisfaction [8]. Therefore, recovery from failures must

be planned. And an autonomous service violation self-heal system is of paramount importance.

Taking the above motivation into account, it is highly valuable to have an SLA-based cloud infrastructure, which aims to enable automatic service negotiation, provisioning, monitoring and optimization, all contributing to delivering significant competitive advantage. As such, potential customers will not be forced to accept static SLAs, or make their decisions on a take-it-or-leave-it basis.

1.3 Contribution

The thesis evaluates the state-of-the-art in regard to improvements in SLA lifecycle management with the following contributions.

To motivation-1 - The operation of negotiating SLAs can be facilitated when SLAs are translated into boolean function representations. We outline how to improve and simplify the structures that model SLAs, using optimization algorithms. We also evaluate the implementation of the SLA representation structural optimization model. The approaches are compared and distinguished by considering different metrics such as size of the structures, negotiation time amongst others. Finally, the entire model is applied to IaaS SLAs negotiation via simulation, through which we evaluate the model and prove that the automated negotiation process is improved both in space and time.

To motivation-2 - The objective of the approach is to minimize implementation and outsourcing costs for reasons of competitiveness, while respecting business policies for profit and risk. Besides, we assume SLA-based resource requests and introduce an advance reservation methodology during SLA negotiation. In the end, our simulations prove that the approach is feasible and works, yielding useful results given the scenario that we chose to implement.

To motivation-3 - We present our strategies for VM selection and allocation during live migration of VMs. We simulate a use case where IaaS and PaaS are combined. Then, we demonstrate that our proposal is efficient in managing trade-offs between the operational objectives of service providers (including financial considerations) and the customers' expected QoS requirements.

To motivation-4 - We apply an appropriate theoretical model for fine-grained, yet simplified and practical monitoring of massive sets of SLAs. Thus, the entire management of SLAs can be efficiently paralleled. Our approach separates the agreement's fault-tolerance concerns, i.e. resource scheduling, outsourcing and (re)-negotiation,

into multiple autonomous layers that can be hierarchically combined into an intuitive, parallelized, effective and efficient management structure. Finally, we demonstrate that this is a realistic approach for the automated management of the complete SLA lifecycle, including negotiation and provisioning, and reinforce the criticial importance of monitoring as the driver of contemporary scalability requirements.

Moreover, in this thesis, by extending the SLA (T) model from project SLA@SOI, we generate the IaaS scenario SLA Extensible Markup Language (XML) template and the evaluator of the SLA template. By transforming the verified SLA XML template into Java object, we then create the Java object parser and allocate the corresponding elements into each planning and optimization component.

1.4 Impact

The results of the dissertation have been peer-reviewed and published in two international journals, two books and four international conference proceedings.

The subsequent list presents the journal articles:

- Fault-tolerant Service Level Agreement Lifecycle Management in Clouds Using Actor System. Lu, K., Yahyapour, R., Wieder, P., Yaqub, E., Abdullah, M., Schloer, B., Kotsokalis, C. In In: Journal of Future Generation Computer Systems (FGCS), ISSN: 0167-739X, DOI: 10.1016/j.future.2015.03.016.

- QoS-Based Resource Allocation Framework For Multi-Domain SLA Management in Clouds. Lu, K., Yahyapour, R., Wieder, P., Kotsokalis, C., Yaqub, E., Jehangiri, A.I. In International Journal of Cloud Computing (IJCC), ISSN 2326-7550, Vol. 1, No. 1, July-September 2013, New York, USA.

In the following, we list the book chapters:

- SLA-Based Planning for Multi-Domain Infrastructure as a Service. Lu, K., Roeblitz, T., Chronz, P., Kotsokalis, C. In: book "Cloud Computing and Services Science", Springer Science+Business Media New York, 978-1-4614-2326-3, (2012).

- SLA-Enabled Infrastructure Management. Kennedy, J., Edmonds, A., Bayon, V., Cheevers, P., Lu, K., Stopar, M., Murn, D. In Book: Wieder. P., Butler, J., Yahyapour R. (Eds.) Service Level Agreements For Cloud Computing, Part 6, Springer-Verlag, 271-287 (2011).

- G-SLAM - The Anatomy of the Generic SLA Manager. Rojas, M., Chronz, P., Lu, K., Yaqub, E., Fuentes, B., Castro, A., Foster, H., Reuda, H., Chimeno, A. In Book: Wieder. P., Butler, J., Yahyapour R. (Eds.) Service Level Agreements For Cloud Computing, Part 4, Springer-Verlag, 167-186 (2011).

Then, we list the conference publications:

- QoS-Aware VM Placement in Multi-Domain Service Level Agreements Scenarios. Lu, K., Yahyapour, R., Wieder, P., Kotsokalis, C., Yaqub, E., Jehangiri, A.I. In IEEE 6th International Conference on Cloud Computing (IEEECloud, Santa Clara Marriott, CA, USA), IEEE Computer Society, (2013).

- Structural Optimization of Reduced Ordered Binary Decision Diagrams for SLA Negotiation in IaaS of Cloud Computing. Lu, K., Yahyapour, R., Yaqub, E., Kotsokalis, C. In: 10th International Conference on Service-Oriented Computing (ICSOC2012, Shanghai, China), Springer-Verlag, (2012).

- QoS-aware SLA-based Advanced Reservation of Infrastructure as a Service. Lu, K., Roeblitz, T., Yahyapour, R., Yaqub, E., Kotsokalis, C. In: Third IEEE International Conference on Cloud Computing Technology and Science (CloudCom2011, Athens, Greece), IEEE Computer Society, 288-295 (2011).

- SLA-Based Planning for Multi-Domain Infrastructure as a Service. Lu, K., Roeblitz, T., Chronz, P., Kotsokalis, C. In: International Conference on Cloud Computing and Services Science, Springer-Verlag, 343-351 (2011).

Furthermore, the author has identified the topics and supervised two Master theses with relation to the overall topic of this dissertation:

- Dagang Chen: Combination of SLA@SOI Infrastructure Service Level Agreement Manager with Open Cloud Computing Interface. Master Thesis. Summer semester 2011. Technical University of Dortmund.

- Shihong Li: SLA-based Planning and Optimization in IaaS. Master Thesis. Winter semester 2010/2011. Technical University of Dortmund.

1.5 Dissertation Organization

The remainder of the thesis is structured as followings. In Chapter 2, we summarize the requirements by analyzing the common challenges of SLA lifecycle management. We therefore outline a number of scenarios to explain the problems in detail so as to derive the problem definitions for the thesis. In Chapter 3, we elaborate on an SLA modeling and templating in IaaS scenario. In Chapter 4, we outline how to improve the SLA representations using optimization algorithms during SLA negotiation. Chapter 5 discusses the business utility modeling, resource outsourcing and advance reservation for infrastructure services that are subject to SLAs. In Chapter 6, we present the implementation of our generic SLA manager, alongside its strategies for VM selection and allocation during live migration of VMs. In Chapter 7, we demonstrate our approach that separates the agreement's fault-tolerance concerns into multiple autonomous layers. Finally, in Chapter 8, we conclude the thesis and discuss some possible directions for future work.

Part II

Requirements

Chapter 2

Requirements and Problem Statements

Compared to traditional IT systems, where service organization has control over the entire stack of computing resources, in cloud IT system, the service providers and the customers collaboratively design, establish, deploy and operate the cloud resources during the whole lifecycle of the services. In this chapter, we target to evolve the requirements by analyzing both the state-of-the-art and common challenges of the cloud SLA lifecycle management. We therefore outline a couple of scenarios to derive the problem definitions of the thesis and explain the problems in details.

2.1 Requirements

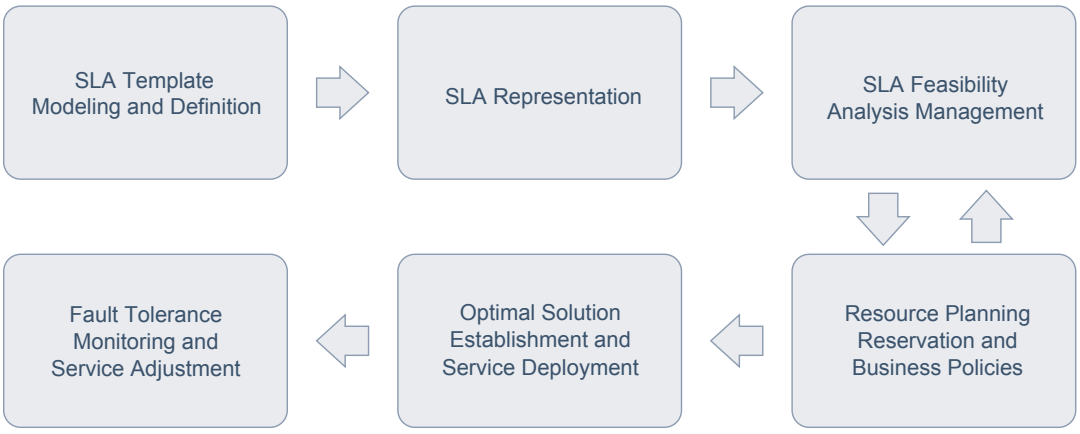


Figure 2.1: Requirement diagram

Based on the common challenges by ISO and IEC in different phases of cloud SLA lifecycle [11, 26], we address the specific problems with detailed requirements as

follows (Figure 2.1):

- **SLA creation**: an SLA definition and an SLA representation are needed to perceive the requirements from cloud customers unambiguously and process them timely. Therefore, we need to reference and extend an SLA model in order to address the domain specific SLA terms and define the SLA. This problem is derived more in details in Section 2.2 and Section 2.3.

- **SLA establishment**: we also need an overall business plan that manages the business relationship with customers. The business plan should perform the SLA representation management, requirements feasibility management and resource planning optimization strategies. Besides, the management of the services of the peer service providers should also be considered during resource outsourcing. This problem is addressed more in details in Section 2.4.

- **SLA monitoring and removing**: the SLA violation monitoring strategy, service adjustment, capability management, fault-tolerance characteristics and actions, are needed to be addressed. Finally, penalty modeling and analysis is also of importance. This problem is discussed more in details in Section 2.5 and Section 2.6.

2.2 Derivation and Selection of SLA Model

The first phase of the SLA lifecycle deals with formulating the service offer in terms of an SLA template. This is further applied to advertise the cloud service to potential customers, in which all the necessary details about the forthcoming electronic contract are described.

According to the definition given in the introduction section, an SLA must contain all information necessary for service consumers and providers to achieve a common understanding of services, priorities, responsibilities, obligations, penalties et cetera. However, a number of different SLA specifications [30, 31, 32, 33, 34, 35], being used within the distributed computing landscape, confirm that an SLA specification should be modeled mainly based on its domain. Since the content of an SLA is highly domain-dependent, a uniform definition is not desired. Therefore, a proper approach for modeling and defining the SLAs should be derived by considering the specific domain of the thesis and the detailed configuration of the services.

2.3 Structural Suboptimal Problem of SLA in Negotiation

An SLA is in essence a set of facts and rules. The facts are globally (with respect to the contract) applicable truths, such as the parties involved, monetary unit, etc [36]. The rules include:

- The conditions that must hold for a certain clause to be in effect.
- The clause itself, typically describing the outcome that the customer expects to harvest, is typically referred to as SLO.
- A fall-back clause in the case that the above clause is not fulfilled. For instance, a compensation or penalty from service provider.

As an example, for the condition "time between 7 a.m. and 6 p.m. on working day", the clause could be "service availability $\geq 99.9\%$", and the fall-back clause could be "a penalty of 10% of original service price". This kind of format actually reflects real-life contracts and their "if-then-else" structures, which perform different actions depending on whether the specified boolean condition is evaluated to be "true" or "false". In this case, the SLA can therefore be presented as a boolean function with standard built-in terms, such as logical operators, comparison operators, arithmetic functions, time-series, temporal context and so forth [34]. The result of such boolean function is either "true" or "false", where "true" means that the SLA request is feasible and "false" implies that the request is not practical.

Several applications can represent boolean functions as rooted, directed and acyclic graphs, which can be further exploited as a canonical representation of SLAs, allowing their efficient and unambiguous management independent of their structure's specifics and supporting decision-making during the SLA negotiation [33, 36]. Nevertheless, the SLA negotiation can be a time-consuming process, mainly owing to the unpredictable number of rounds of negotiation and the possible complicated dependencies among SLAs. For instance, a SaaS provider negotiates with one or more IaaS providers for computing resources to host the proposed applications. First of all, establishing an SLA needs one or more rounds of negotiation until both sides agree with the stipulation of the contract, so the negotiation time is unpredictable. Second, a SaaS SLA could have dependencies with one or more IaaS SLAs. Thus,

- On one hand, the SLA representations are maintained throughout the whole lifecycle of SLAs, so the suboptimal SLA representations with a large number of redundant elements occupy extra memory resource.
- On the other hand, the SLA management system has to parse the SLA request into a list and analyze each of its options. An SLA representation may have semantical redundancy that improperly reflect a customer's requirements. As a result, this intensifies the complexity and further reduces the time efficiency of the SLA negotiation process.

Accordingly, in this thesis, the aforementioned challenge is categorized as structural suboptimal problem of the SLA representation during negotiation.

2.4 Business Policies and Resource Planning in SLA Negotiation

In clouds, an overall business plan that manages the business relationship between the cloud providers and customers is of importance [26]. An IaaS service provider, like any service provider, seeks the optimal Return on Investment (ROI) and achieving business sustainability. These could be translated to attractive and competitive pricing while allowing for an acceptable profit margin, but also avoiding penalties and building a reputation that leads to longer contracts and returning customers.

Meanwhile, the resources are finite. It might be the case that the IaaS service providers must extend their capacities by outsourcing if they run out of resources (Figure 2.2), so that they do not lose the customers even if that may compress their profit margins for the specific contract. Other reasons why this is possible are requests for unsupported resource types, or unsupported storage locations etc. Independent of the reason, a service provider may become a customer itself to another provider(s). As suggested by ISO, the management of the services of the peer service providers should be considered carefully during resource outsourcing [26]. This subcontracting requirement is further underlined in a scenario where autonomous agents negotiate for resources on demand, in a fully dynamic environment without any human intervention.

We specifically consider the IaaS providers who own only limited resource capacity. Therefore, they need to offer enhanced resource management capabilities to customers that need resources at some point in the future. Without appropriate mechanisms there are only two unpreferred options: (1) overprovisioning incurring

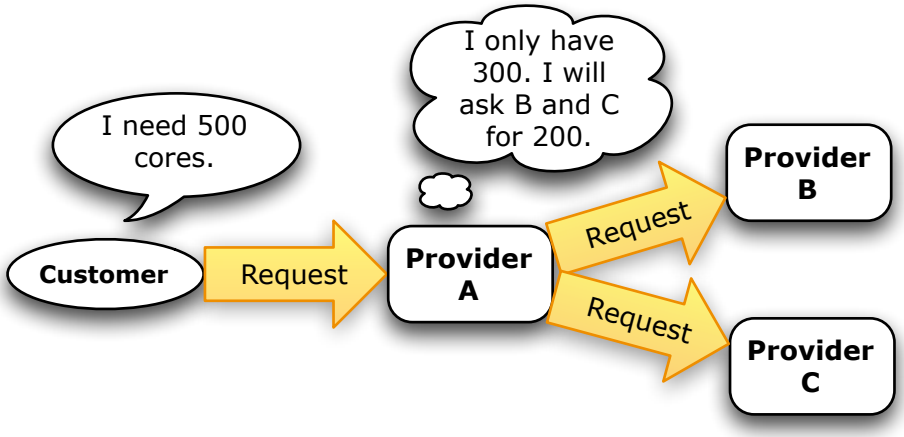


Figure 2.2: Multi-domain resource provisioning

additional cost to the provider, and (2) early resource allocation prior to their actual use incurring additional cost to the customer. For some cloud providers, e.g., Amazon EC2 [6], Google App Engines [7], resources can be provisioned in the form of on-demand mode. However, for Small and Medium-sized Enterprises (SME) with restricted resources, not all the requests raised from the customers can be entertained when multiple-requests surge at the same time. Thus, it becomes necessary to establish a mechanism for efficiently arranging, managing and monitoring the partitioned virtual infrastructures.

The problem that we want to solve is how to plan for internal resources and outsourcing (subcontracts for external resources) as well as efficiently represent advance reservations within the IaaS scenario in such a manner that:

- Service provider's profitability requirements are satisfied. The financial risk is maintained low. Service provider's reputation remains as good as possible. The price quotes provided to customers are minimized for reasons of competitiveness.

- The service provider has to consider the feasibility of this request based on the resource availability during the requested time interval.

- If the service provider has enough resources, the request can be satisfied at a corresponding price. On the contrary, if the request cannot be entertained as such, an alternative solution decorated as a counter-offer should be provided to the customer. This forms a business strategy that attempts to satisfy a customer by intelligently offering flexible alternative solutions rather than a refusal.

- Partitioning of the computing resources inevitably leads to fragments. An important question is: how to eliminate the fragments in the virtual pool so as to improve the resource utilization?

2.5 Resource Management for Multi-Domain Cloud in SLA Operation

In IaaS, through virtualization technologies, physical resources of data centers can be partitioned into flexible and scalable virtual computing units, namely VMs. However, large-scale data centers introduce large power-consumption costs. Thus, a technique that dynamically reconfigures the IT infrastructure to reduce the total power consumption becomes necessary. As such, VM consolidation emerges to execute the VMs on as few servers as possible, in order to concentrate the workloads and limit the number of physical servers powered on [38].

VM consolidation is usually treated as an objective of the service provider. From the customer's perspective, an automated SLA negotiation may be used to accommodate heterogeneous requirements against a service provider's capabilities and acceptable usage terms.

In order to commit to the requested QoS terms (e.g., service performance and availability), service providers have to assess their own resource management strategies so as to balance the achievable profit based on a guaranteed delivery of service(s) and avoid penalties in case the agreement is violated. An aggressive consolidation of VMs, however, may lead to performance degradation when the service faces increasing demand, resulting in unexpected rise of resource utilization.

By means of VM live migration technologies, both VM consolidation and service performance can be coordinated. However, short downtimes of services are unavoidable due to the overheads of moving the running VMs. Hence, the respective service interruptions in IaaS reduce the overall service availability and it is possible that the customer's expectations on responsiveness are not met [39].

Moreover, it might bring exponential service violation penalties to its associated domains (e.g., SaaS and PaaS). For example, the solutions in [38] provide availability from 99.7% to 99.93%. For e-commerce and other industrial use cases, a service availability value below 99.9% is usually considered unacceptable [40]. Therefore, in order to provide high service availability, to avoid service violation and the consequent penalties, the number of VM live migrations should be monitored and controlled.

2.6 Fault Tolerance Management in SLA Operation

Automated SLAs have been proposed for cloud services, as contracts used to record the rights and obligations of the two parties. Automation refers to the electronic formalized representation of SLAs and the management of their lifecycle by autonomous agents. Whereas the application of SLAs to cloud computing is directly implied by the very nature of it as a method to outsource operational responsibility, their automated management is becoming increasingly relevant for reasons of scale.

Within the phase of SLA operation, it is sometimes even the customer's responsibility to monitor and demonstrate any SLA violation [6], such as CloudWatch [41] from Amazon, Nagios [42] and OpenTSDB [43], through which developers or system administrators can use them to collect and track historical data of many system metrics, gain insight and react immediately to keep their applications and businesses running smoothly.

However, all the above approaches are only designed for reporting a historical records of outages or failures and the failures have to be manually resolved. In ITSM, one level is frequently unable to resolve the incident in the first instance and so has to turn to a specialist or superior who can make decisions that are beyond the service desk's area of responsibility. This process is referred to as escalation [10].

The escalation management plays a pivotal role in cloud service maintenance for managing the high-priority issues that concern significant impacting customers. It is responsible for defining and deriving action plans to return customers back to regular operations, triggering communication with internal and external stakeholders while at the same time managing customer communication and safeguarding customer's satisfaction [8]. Ultimately, in any business involving computer-based work, creating an infrastructure with clear procedures and preventive maintenance can minimize the likelihood and impact of failures.

The system should understand the diversified versions of possible failures and properly respond to the potential violation alarm. Therefore, recovery from failures must be planned. And an autonomous service violation self-heal system is of paramount importance [44].

2.7 Discussion

In this chapter, we summarize the requirements enlightened by the common challenges of cloud SLA lifecycle management. We therefore explain a couple of scenarios and the requirements in great detail to lay the foundations for the concept outlined in the following chapters. In next chapter, we introduce the derivation and selection of SLA model as a starting point of the SLA lifecycle. By using this model, the SLA management could set up a specific SLA representation and further analyze the requirements of the customer within the SLA.

Part III

Models, Implementation and Evaluation

Chapter 3

SLA and SLA Template Modeling During SLA Definition

3.1 Derivation and Selection of SLA Model Specification

A number of SLA related projects and implementations exist, such as IANOS (CoreGRID) [30], SmartLM [31], SLA@D-Grid [32], IBM Web Service Level Agreement (WSLA) [33], SLA model in SLA@SOI [34] amongst others. Most of them integrate Web Service Agreement (WS-Agreement) into their architectures. WS-Agreement is a web services protocol for establishing agreement between two parties, i.e. a service provider and a consumer, using an extensible XML language for specifying the nature of the agreement and agreement templates to facilitate discovery of compatible agreement parties [35]. Especially, the SLA (Template) model (acronymed as SLA (T) model) from SLA@SOI project is also derived to a large degree from the WS-Agreement specification, with some key extensions. We choose the SLA (T) model as the SLA specification in this thesis for the following reasons:

- WS-Agreement is concerned specifically with web services, while the SLA (T) model is supposed to also handle human-based services (as involved in use cases line) and focuses on gathering the general requirements, modeling the concepts and abstracting the necessary and relevant distinctions. Specifically, the primary objectives in developing the abstract SLA (T) syntax are to support the varying requirements of use case line and to ensure that SLA (T) information content is sufficiently well specified to support scientific innovations developing line. Hence, the present conception of "service" is generic, and not confined by specific requirements for instantiation.

- WS-Agreement is a specification of XML syntax for representing SLAs, and has only informal and restrictive semantics. The SLA (T) model, instead, is intended to be independent of syntactic constraints that can be applied to formally describe SLAs and their respective templates in a machine-readable and language-independent manner.
- Semantic checking, validator and parsing suits in the SLA (T) model enable the SLA to be described precisely without syntactic and semantic errors.
- The abstract SLA (T) syntax can be easily and seamlessly transformed into other machine-readable representations, such as Java API, Binary Decision Diagram (BDD), XML Schema and Backus Normal Form (BNF) Grammar, which could be further adapted by monitoring and fault tolerance components. The relations between the different modeling levels are illustrated in Figure 3.1.

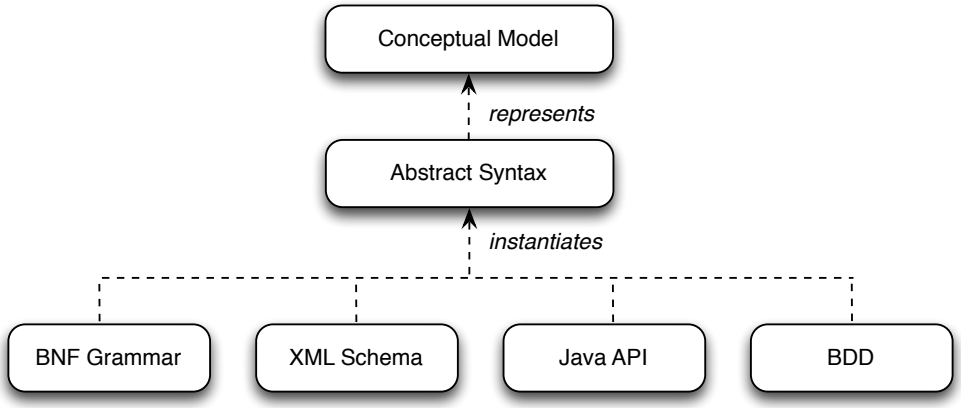


Figure 3.1: SLA model high-level overview

3.2 Syntax of SLA and SLA Template

The above objectives exhibit two broad constraints, which can be accommodated by the abstract SLA (T) syntax. On one hand, we need a model for SLA (T) that is sufficiently specific and detailed to support those aspects of SLA (T) management, which are common to all domains in the project SLA@SOI, namely business domain, platform domain and infrastructure domain. While on the other hand, the strong semantic requirements imply high-specificity and customization.

Generally, based on the abstract SLA (T) syntax, we derive the SLA template in IaaS scenario, which separates the content of an SLA template into 5 content types (Figure B.1 in Appendix B.1):

- SLA template attributes, e.g., Universally Unique Identifier (UUID) for the template (Appendix A.1).
- Details of the agreement parties. Parties in a cloud computing system are its stakeholders [26]. e.g., Identifier (ID) role of each party (Appendix A.1).
- Service description that is encapsulated as interface declaration.
- Variable declaration that serves as variable reference within a term in the agreement, it is provided as a syntactic convenience to improve readability and avoid redundancy in the content.
- Terms of agreement specify the detailed functional properties as well as QoS guarantees (Appendix A.3).

As illustrated in Figure B.1 in Appendix B.1, an SLA is simply an SLA template with an extended set of attributes specifying the time (when the SLA is agreed), the duration of the agreement (the time period during which it is in effect) and a reference of the SLA template on which the SLA is based. Specifically, it is also possible to adopt an extensible modular approach to develop a core set of modules, referred to as "vocabularies", which encapsulate the information such as the guarantees on the functional and qualitative behaviour of services, the domain specific QoS metrics and constraints on service delivery. Therefore, the individual use case is responsible for customizing additional vocabularies as extensions to the core SLA (T) model [34].

3.3 Discussion

In this chapter, we outline the SLA model that is referenced by the thesis, thus it is possible to address and define the domain specific SLA terms. We further explain the syntax of the SLA (T) in details.

Having a proper SLA definition, the SLA management should transform it to a specific SLA representation in order to further analyze the requirements of the customer within the SLA. In next chapter, we introduce an SLA representation and propose its corresponding structural optimization strategies.

Chapter 4

Structural Optimization of SLA During Establishment

4.1 Scenario Introduction

Typically, in SLA creation phase, service provider abstracts the services in perspicuous terms, e.g., the instances of Amazon EC2 [6]. As such, non-technical customers can focus on the services jointly rather than concerning them much in detail. Nevertheless, the service provider would best allow customers with flexibility and adaptability [10]. Chandra et al. [45] suggest that fine-grained temporal and spatial resource allocation may lead to substantial improvements in capacity utilization.

In IaaS scenario, the proposition above indicates that the service consumers are free to customize the VM configuration of an SLA. Particularly, the granularity of customizing VMs evolves from setting the number of predefined VM to the detailed qualitative characteristic of each term within a VM. The qualitative characteristics supported in SLAs is a topic heavily researched in the last decade (e.g., [46, 47]). In this thesis, instead of defining the exact qualitative characteristics that are requested by customers, we assume that there is a list of such supported characteristics, which is published by the cloud provider within SLA templates. SLA templates are customizable documents that serve as a basis for bootstrapping SLA negotiation. The customizable SLA terms for VM are listed in Table 4.1, where each term consists of one or more options and each option is issued with one variable. The VMs are compute instances, connected with Operating System (OS) image and network instances. The storage pool is formed by several images, which means that customers are able to upload their own images. Besides, service availability is one of the most important QoS metrics with respect to cloud service. In [48], authors outline that the service availability guaranteed by three large cloud providers (i.e. Amazon, Google

and Rackspace Cloud) is more than 99.9% in order to obtain good reputation in today's competitive market. Therefore, we propose to provide a basic service availability of 99.9% and an advanced availability of 99.99%. The last one implies that the service provider must pay special attention (e.g., extra resources backup, monitoring strategy etc.) onto the service during SLA monitoring phase in order to avoid SLA violation. Eventually, a provisioning request (in the form of an SLA establishment request by the customer, or a request for a price quote) will include a list of quantities for resources with specific characteristics and specific qualities.

Table 4.1: SLA terms and descriptions

SLA term	**Option**	**Variable**
Service_name	[service name]	[x_1]
Business_hours	[09:00-17:00]	[x_2]
VM_number	[1]	[x_3]
CPU_core	[1, 2, 4]	[x_4, x_5, x_6]
CPU_speed	[2.4 GHz, 3.0 GHz]	[x_7, x_8]
Memory	[1 GB, 2 GB]	[x_9, x_{10}]
Network	[Net-1, Net-2]	[x_{11}, x_{12}]
Storage_image	[Private, OS-1, OS-2]	[x_{13}, x_{14}, x_{15}]
Service_availability	[99.99%, 99.9%]	[x_{16}, x_{17}]

In Section 1.2 and 2.3, we derive the structural suboptimal problem for SLA Reduced Ordered Binary Decision Diagram (ROBDD) representation during SLA creation phase. In the rest of this chapter, our contributions are stepwise described by:

- explaining the modeling of SLA with ROBDD;
- specifying the approaches for simplifying and controlling of the ROBDD structure in order to improve the efficiency of SLA negotiation;
- experimentally approving the strengths and weaknesses of different approaches with an IaaS use case based on the cloud infrastructure provided by the GWDG for its customers and scientific communities.

4.2 Modeling SLA with ROBDD

This section serves as a general, high-level introduction to BDDs and their basic properties.

BDDs, based on Shannon's decomposition theorem [49], are well-known in the domain of Computer Aided Design (CAD) for Very Large Scale Integrated (VLSI) circuits. They can represent boolean functions as rooted, directed and acyclic graphs, which consist of serveral decision nodes (denoted as circles) and two terminal nodes (denoted as squares) called 0-terminal and 1-terminal. Each decision node is labeled by a boolean variable, which has two child nodes called low child (connected with hashed line) and high child (connected with solid line). A path from the root node to the 1-terminal represents a variable assignment for which the boolean function is true. Such a path is also called "1-path".

A BDD, with all variables occurring in the same order on all paths from the root, is said to be Ordered Binary Decision Diagram (OBDD). Furthermore, if all identical nodes are shared and all syntactically redundant paths are eliminated, the BDD is said to be reduced, abbreviated to ROBDD [50].

$$f(x_1, x_2, x_3) = x_1 \vee x_2 \vee x_3 \tag{4.1}$$

For example, Equation 4.1 is a disjunction boolean function with 3 variables. Its corresponding ROBDD is illustrated in Figure 4.1 (a), including 3 decision nodes (x_1, x_2, x_3) and 3 1-paths (x_1 = true, x_2 = false, x_3 = false), (x_1 = false, x_2 = true, x_3 = false) and (x_1 = false, x_2 = false, x_3 = true).

An application of SLA ROBDD has been proposed in [36]. This application can represent boolean functions as rooted, directed and acyclic graphs, which can be further exploited as a canonical representation of SLAs, allowing their efficient and unambiguous management independent of their structure's specifics.

In an SLA ROBDD structure, a 1-path is specified as a potential SLA proposal or offer. Compared to other techniques to represent boolean functions, e.g., truth tables or Karnaugh maps, ROBDDs often require less memory and offer faster algorithms for their manipulation [51]. Compared to other representations of SLA, e.g., WSLA [33] and the SLA model in [34] that focus on enabling interoperability between independent agents, ROBDDs focus on a system-internal representation that is capable of efficient supporting decision-making during the SLA negotiation. As such, ROBDDs are ideal for modeling SLAs due to their capability to provide canonical representations generated on the grounds of "if-then-else" rules. Especially, ROBDDs

can express SLAs unambiguously. Equivalent SLAs, which are structurally different, are eventually represented by the same ROBDD. Hence, ROBDDs can be used internally in systems that facilitate the process of negotiating SLAs, subcontracting parts of them, optimizing their utility and managing them during the runtime for monitoring.

4.3 ROBDD Structural Optimization

A ROBDD, however, may not be optimally structured upon production. A suboptimal ROBDD structure increases the complexity of the SLA negotiation. In the following subsections, the reduction of 1-paths via the application of Term Rewriting Systems (TRSs) with mutually exclusive features is demonstrated. Apart from that, we outline two approaches to reduce the number of paths and nodes towards ROBDD structures by using ROBDD optimization algorithms.

4.3.1 Term Rewriting System with Mutual Exclusiveness in ROBDD

According to the canonicity Lemma in [50], by reducing all the syntactically redundant paths, there is exactly one ROBDD with n basic variables in the order of $x_1 < x_1 < ... < x_n$. In this unique ROBDD, all variables of a disjunction function are mutually exclusive. Based on the truth table, e.g., a disjunction function is true when all its variables are true. However, such an assignment does not exist in ROBDD; as the ROBDD checks these variables one after another, the first true variable already ensures the value of this function to be true and that leaves the rest of the variables unevaluated. For instance, as we explained in Section 4.2, the ROBDD (Figure 4.1 (a)) of Equation 4.1 has 3 decision nodes and 3 1-paths and its structure cannot be simplified anymore. Therefore, if the inputs of a disjunction function are the basic variables, they are mutually exclusive and the ROBDD contains no redundant decision nodes and 1-paths. In contrast, when the inputs are not basic variables, despite the mutually exclusive feature among the inputs (only one of the inputs will be selected), the ROBDD of this disjunction function might have redundant decision nodes and 1-paths. *Semantically, options of most SLA terms are mutual exclusive, which means the customer can only choose one of them and this term must be represented as a combination of all the options with binary operators "logical conjunction" ($\wedge$), "logical disjunction" ($\vee$) and unary "negation" ($\neg$) explicitly.* While constructing a ROBDD, using Table 4.1, a simple ROBDD (see Equation 4.2) can be customized by specifying

SLA term "Network" with 2 mutually exclusive options "x_{11}" and "x_{12}" and ROBDD term "Service_availability" with 2 mutually exclusive options "x_{16}" and "x_{17}". This SLA indicates that inputs "$x_{11} \wedge x_{16}$" and "$x_{12} \wedge x_{17}$" both are acceptable for the customer.

$$SLA = (x_{11} \wedge x_{16}) \vee (x_{12} \wedge x_{17}) \tag{4.2}$$

Its corresponding ROBDD (see Figure 4.1 (b)) includes 6 decision nodes and 4 1-paths. However, two of those paths, namely (x_{11} = true, x_{12} = true, x_{16} = false, x_{17} = true) and (x_{11} = true, x_{12} = true, x_{16} = true), are not correct, since x_{11} and x_{12} cannot be true concurrently. In IaaS, selecting both "Net-1" and "Net-2" as network configuration is an illogical clause. Therefore, this inaccurate SLA representation creates two unrealistic semantically redundant 1-paths and such 1-paths should be eliminated at the very beginning of the SLA negotiation.

In order to optimize the structure of the ROBDD, TRS for boolean functions [52] [53] is studied, which could be applied in SLA terms selection so as to reduce the number of redundant 1-paths. In mathematics, rewriting systems address a wide range of methods of transforming terms of a formula with other terms. In TRS, a term can be recursively defined to a constant c, or a set of variables $x_1 ... \ x_n$, or a function f [53]. The terms are composed of binary operators logical conjunction "$\wedge$", logical disjunction "$\vee$" and unary operator "$\neg$". In IaaS, an SLA term contains one or more variables, namely the options. We make an effort to rewrite the set of SLA terms while customizing SLA templates in a way that depicts the customer's request precisely. Consequently, we propose that specifying an SLA term (t) with 2 options (α_1 and α_2) can be rewritten as illustrated in Equation 4.3.

$$t \rightarrow (\alpha_1 \wedge \neg\alpha_2) \vee (\neg\alpha_1 \wedge \alpha_2) \rightarrow \alpha_1 \oplus \alpha_2 \tag{4.3}$$

Mutual exclusiveness cannot be simply represented as a combination of all the options with the exclusive disjunction "$\oplus$", when there are more than 2 options in an SLA term. The reason behind is that when the number of "true" variables is odd then the output is true. And the output is false when the number of "true" variables is even [54]. For example, all-true assignment makes the expression $\alpha_1 \oplus \alpha_2 \oplus \alpha_3$ true, which is however not what we expect. Thus, when an SLA term (t) contains n ($n \geq 3$) options (α_1,..., α_n), we have the following assumptions:

$$N = \{1, ..., n\} \tag{4.4}$$

$$A \cup B = N \tag{4.5}$$

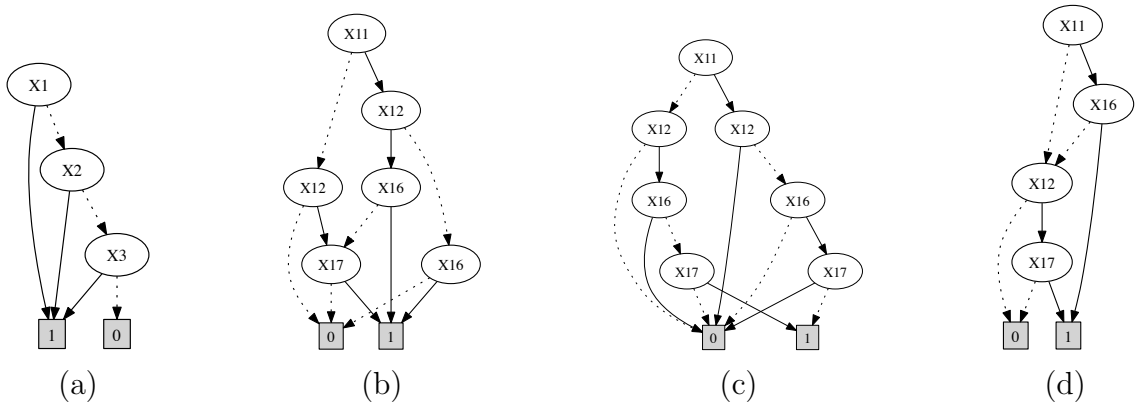


Figure 4.1: (a) The ROBDD of disjunction function, (b) Mutual exclusiveness of disjunction function but with semantically redundant 1-path in ROBDD, (c) Mutual exclusiveness of disjunction function without semantically redundant 1-path in TRS ROBDD, (d) ROBDD after path optimization

$$A \cap B = \emptyset \tag{4.6}$$

$$A \neq \emptyset, B \neq N \tag{4.7}$$

From Equation 4.4 to 4.6, N is a set of sequential numbers of all options. N can be further divided into 2 disjoint subsets. Options that concern a customer are put into set A and indifferent ones are in set B. Equation 4.7 implies that the customer should select at least one option. Thereby, specifying an SLA term (t) with n ($n \geq 3$) options can be rewritten as Equation 4.8.

$$t \rightarrow \bigvee_{l \in A} \left(\left(\bigwedge_{k \in A} \neg \alpha_k \right) \wedge \alpha_l \right) \wedge \left(\bigwedge_{m \in B} \neg \alpha_m \right), l \neq k \neq m \tag{4.8}$$

For SLA term t, $\bigcup_{l \in A} a_l$ is a set of options that the customer is flexible with and the rest of the options in A ($\bigcup_{k \in A} a_k$) should be denied *explicitly* with "¬". In the meanwhile, the unconcerned options in N, namely set B, should also be negated with "¬" in order not to conflict with other SLA option(s) from the same customer. Therefore, when $n = 3$, Equation 4.8 can be specified as Equation 4.9, in this case the set B is empty, which means the customer is flexible with any option of α_1, α_2, α_3.

$$(\alpha_1 \wedge \neg\alpha_2 \wedge \neg\alpha_3) \vee (\neg\alpha_1 \wedge \alpha_2 \wedge \neg\alpha_3) \vee (\neg\alpha_1 \wedge \neg\alpha_2 \wedge \alpha_3) \tag{4.9}$$

Based on above concepts, Equation 4.2 can be written as Equation 4.10 with 7 decision nodes and 2 1-paths in its TRS ROBDD (see Figure 4.1 (c)). The number of redundant 1-paths is reduced efficiently although the node size increases by 1.

$$(x_{11} \wedge x_{16} \wedge \neg x_{12} \wedge \neg x_{17}) \vee (x_{12} \wedge x_{17} \wedge \neg x_{11} \wedge \neg x_{16}) \tag{4.10}$$

In summary, the aforementioned term rewriting concepts can be set into a dictionary and updated dynamically according to the use case, whereby the semantically redundant 1-paths can be eliminated efficiently. This also reduces the complexity of planning and optimization processes for SLA management architecture. However, the shortcoming is that this approach might introduce extra decision nodes.

4.3.2 ROBDD Variable Swap and Sifting Algorithm

Alternatively, ROBDD structural optimization algorithms [50] [51] [55] [56] enlighten us with the theoretical foundation and can be applied to optimize the size of ROBDD. The algorithms make use of basic techniques such as variable swaps and the sifting algorithm. As approved in [51], *a swap of adjacent variables in a BDD only affects the graph structure of the two levels involved in the swap, leaving the semantic meaning of boolean function unchanged.* Based on the variable swap, the classical sifting

Algorithm 1 Sifting algorithm [57]

```
sort level numbers by descending level sizes and store them in array sl;
for i = 1 → n do
  if sl[i] = 1 then
    sift_down(i, n); // the BDD size is recorded in sift_down();
  else if sl[i] = n then
    sift_up(i, 1); // the BDD size is recorded in sift_up();
  else if (sl[i] − 1) > (n − sl[i]) then
    sift_down(i, n);
    sift_up(n, 1);
  else
    sift_up(i, 1);
    sift_down(1, n);
  end if
  sift_back();
end for
```

algorithm is described in Algorithm 1, where the levels are sorted by descending level sizes. The largest level contains the most nodes and is considered first. Then, the variable is moved downwards until the terminal nodes and upwards from the initial

position to the top. In the previous steps, the ROBDD size resulting from every variable swap is recorded. In the end, the variable is moved back to the position, which led to a minimal ROBDD size. Here, the size could be either the number of nodes or the 1-paths.

4.3.3 Node Optimization

The sifting algorithm, based on the efficient exchange of adjacent variables, is able to dynamically reorder the structure of ROBDD in a way to change the number of decision nodes. While the sifting algorithm is executing, we record the ROBDD node size for each variable swap. In the meanwhile, we also store all the $< node,\ 1-path >$ pairs into a $< node,\ 1-path >$ array, which can further be used in Section 4.3.5 for determining the optimal $< node,\ 1-path >$ pair. In the end, a ROBDD with minimum number of decision nodes is derived.

In Section 4.3.1, we strive to define the TRS ROBDD accurately enough so that no semantically redundant 1-paths exist. Thereby, we say *the quantity and semantic meaning of 1-path of TRS ROBDD do not vary with the changes of variable ordering of TRS ROBDD*. As we show that a TRS ROBDD might introduce extra decision nodes, we can further use node optimization to reduce its node size. Clearly, in this case node optimization only improves the node size and keeps the 1-path unchanged.

In memory, each decision node requires an index and pointers to the succeeding nodes [55]. Since each decision node in a ROBDD has two pointers, the memory size required to represent a ROBDD is given by Equation 4.11.

$$Memory(ROBDD) = (1 + 2) \times nodes(ROBDD) \tag{4.11}$$

4.3.4 Path Optimization

Apart from the ROBDD node optimization, another criterion (i.e. 1-paths number) for ROBDD optimality is also considered. As the variable ordering heavily influences the number of nodes within a ROBDD, the sifting algorithm can be modified to minimize the number of 1-paths instead of the node size for a given ROBDD. After each swap of two adjacent variables, i.e. after processing all nodes in the two levels, changes are only propagated from those two levels down to the terminal nodes. During modified sifting no upper limit on the number of 1-paths is used to stop the swapping operations [51].

Similarly, we record the 1-paths number of ROBDD for each variable swap. In the meanwhile, we also store all the $< 1-path,\ node >$ pairs into a $< 1-path,\ node >$

array, which can further be used in Section 4.3.5 for determining the optimal $< 1 - path,\ node >$ pair. In the end, a ROBDD with minimum number of 1-paths is derived.

Although for Equation 4.2, the 1-paths number can be reduced from 4 to 3, path (x_{11} = true, x_{12} = true, x_{16} = false, x_{17} = true) still exists (see Figure 4.1 (d)), where x_{11} and x_{12} are true at the same time. Path optimization relieves the work of SLA management, but it does not eliminate the semantically redundant 1-paths completely. Thus, the SLA management still needs to evaluate the validity of each 1-path despite the partial reduction of 1-paths.

4.3.5 Multicriteria Optimization Problem

As it is demonstrated in [51], the number of paths can be significantly reduced for some benchmarks. At the same time, the number of nodes does not necessarily increase and may even be reduced.

Algorithm 2 Calculate optimal (node, path) pair

```
store node and 1_path size of node_minimization() into n_nopti and p_nopti;
store node and 1_path size of path_minimization() into n_popti and p_popti;
if n_nopti = n_popti then
  return (n_popti, p_popti);
else if p_nopti = p_popti then
  return (n_nopti, p_nopti);
else
  return node_path_pair_selection();
end if
```

The path optimization algorithm re-constructs a ROBDD with the minimal number of 1-paths, but not necessarily the minimal number of decision nodes. Similarly, node optimization algorithm re-constructs the ROBDD with the minimal number of decision nodes, but not necessarily the minimal number of 1-paths. As Lemma 5.5 in [51], for all boolean functions of two or three variables there exists a ROBDD that is minimal both in size and the number of 1-paths. This is however not true for functions of more than three variables. Therefore, this becomes a multicriteria optimization problem for gaining a minimal number of decision nodes and 1-paths (see Algorithm 2).

In Algorithm 2, if the node size of a ROBDD after executing the path optimization is equal to that after executing the node optimization, the result of the path optimization will be taken. An analogous statement holds if the path size of a ROBDD after

executing the node optimization is equal to that after executing the path optimization, thus we take the result of the node optimization. These two situations mean we can get minimal size of paths and decision nodes at the same time. Otherwise, we cannot have an optimized ROBDD in both ways. A compromise between the two measures should be considered in *node_path_pair_selection*(). Here, end users have to specify it according with their requirements. For example, the customer may only concern the number of 1-paths regardless of number of nodes or the other way around. A sample solution based on geometric distance [58] for this multicriteria optimization problem could be resolved by Equation 4.12. Selection of a point that is closest to the optimal point. A preference is given to the point with the smallest number of 1-paths, when multiple points have the same distance to the optimal point.

$$Distance\_to\_optimal\_point = \sqrt{(n\_opti)^2 + (p\_opti)^2} \tag{4.12}$$

4.4 Implementation and Experimental Verification

We design and implement the model in accord with the Unified Modeling Language (UML) diagram B.2 illustrated in Section B.2 in Appendix.

Based on the SLA template (Table 4.1), we assume that a customer starts an SLA negotiation for service "IaaS-1", given that business hours are between 09:00 and 17:00. The customer needs one VM with 2 or 4 Central Processing Unit (CPU) cores, CPU speed is either 2.4 Gigahertz (GHz) or 3.0 GHz, memory size is 2 Gigabytes (GB), network is 10 GB/s Net-1, either OS-1 or OS-2 is selected and customer's private image is uploaded, service availability must be 99.99% or higher; *Or* a SLA with 1 CPU core, CPU speed must be 2.4 GHz, memory size is 1 GB, network is 10 GB/s either Net-1 or Net-2, either OS-1 or OS-2 is selected, no private image is uploaded, service availability is at least 99.9% or higher. Furthermore, JavaBDD [59], a Java library for manipulating BDDs, is the tool we chose for setting up the BDD handling and programming environment. Table 4.2 illustrates the set of facts and clauses that we will use for this use case scenario. These facts and clauses can be considered as boolean variables, which are evaluated to be true or false. The SLA can also be correctly evaluated if it is modeled according to the following equations.

$$f_1 = x_1 \wedge x_2 \tag{4.13}$$

$$f_2 = x_3 \wedge (x_5 \vee x_6) \wedge (x_7 \vee x_8) \wedge x_{10} \wedge x_{11} \tag{4.14}$$

Table 4.2: Example clauses of an SLA template

Variable	**Proposition**	**Proposition Type**
x_1	Service_name = "IaaS-1"	Fact
x_2	Business_hours = 09:00 - 17:00	Fact
x_3	VM = "1"	Clause
$x_5 \vee x_6$	CPU_core = "2" or "4"	Clause
$x_7 \vee x_8$	CPU_speed = "2.4 or 3.0" GHz	Clause
x_{10}	Memory = "2 GB"	Clause
x_{11}	10 Gb/s Network = "Net-1"	Clause
x_{13}	Storage = "Private image"	Clause
$x_{14} \vee x_{15}$	Storage = "OS-1 image" or "OS-2 image"	Clause
x_{16}	Service_availability $\geq$ 99.99%	Clause
x_3	VM = "1"	Clause
x_4	CPU_core = "1"	Clause
x_7	CPU_speed = "2.4 GHz"	Clause
x_9	Memory = "1 GB"	Clause
$x_{11} \vee x_{12}$	10 Gb/s Network = "Net-1" or "Net-2"	Clause
$\neg x_{13}$	Storage != "Self image"	Clause
$x_{14} \vee x_{15}$	Storage = "OS-1 image" or "OS-2 image"	Clause
x_{17}	Service_availability $\geq$ 99.9%	Clause

$$f_3 = x_{13} \wedge (x_{14} \vee x_{15}) \wedge x_{16} \tag{4.15}$$

$$f_4 = x_3 \wedge x_4 \wedge x_7 \wedge x_9 \wedge (x_{11} \vee x_{12}) \tag{4.16}$$

$$f_5 = \neg x_{13} \wedge (x_{14} \vee x_{15}) \wedge x_{17} \tag{4.17}$$

$$SLA = f_1 \wedge ((f_2 \wedge f_3) \vee (f_4 \wedge f_5)) \tag{4.18}$$

This SLA request is firstly transformed to an initial ROBDD with 29 decision nodes and 40 1-paths. Memory requirement is 87 indices and pointers. By applying

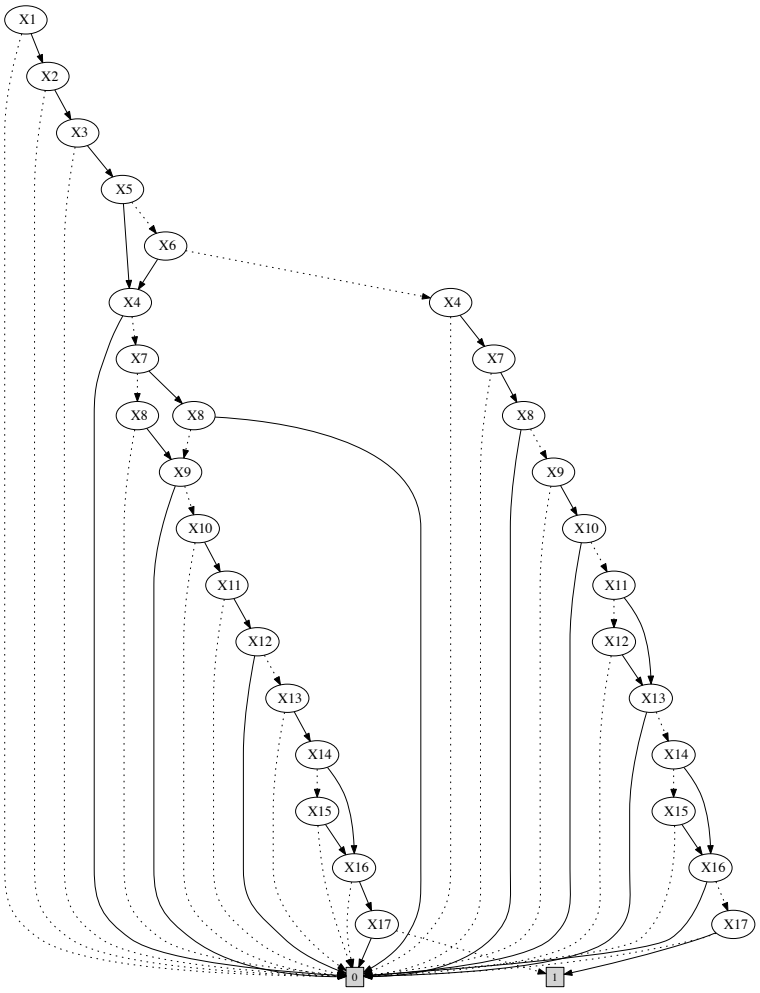


Figure 4.2: The ROBDD with 30 decision nodes and 12 1-paths by applying term rewriting and node optimization

the term rewriting, the number of 1-paths of the initial ROBDD is reduced to be 12, however, the decision nodes are increased by 2 (31 decision nodes). As already mentioned in Section 4.3.3, we can further optimize this ROBDD by using the node optimization. There will be 30 decision nodes in this ROBDD (see Figure 4.2). Memory requirement is 90 indices and pointers. Alternatively, by applying the path and node optimization, the initial ROBDD is optimized to be with 21 decision nodes and 12 1-paths. Memory requirement is 63 indices and pointers. (see Figure 4.3).

Eventually, we attempted to simulate the similar SLA negotiation above for 1000 times to compare the performance of three approaches. We reused our planning and optimization algorithms in [60] to balance the price, profit and failure rate. Each time, the SLA template was customized randomly by selecting different combinations of options for all the SLA terms. Each SLA template was first transformed into an

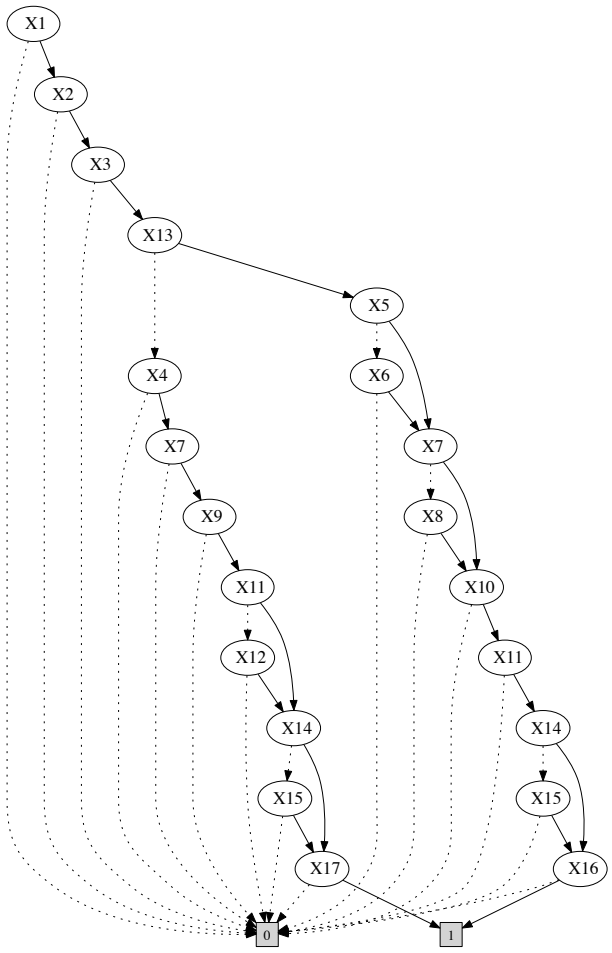
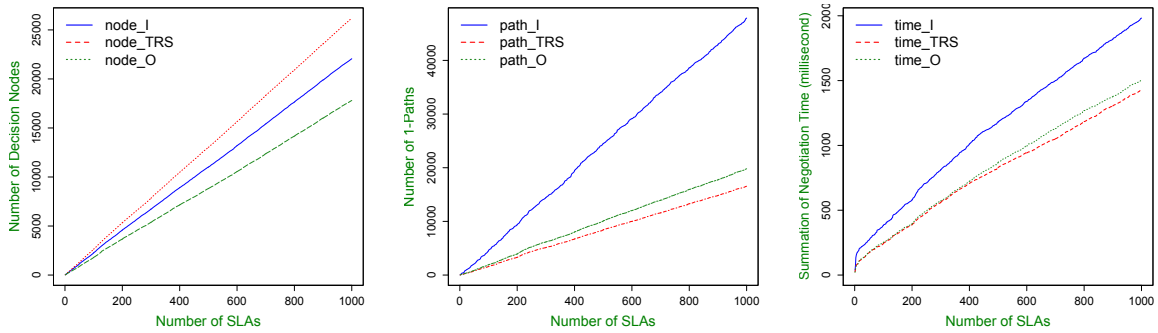


Figure 4.3: The ROBDD with 21 decision nodes and 12 1-paths by applying path and node optimization

initial ROBDD using the approach in [36]. Then we rewrote the same initial ROBDD using TRS, and following that we used BDD node optimization to reduce the nodes of TRS ROBDD by the greatest extent. Finally, we optimized the initial ROBDD by using BDD optimization algorithms. All the decision nodes and 1-paths for each approach were aggregated and compared with each other. Additionally, we assumed that each round of negotiation starts when the customer submits the SLA template to SLA manager and stops when the SLA manager sends an offer or a counter-offer back to the customer. The total negotiation time of each approach was counted. Thus, the whole simulation took approximately 13521 seconds on a 1.8 GHz processor. Initial ROBDDs had 22123 decision nodes ($node\_I$) and 47880 1-paths ($path\_I$) and the negotiation time ($time\_I$) was 1982 milliseconds (ms). Figure 4.4 (a) illustrates that the number of decision nodes increased proportionally in all approaches. The TRS ROBDDs had the most decision nodes ($node\_TRS$=26421), because the least



(a) (b) (c)

Figure 4.4: The number of nodes (a), 1-paths (b) and negotiation time (c) statistics of the initial ROBDD (blue), the TRS ROBDD (red) and the ROBDD after running BDD optimization algorithms (green)

number of 1-path ($path\_TRS$=16554) in Figure 4.4 (b) and the least negotiation time ($time\_TRS$=1432 ms) in Figure 4.4 (c) were realized at the cost of nodes. *Eclectically*, BDD optimization algorithms not only reduced the number of decision nodes ($node\_O$=17838) and redundant 1-paths ($path\_O$=19804) efficiently, but also showed their time saving ($time\_O$=1505 ms) feature.

The TRS approach accurately represented the requirements of the customer, therefore all its 1-paths were valid paths. By setting $path\_TRS$ as a benchmark, the differences between $path\_TRS$ and the 1-path number of other approaches were semantically redundant 1-paths. Thus, the $time\_I$ and $time\_O$ were greater than the $time\_TRS$, since they had to use an extra algorithm to verify the invalid 1-paths during the SLA negotiation. Experimentally, it was proved that after running 1-path verification, the rest 1-paths of the initial ROBDD and the ROBDD by applying path and node optimization was exactly the same as the one of the TRS ROBDD with respect to quantity and semantic meaning.

In summary, the TRS/node optimization led to the most reduction in number of 1-paths (65.43%) and negotiation time (27.75%), although the number of decision nodes had an increase of 19.43%. Furthermore, BDD node/path optimizations led to the most reduction in number of decision nodes, which was 19.37%. Moreover, they had reduction in number of 1-paths (58.64%) and negotiation time (24.07%), but it may still not completely eliminate all the semantically redundant 1-paths, for which reason, the SLA manager requires more time to verify 1-paths during the negotiation. However, it could be a good eclectic approach. From another standpoint, for example, at 1000 ms of the SLA negotiation, the respective SLA number of three

approaches were $SLAs\_I$=397, $SLAs\_TRS$=650 and $SLAs\_O$=605. Therefore, the TRS approach had the most SLAs and potentially led to more profits and higher customer satisfaction.

4.5 Discussion

In this chapter, we show that by applying the ROBDD node optimization, the number of nodes can be decreased efficiently. Thus, SLAs occupy less memory space. Additionally, the size of semantically redundant 1-paths can be eliminated through the ROBDD path optimization. Consequently, the SLA negotiation is accelerated by assessing fewer SLA proposals (1-paths). Furthermore, we build on the observation that the options of an SLA term may be mutually exclusive. Thus, an SLA term is rewritten as correctly selecting an option for this term. This process can be treated as an implementation of an SLA term rewriting system. Hence, the approach above can eliminate the number of semantically redundant 1-paths at ROBDD creation phase. Finally, we discuss the strengths and weaknesses of the approaches with an IaaS use case.

Having an optimal SLA ROBDD representation structure, SLA management architecture is able to perform the corresponding analysis processes for each potential offer in regard to its feasibilities. The feasibilities in this thesis refer to two aspects, namely the feasibility of business policies and the feasibility of resource availability. In next chapter, we propose a model of business policies and resource allocation to conduct the feasibility problems during SLA negotiation.

Chapter 5

Business Policies and Resource Planning During SLA Negotiation

An SLA is used as a means to establish formally the agreement between the provider and the customer. This differs significantly from simple provisioning requests, as it contains guarantees for quality and penalties. IT service management in the broader sense overlaps with the disciplines of business service management and IT portfolio management, especially in the area of IT planning and financial control [10]. Namely, efficiently leveraging the IT business modeling that is subject to IT resource management is the chief task during SLA negotiation. Most recently, as it is defined by the ISO and the IEC, managing business plan is one of the important roles of cloud service provider for preparing and adjusting a business plan that typically addresses the following activities [26]:

- The offering of one or more cloud services to the customers;
- Handling both financial and technical aspects of the services, the target customer set, the contracts and SLAs, the channels to market and sales targets;
- Tracking the sales and service usage against the plan to ensure that financial targets are achieved for the cloud service provider;
- Outsourcing or operating the cloud services of the peer (third-party) cloud service providers with activities, such as selecting and using one or more services of a peer cloud service provider; managing the business aspects of the cloud services of a peer cloud service provider.

Recently, many publications [61, 62, 63, 64] discuss the topic of SLA-based business utility modeling. We however contribute to SLAs negotiation for the infrastructure

cloud service by leveraging the bound of resource availability and outsourcing to other providers if the demand cannot be served locally.

During and after the SLA negotiation, it is essential to reserve the cloud resource in advance for the customer. It implies that before a provider guarantees a customer access to a resource, the provider has to assess the feasibility of resource availability and reserve it in advance. Many publications [2, 65, 66, 67, 68, 69] propose various methodologies of service advanced reservation. However, in our work, we propose a reservation methodology in detail with well-developed SLA management strategy, which clearly addresses when, why and how to balance all the VMs in a server.

In the rest of the chapter, we stepwise outline the business policies optimization and resource reservation planning problem, which we have derived in Section 1.2 and 2.4.

5.1 Business Modeling of IaaS Cloud

Certain assumptions made as a basis for the proposed solutions:

- If the provider owns resources (i.e. it is not a pure resource reseller) it always prefers to utilize them first, before turning to subcontractors.

- There are no locality or other kinds of dependencies between the requested resources. They can be arbitrarily split between different locations and administrative domains, if needed. Nevertheless, we must make an effort to keep as many as possible co-located, to facilitate data exchange and as well as performance. Thus, if resources are outsourced, the number of subcontractors must remain as low as possible.

- The customer's requirements are strict. There is no SLA negotiation other than consecutive requests for quotes, possibly followed by a message to establish the agreement for the latest quote. The response to a customer request for N resources of some type, at certain quality, is a tuple of the form (n, c, t). Element n implies the confirmed resource quantity, c stands for the price and t means the validity period for this quote, i.e. starting time t_{start} and ending time t_{end}. Based on the starting time, we distinguish the provisioning of the service as immediate provisioning (t_{start} is 0) and advance reservation (starting time is a time point in the future). If the provider cannot offer as many resources as the user requested, it is considered possible to return a value $n < N$.

- There is a specific and unambiguous penalty scheme for violated SLAs. All parties involved, both end-customers and providers, are aware of this scheme either by default or via negotiation. Since the penalty is part of SLA operation and termination phase, we will model and discuss it more in the next part of the thesis. In this chapter, without loss of generality, in our approach this scheme is defined as a fixed penalty in the case that the SLA is violated.

- Providers decide at runtime on the importance of accepting and enforcing each incoming SLA request. They make dynamic decisions about how far they will go to ensure that an SLA is not violated, based on business criteria such as costs, profit and failure risks.

Let us assume resource types $R^1, R^2, ..., R^n$, each bearing characteristics $H_1^i, H_2^i, ..., H_{m_i}^i, 1 \leq i \leq n$. Those characteristics refer to inherent resource details, e.g., clock speed for CPU cores, memory, storage, network, etc. More detailed information can be referenced in Table 4.1 of Section 4.1, here we do not consider the exact semantics of the qualities. Rather, they will come into the model as coefficients for defining costs of investment.

5.1.1 Internal and External Implementation Cost

As discussed in Section 2.4, we aim to minimize the cost for implementing a solution, while respecting constraints for profitability and financial risk related to SLA violations (i.e. failure to comply with SLAs). As Equations 5.1 to 5.5 present, we model cost of investment C^i for a solution involving resource R^i as the sum of internal cost of investment C_I^i (i.e. resources utilized internally) and external cost of investment C_E^i (i.e. sub-contracted resources). Internal cost of investment is then modeled after the base cost per resource unit C_B^i for a solution given standard (baseline) QoS. C_B^i is multiplied by a factor σ^i that models the cost of increased QoS as it is given within the quote/SLA request by means of conditions $Q_1^i, Q_2^i, ..., Q_{r_i}^i$ for the qualitative characteristics of the resources, and applying price reductions for bulk purchases (i.e. large values for resource quantity N^i). Function d^i models such price reductions. Especially, it also takes the provisioning type (i.e. immediate provisioning or advance reservation based on the starting time), the utility of the service into account. Then, the value it returns represents the percentage of resources that the customer receives for free. Finally, the total amount of resources requested by the user is multiplied

with the previous figures. The total cost of investment C is the sum of cost for all resources.

$$\sigma^i = [1 - d^i(N^i, t_{start}, t_{end})] \cdot f^i(Q_1^i, Q_2^i, ..., Q_{r_i}^i) \tag{5.1}$$

$$C_I^i = N^i \cdot \sigma^i \cdot C_B^i \tag{5.2}$$

$$C^i = (1 + \beta^i) \cdot C_I^i + C_E^i \tag{5.3}$$

$$C = \sum_i C^i f^i(Q_1^i, Q_2^i, ..., Q_{r_i}^i) \geq 1 \tag{5.4}$$

$$0 \leq d^i(N^i) \leq 1 \tag{5.5}$$

The improved quality represented by function f^i corresponds to increased measures, such as committing additional resources to improve availability, or have more people in the data center. As such, it also reflects increased costs. For instance, as the degree of availability approaches the ideal of 100% availability, the cost of the solution increases more rapidly. Thus, the cost of 99.95% availability is significantly greater than the cost of 99.5% availability, the cost of 99.5% availability is significantly greater than 99% amongst others [44]. In general, each provider has its own methods to implement increased QoS for a specific service, therefore the only generic modeling option is to use its effect on the implementation cost.

Variable β^i is a quality multiplier that affects the internal cost of investment in Equation 5.3. It indicates the provider's dynamic policy with regard to the additional measures to take, in order to safeguard the respective guaranteed quality of a certain specific SLA.

5.1.2 Profit

Function f^i returns a cost result for some standard failure probability per resource, e.g., 5%. Yet, a provider may wish to diverge from typical contract violation risks and further decrease this probability, by setting β^i to a value larger than 0. Overall, it is not sensible to charge the customers for additional quality (than what they originally requested), only because it is in the provider's best interest. Thus, the provider will have to compress its originally targeted profit F, by subtracting these extra costs:

$$F^i = g^i(C_I^i, C_E^i) - \beta^i \cdot C_I^i \tag{5.6}$$

$$F = \sum_i F^i \tag{5.7}$$

In general, it is reasonable to model profit based on the cost of implementation (e.g., as a percentage of it), as it can be given by function g^i. This function also models other factors that affect profit, and which do not have to do with the quality of the implemented solution. For instance, in order to sign up a customer in hope of additional future contracts, a provider may actually make no profit, but rather sell at cost level. Function g^i would then return a value of 0. We consider such decisions to be made on a business level, by means different than the system described in this thesis.

5.1.3 Failure Probability

We assume by default that a superior solution is also a more expensive solution; and that a more expensive solution is at least as good as the cheaper ones. Therefore, as β^i increases, the cost of implementation also increases (or, at the very least, does not decrease). If reaching the requested QoS for a resource R^i demands –according to some model– a factor of β_1^i, then using any larger factor β_2^i in the calculations would mean increased cost, but also perhaps further increased quality. In other words, it would be less probable that the requested QoS will not be met and that the SLA will be violated. Thus, if P_V^i is the probability that an SLA will be violated due to R^i failing to deliver (Equation 5.8), we have:

$$P_V^i = h^i(\beta^i, T^i) \tag{5.8}$$

$$T^i = \begin{cases} 0, & \text{if s} = 0 \\ \frac{1}{s} \cdot \sum_{j=1}^{s} T_j^i, & \text{if } s > 0 \end{cases} \tag{5.9}$$

$$T_j^i = \frac{violatedSLA_j^i}{SLA_j^i} \tag{5.10}$$

$$\frac{dP_V^i}{d\beta^i} \leq 0 \tag{5.11}$$

T_j^i is a measure of reputation of subcontractor j involved in the delivery of resource R_i and the respective SLA. More specifically, it models the historical failure rate (violated SLAs as a percentage of total SLAs (Equation 5.10)) for previous contracts established for this resource, with the specific subcontractor. As such, the failure rate is always expected to take values between 0 and 1.

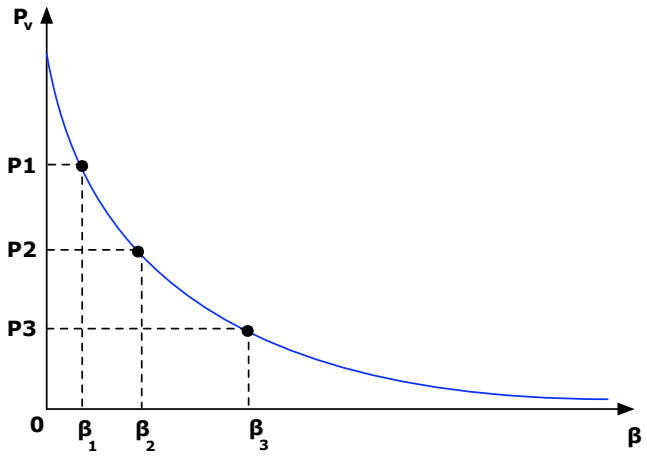


Figure 5.1: Relationship of failure probability and quality factor

T^i aggregates the failure rates of all subcontractors for the specific resource and the specific SLA currently under negotiation. If all resources are committed locally then T^i equals zero, otherwise it is given by Equation 5.9, where s is the number of subcontractors chosen and involved in the new contract.

Figure 5.1 illustrates Equation 5.11 in a more intuitive manner for an example relation of reverse logarithmic nature. The graph never reaches the horizontal axis, assuming that there are events outside the control of the provider (force majeure), hence the probability of failure will never equal zero. Nevertheless, with none or small additional costs committed (low value for β) it is more probable to diverge from the QoS level guaranteed by the SLA. The further we go to safeguard the agreed QoS level (by providing more resources via the β variable), the less probable it is to violate the SLA.

A similar graph would represent the relationship between failure probability and subcontractor reputation. The higher the reputation (i.e. the lower the past failure ratio), the smaller would be the probability that the SLA with the subcontractor, and hence, the original SLA with the end customer would fail.

5.1.4 Return on Investment

In business, ROI refers to an investment of some resource yielding a benefit to the investor. A high ROI means the investment gains compare favorably to investment cost. To calculate the ROI, the benefit / return of an investment is divided by the cost of the investment. In purely economic terms, it is one way of considering profits in relation to capital invested [70]. In ROI model, other than implementation cost

and profit, in case the service is violated, it is service providers responsibility to pay for the compensation, i.e. the penalty. This definitely influents the implementation cost and should be considered as part of investment cost. Thus, we can further extend the ROI as Equation 5.12 for resource i and as Equation 5.13 for all resources:

$$ROI^i = \frac{g^i(C_I^i, C_E^i) - \beta^i \cdot C_I^i}{(1+\beta^i) \cdot C_I^i + C_E^i + P_i(QoS_1, ..., QoS_t)} \tag{5.12}$$

$$ROI = \frac{\sum_i F^i}{\sum_i C^i f^i(Q_1^i, Q_2^i, ..., Q_{r_i}^i) + \sum_i P_i} = \frac{F}{C+P} \tag{5.13}$$

5.1.5 Complete Problem Definition

Based on the previous sections, the problem we are trying to solve is to minimize cost (Equation 5.14) while profit and the probability of failure remain within acceptable limits as dictated by high-level business rules (Equations 5.15 and 5.16). Consequently, the entire problem boils down to how to maximize the value of Equation 5.13 for sake of gaining optimal ROI. F^* represents a minimum acceptable profit, that depends on the customer's profile; and P_V^{i*} represents a maximum acceptable failure probability, which may also be associated with specific customers or other business conditions at the time of negotiation.

$$\sum_i (1+\beta^i) \cdot N^i \cdot \sigma^i \cdot C_B^i + C_E^i \tag{5.14}$$

$$\sum_i \left[g^i(C_I^i, C_E^i) - \beta^i \cdot C_I^i \right] \geq F^* \tag{5.15}$$

$$h^i(\beta^i, T^i) \leq P_V^{i*} \tag{5.16}$$

$$0 \leq \beta^i, \forall i \tag{5.17}$$

Equation 5.15 can be transformed to be Equation 5.18.

$$\beta_{max}^i \leq q^i(F^i, F^*, C^i) \tag{5.18}$$

We can further transform the Equation 5.16 as:

$$\beta_{min}^i \geq j^i(T^i, P_V^{i*}) \tag{5.19}$$

Therefore, if there is no intersection between Equation 5.18 and 5.19 as Figure 5.2 (a) illustrates, which means service provider cannot offer the service with acceptable

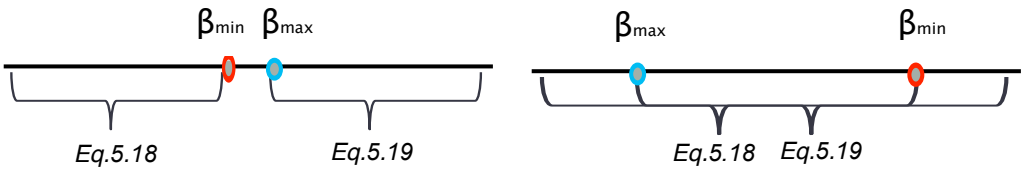


Figure 5.2: β^i_{max} for gaining enough profits and β^i_{min} for keeping failure rate low. (a) without intersection area. (b) with intersection area.

failure rate and profit. In this case, service provider either has to renegotiate with customer for a service with new configuration or refuse the coming request. On the other hand, if there is such an intersection (Figure 5.2 (b)), according to our strategy, service provider selects the value of β^i_{min} for the sake of maximum profit and acceptable failure rate.

Eventually, they all depend on the decision variable vector $\mathbf{b} = (\beta^1, \beta^2, ..., \beta^n)$. Especially for failure probability, it is useful to underline that its minimization is practically equivalent to optimizing long-term reputation. As a side note, it should be mentioned that changes in $\mathbf{b}$ do not affect the quality of outsourced implementations (i.e. there is no compensation), although they can improve the total failure probability P^i_V.

The increase or decrease of the quality multiplier for the resources may affect profit and failure probability in converse ways. Higher quality means higher costs, lower profits, and less chances to fail. What we need to achieve is to find the lowest possible quality multipliers according to risk management policies, the highest possible according to profitability policies, and confirm they are not excluding each other. Then, the lower values can be used to compute the cost of implementing the solution with maximum profit and within acceptable limits for failure probability.

We can also see that cost and failure probability are affected by the costs for external resources $\mathbf{C_E} = (C^1_E, C^2_E, ..., C^n_E)$ and the failure rates of the subcontractors for those resources, $\mathbf{T} = (T^1, T^2, ..., T^n)$. Because of the first assumption outlined in Section 5.1, we can first solve the problem of optimized outsourcing to subcontractors for excess resources, and then proceed to solve the Equations 5.14, 5.15 and 5.16 taking into account the solutions from the former. The fact that we have two criteria for selecting the distribution of the resources makes things more complex. We will apply a simple heuristic and try to also reduce the total number of subcontractors (due to the second assumption of Section 5.1), according to their resource availability. If

necessary according to business requirements, reputation constraints will be applied to exclude providers that do not meet certain thresholds.

Before continuing to discuss the proposed solutions to the problems, it must be noted that the case of an isolated cloud provider (i.e. a provider that does not delegate resource provisioning to other providers, rather is bound solely to the availability of its own resources) can be modeled as above, with C_E^i and T^i always equal to 0. Additionally, as also mentioned earlier in the text, the case of a resource reseller who has no private infrastructure is sufficiently represented as well. It only has to solve the bi-criterion resource assignment problem and then add a profit to the price according to business policies.

5.2 Architecture of Subcontracting

5.2.1 Subcontracting

Through virtualization technologies such as VMWare [71] and Xen [72], physical resources can be partitioned into a flexible and scalable virtual computing platform. Customers can customize and rent VMs according to their preferences, meanwhile, service providers seek to maximize the utilization of their resources to yield optimal profits. Large scale service providers with virtually unlimited resource capacity, like Amazon EC2 [6], can provision virtualized resources immediately. Hence, customers do not need to negotiate beforehand. However, small and medium-sized providers with relatively limited resource capacity may not serve all requests immediately. Recently many publications [21, 22, 23, 73, 74, 75] discuss the topic of SLA management for cloud computing, but most of them are looking at it from a conceptual and architectural point of view, lacking of providing a consistent methodology for selecting infrastructure subcontractors and generating requests to them. Thereby, outsourcing via subcontracts should be included as part of the decision process in order to achieve additional profit but also to benefit customers when local resources are not sufficient. Taking this as motivation, we introduce our subcontracting model in the following part of the section.

The first step to execute is to see whether there is a part (perhaps all) of the requested resources in the incoming SLA request that the provider cannot offer. Let N_L^i be the number of local resources of type R^i, characteristics $H_1^i, H_2^i, ..., H_{m_i}^i$ and quality properties $Q_1^i, Q_2^i, ..., Q_{r_i}^i$. We need to take into account the resource constraints of candidate subcontractors. According to the SLA templates they publish, we can see whether they offer the resources we need, so that we can contact them

with an SLA request. As we stated, our eventual choice must take into account their reputation (failure history) and the price quotes they provide. These two competing criteria constitute a multi-objective optimization problem, which is oftentimes solved so that a set of equally good (non-dominated) solutions are produced. Then, a decision maker chooses one of those which is considered "optimal" according to her judgement. Conversely, we will employ the scalarization technique of ideal point [76], where we are measuring a point's distance from what would be an ideal combination for cost and failure ratio. Clearly, that would be the point $(0, 0)$, i.e. perfect service given for free. The reason we are scalarizing, instead of searching for multiple candidate solutions in the form of a Pareto front [76], is that we wish to implement this step in a completely automated manner, without involving a human decision maker. Performance considerations also apply.

We start with the set V of all candidate subcontractors, according to the SLA templates they publish. If needed, according to risk mitigation strategies and respective business rules, we remove from the set all candidates that do not meet a certainly low threshold for failure history. Following, we submit a quote request for the full amount of N_L^i, to all members of V. Some of them may be unable to satisfy the request for the amount of requested resources, and respond with the maximum capacity they can offer (e.g., as in Figure 5.3, for resource amount L_2, candidates A and D do not have sufficient resources).

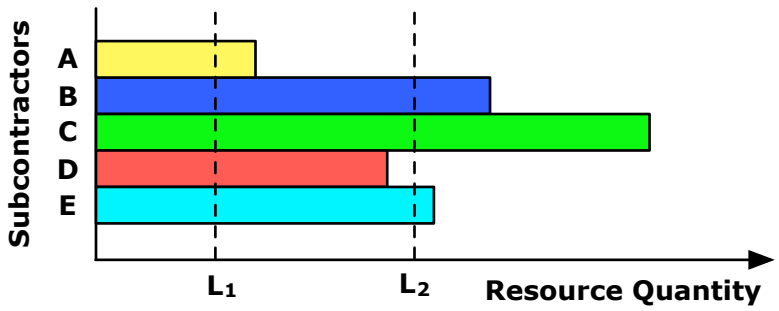


Figure 5.3: Resource request levels, and subcontractors with different capacities

If this capacity is too small, under some predefined threshold, the respective subcontractors are removed from V and the process, in order to avoid trivial contracts. If it is significant (although insufficient), a second request is sent to them for the maximum possible amount of resources. Eventually, for all providers we have a price that corresponds to bulk purchases but respects their resource limitations.

5.2.2 Heuristic Selection Criteria of Service Providers

For each provider in V, we compute its distance from the ideal point, as

$$D = \sqrt{C_j^{i2} + T_j^{i2}} \tag{5.20}$$

C_j^i being the price per resource unit from provider j and T_j^i being its historical failure rate (Figure 5.4). After we choose the closest one, we strive to establish an agreement for a quantity that respects its declared capacity. If there is more than one with the same (smallest) distance, we choose either the cheapest per unit or the one with the largest capacity, depending on what is least expensive for the total quantity. In the extreme case that costs are the same, we choose one at random. If at the end of this process there are still unassigned resources, we remove currently utilized subcontractors from V and repeat with the new excess amount until there are no unassigned resources. The process is executed for all resource types for which we do not have enough local capacity. The agreements can contain service performance

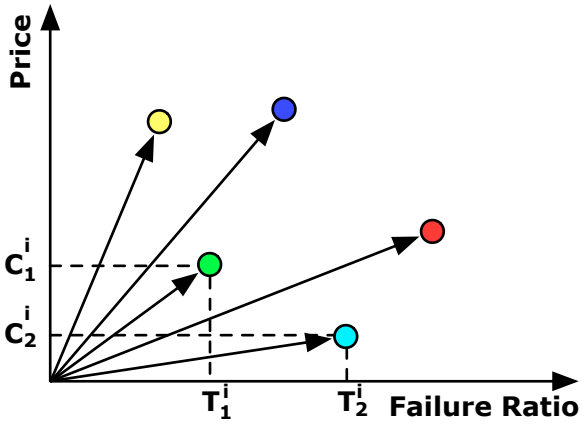


Figure 5.4: Price per unit and failure rates for subcontractors

metrics for the levels of availability, response time or both, and their corresponding service level objectives. Once the agreements are in place, organizations can then measure the quality of service provided by the service providers. SLA violations, if any, can be known in advance so that companies can take action to ensure their online reputation is not adversely affected. For instance, by migrating the violated service to another host that has more available capacity, or simply entertain the customer with compensation. Therefore, we say:

$$Fx_j^i = \frac{resolvedSLA}{violatedSLA} \tag{5.21}$$

where Fx_j^i being the historical SLA violation fixing rate. The higher the value of Fx_j^i, the better after-sales service that such service provider is offering. Thus, based on the previous model, we can extend it to be a three dimensional decision-making space:

$$Service\ Provider\ Selection = \{T_j^i, (1 - Fx_j^i), C_j^i\} \tag{5.22}$$

We can further formulize the distance from the ideal point by setting each metric with a quantification variable as:

$$D = \sqrt{(\theta * T_j^i)^2 + (\lambda * (1 - Fx_j^i))^2 + (\sigma * C_j^i)^2} \tag{5.23}$$

where $\theta+\lambda+\sigma = 1$. (0, 0, 0) implies the ideal service with no SLA violation and price is free. (1, 0, 0) means that customers only concern about failure rate. Similarly, (0.5, 0.3, 0.2) means failure rate is more important than other facets. Thus, it is more applicable to customize the service provider selection by assigning the concerned metrics with weights.

5.3 Local Resource Configuration

Until now, we have the external cost per resource C_E^i and the subcontractors failure rate per resource T^i. Using these values, we can solve Equations 5.15 and 5.16 to come up with a decision space for vector $\mathbf{b}$ depending on the dynamic, customer and/or request-specific thresholds for minimum acceptable profit F^* and maximum acceptable failure probabilities P_V^{i*}. Apparently, for Equation 5.15 we will receive a maximum possible value of each β^i, while for Equation 5.16 we will receive minimum values. If the latter are higher than the former, then this means that the problem cannot be satisfied, and therefore the incoming SLA request must be rejected (or dealt with according to the provider's best understanding). Otherwise, the lowest values can be chosen to be used in Expression 5.14, for computing the final price quote (as the sum of implementation costs and total profit) to return to the prospective customer. As a result, this approach dramatically lowers costs investment and increases profits so as to achieve an optimal ROI.

Eventually, to apply the complete methodology, a provider would need to have the following available in advance: A function to provide price reductions for bulk purchases, advance reservation and counter-offer deviation; A baseline resource price, and a function to associate increased QoS to a price relevant to standard prices; The expected profit as a function of implementation and outsourcing costs; A minimum acceptable profit depending on each request (e.g., customer, class of service, etc); And

the failure probability as a function of such failure / violation precaution measures. While the former four are business-related and largely the result of respective high-level decisions, the latter is a statistical property that can be acquired by design-time models and monitoring data.

5.4 Resource Advance Reservation

Other than outsourcing to third party, for the sake of increasing the utilization of the local resources and satisfying as many customers as possible, an advance reservation methodology during SLA negotiation becomes necessary. Relying on advance reservation, IaaS providers are able to lock infrastructure resources to guarantee that customers can invoke the services during agreed time interval in the future.

In utility computing, advance reservation based resource management is still an active area of research. In grid computing, Venugopal et al. [24] and J. Xu et al. [69] propose different advance resource reservations protocols by verifying the nodes capacity without considering virtualization. Castillo et al. [68] proposed an approach for managing the advance reservation using computational geometry in grid computing. However, due to the differences between cloud and grid computing with regard to application and virtualization [2], resources could be partitioned in a virtual way rather than be occupied completely by a job. Furthermore, there could be several VMs that run simultaneously in a server, which will introduce a more complicated fragmentation situation. Thus, we take one step further to represent resource availability and fragmentation of the virtual resources dynamically and to merge and demerge the request points with reservation points by efficiently allocating and reallocating the VMs as well as VM (live-)migration. In clouds, some studies (e.g., Haizea) from Sotomayor et al. [65] argue that in small clouds resources could be limited and in this case requests for resources have to be prioritized and queued. Moreover, Lucas et al. [67] proposed Flexible Advance Reservations (FAR) lease, which is supposed to be an extension of Haizea. And Netto et al. [66] propose a concept of "flexible time slot" that efficiently solves the reservation problem. However, in our work, without prioritizing or queuing the requests, we propose a reservation methodology in detail with well-developed SLA management strategy, which clearly addresses when, why and how to balance all the VMs in a server. For that reason, a suspension could happen if and only if it does not break the guarantees of SLA. Furthermore, the fragmentation problem is controlled efficiently and the advance reservation is addressed explicitly by considering time span as well as resource availability.

In addition to the notations we introduced in Section 5.1, the following notations will be adopted in later sections:

- S_i, where $i \geq 1$, represents a physical server which is described by standard resources such as number of CPUs, gigabyte(s) of memory and hard disk.
- t_i, where $i \geq 1$, specifies a point in (one-dimensional) time. Its unit could be either coarse-grained (e.g., a day) or fine-grained (e.g., an hour).
- p_i, where $i \geq 1$, is a point that represents a time interval in computational geometry context.
- Fr_i, where $i \geq 1$, it is a fragment in a reservation histogram [77]. It can be mapped to a point p_i in the computational geometry context.
- R_i, where $i \geq 1$, is a request from a customer. It is a *tentative* point in the computational geometry context. If an agreement is finally reached by both parties, the tentative request point will be distributed into at least one or maximally all the existing points of the plane. Meanwhile, 0, 1 or 2 more new point(s) will be deduced during distribution.
- Basic Unit (BU): we assume that a resource's capacity may be provisioned in finite multiplies of a basic unit capacity. For example, if server S_1 is equipped with four Quad-Core Intel Xeon processors, 24 GB memory and 2400 GB hard disk, a basic unit capacity could be configured at 1 core, 1.5 GB memory and 150 GB hard disk. In this case, S_1 can provide up to 16 BUs. In reality, the service providers could customize the BU according to their scenarios.

5.4.1 Computational Geometry Representation

Figures 5.5 (a), 5.6 (a) and 5.7 (a) represent reservation histogram scheduling. Several requests from customers are scheduled on S_1, where x-coordinate indicates time t_i and y-coordinate indicates quantification of S_1. Therefore, customers can customize the VM configuration by setting the number of BU according to their preferences.

Our approach extends the concepts and applications from computational geometry [68, 78] to represent the time intervals corresponding to periods as points on a plane, as illustrated in Figures 5.5 (b), 5.6 (b) and 5.7 (b). The x-coordinate indicates ending time and the y-coordinate indicates starting time. Since the ending time of a service is greater than the starting time, all points will always be above the diagonal. The whole server is represented as a plane, in which lie points and segments. Each

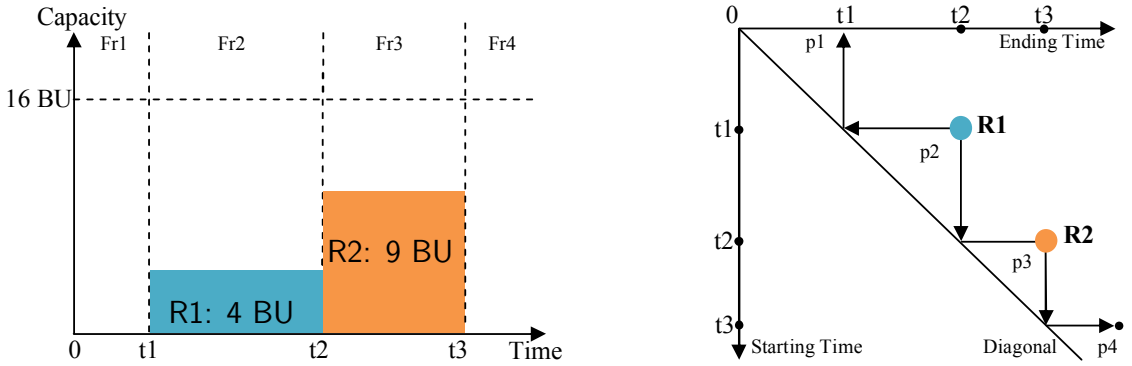


Figure 5.5: Representation of a schedule with two reservations as a histogram (a) and using computational geometry (b).

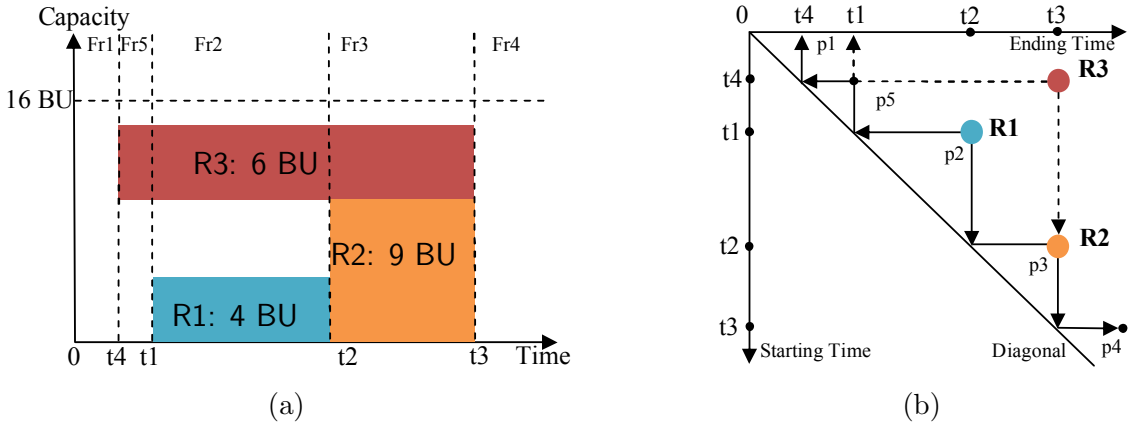


Figure 5.6: Representation of a schedule with three reservations as a histogram (a) and using computational geometry (b).

point indicates a period of time over which the server has a consistent and continuous available resource configuration. The configuration information is dynamically updated according with the new requests. The more the points in the plane, the higher is the degree of the server's partitioning. Therefore, fragmentation means that a server contains a number of time slices throughout its reservation time and the virtual resource configuration in each time slice differs from its neighboring slices.

In Figure 5.5 (a), a customer sends request R_1 for 4 BUs as VM configuration in a time span from t_1 to t_2. And S_1 is capable of satisfying R_1. Meanwhile, S_1 is partitioned to be three fragments with respective available resource quantities: fragment Fr_1 with 16 BUs, Fr_2 with 12 BUs and Fr_3 with 16 BUs. We can map the reservation histogram in Figure 5.5 (a) to a computational geometry representation in Figure 5.5 (b). There, R_1 is a tentative point when two parties reach an agreement and is finally merged to be p_1, p_2 and p_3. Then, in a time span from t_2 to t_3, R_2

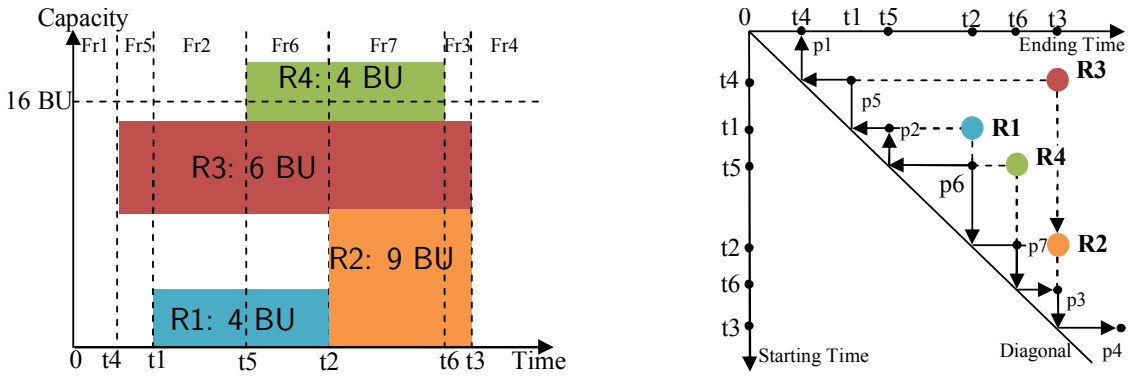


Figure 5.7: Representation of a schedule with three already scheduled reservations and a requested reservation as a histogram (a) and using computational geometry (b).

arrives for 9 BUs as VM configuration and is handled successfully by S_1. This leads to an update of fragment Fr_3 with available resources of 7 BUs and a new fragment Fr_4 with 16 BUs. Consequently, in Figure 5.5 (b), a new point p_4 is created through R_2. Applying this scheme to request R_3 in Figures 5.6 (a) and (b) leads to the new point p_5.

Figures 5.5, 5.6 and 5.7 illustrate how these simple VM requests can be mapped into a computational geometry context. A point on the plane represents a one-dimensional attribute, namely the number of available BU for one VM. However, the service provider would best allow customers with flexibility, adaptability and choice [10]. For IaaS, customers are free to customize the VM configuration. Thus, different VM configurations from different requests cannot be simply quantified to a coarse grained composition of BU and such a reservation histogram scheduling becomes inadequate.

Our approach applies not only to the above situation but also to flexible VM configuration, because an accepted request will dynamically impact its related points, inside which the server information at that specific time interval will also be updated respectively. In this case, a point on the plane represents three-dimensional attributes, namely the available cores, memory and hard disk sizes. Therefore, in our experimental part in Section 5.5.2, SLA requests are generated randomly with various VM configurations.

5.4.2 Service and Resource Availability

The term "available" describes a system that provides a specific level of service as needed. This idea of availability is part of everyday thinking [44]. In IaaS, effective availability management can directly impact customer satisfaction and determines the reputation of service providers [10]. It is therefore necessary to ensure that the service provider delivers the right levels of QoS (e.g., service availability) to satisfy its customers in parallel with its own business objectives.

Service availability is the ability of an IT service or component to perform its required function at a stated instant or over a stated period of time [10]. Specifically, any loss of service, whether planned or unplanned, is known as an outage. Downtime is the duration of an outage measured in units of time (e.g., minutes or hours). High availability implies a service level in which both planned and unplanned computer outages do not exceed a small stated value [44]. As a consequence availability is the probability that the service is up and running. When a service is consumed by a customer, the service provider has to ensure that the service availability is in accordance with the one defined in the SLA, in order to avoid the potential penalties due to SLA violations. In [10], the scope of service availability management covers planning, implementation, monitoring and adjustment of IT infrastructure availability, where planning of availability happens during the SLA negotiation phase. Before any Service Level Requirement (SLR) is accepted and ultimately the SLR or SLA is agreed between the business and the IT organization, it is essential that the availability requirements of the business are analyzed to assess if and how the IT infrastructure can deliver the required levels.

During IaaS SLA negotiation, a user's service availability request should be translated into corresponding service provider's internal technical terms. One such term is *resource availability*, which means the percentage of time that infrastructure resources are available through the entire service life cycle. In [48], authors outline that the service availability guaranteed by three large cloud providers (Amazon, Google and Rackspace Cloud) is more than 99.9% in order to obtain good reputation in today's competitive market. Therefore, as a service provider, we propose to provide 99.9% service availability as well. For that we have to ensure that the resource availability of required service lifecycle is 100% during the negotiation and planning phase of service availability management. Our method is able to check the resource availability of SLR in advance. In Figures 5.5 (b) and 5.6 (b), all the requests can be satisfied with 100% resource availability, whereas in Figure 5.7 (a)/(b), when R4 requests 4 BUs as VM configuration in a time span from t_5 to t_6, the service provider lacks resources

from t_2 to t_6. For managing this mismatch between the requirement and the free capacity, there are five possible reactions:

- The service provider can simply reject the request.
- For the existing resource capacity, an attractive price can provided.
- Outsourcing to third parties for extending local resources can take place.
- Suspending and resuming other service(s) in some rational manner [65].
- Finding alternative solution(s) considered close to the original requested time interval.

Here, we focus on the last approach. It is always nice to have renegotiation, when the service providers lack resources during SLA negotiation. The customers could resubmit the SLA request with the same or new functional and non-functional properties. Instead of direct rejection, the service providers can provide counter-offer with more options, therefore, the customers could be guided to choose or make right requests based on their preferences. In this case, the Function d^i in Equation 5.1 considers the absolute deviation between original request and the counter-offer, which leads to a price reduction. Thus, the SLA negotiation becomes more customer-oriented. In computational geometry context, finding an alternative solution is equal to finding a proper location for R_4 in Figure 5.8. The alternative solution planner moves the request point R_4 on the track of the segment which goes through the center of R_4 and also parallel to the diagonal, because all points on that track always have the identical time interval as R_4, which means that the service providers are able to ensure that the customers can use the VM with exactly the same duration as they expect. We name such kind of segment as the *target segment* of the request point. R_4 keeps moving on the target segment in two directions one after another until the server is able to provision the service with resource availability of 100%. Then the service provider will send this counter-offer to the customer. The searching scope is controlled within certain time units. If the service provider fails to find an alternative solution within searching constraint, the service provider could outsource to other service providers for extending its resource capacity temporally.

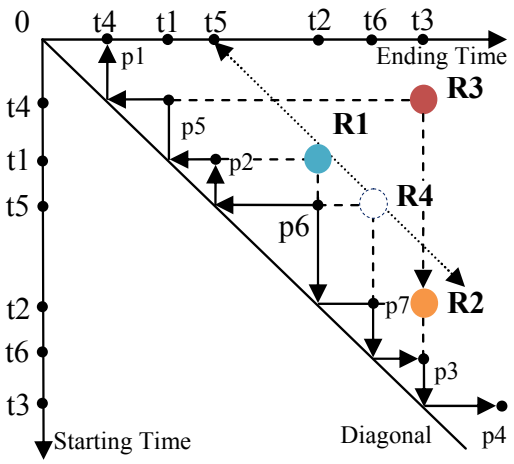


Figure 5.8: Finding alternative solutions by moving the request point. The moving is indicated by diagonal arrows starting at **R4** pointing towards the upper left and the lower right, respectively (compare with Fig. 5.7(b)).

5.4.3 Virtual Fragments of Server

In Section 5.4.2, we propose a strategy for modeling and controlling resource availability. By moving the request point the resource unavailability problem can be resolved. However, this might not be an optimal solution.

From a computational geometry representation, the number of points reflects the degree of fragmentation. The more points introduced, the more complicated a plane has to be maintained. Furthermore, the increased fragmentation reduces the potential scheduling opportunities and results in lower utilization [66]. Rarely used points must be eliminated at the very beginning.

We strive to merge the request point with available points and so as to reduce the number of fragments or at least keep the fragments unchanged.

First, the alternative solution planner searches for points p_i on the target segment of the request point within limited scope (e.g., $\pm$ 20 time units). If there exist such points and one or more of them satisfy the requirements for VMs, the best one is chosen according to selection criteria, like the shortest distance, optimal profit amongst others. If there is no such point, that is, R_4 does not fit into any available point, the alternative solution planner looks for some point that *contains* R_4. If not, the planner will check if some point(s) is(are) *contained* by R_4. We say that $p_x = (t_1, t_2)$ completely contains $p_y = (t_3, t_4)$ if and only if $t_1 \leq t_3$ and $t_2 \geq t_4$. If any of above conditions (contains or contained) holds, we found candidate points for the request. If the available capacity at all of these points is larger than or equal to the requested capacity, we can merge R_4 into the chosen point(s) (p_2 and p_3 in Figure 5.8). Possibly

longer than requested durations may be compensated with an attractive price if the total requested duration is slightly longer than the chosen point(s)'s duration or with a preferential activity that customer can use the VMs longer than expected if the total requested duration is slightly shorter than the chosen point(s)'s duration.

From long term perspective, reservations with too many fragments will lead to poor resource utilization and lower profits. Sometimes providing counter-offer or even rejection is wiser than satisfying the original request. For each server with specific hardware configuration, we propose a *upper bound for its fragmentation*. Namely, a server with maximal number of fragments cannot satisfy the coming SLA request by introducing new fragment. As we mentioned, a counter-offer, not leading to new fragment, can be provided to the customers. Normally, the higher capability the server has, the higher the fragmentation upper bound could be. Finally, the expired reservations (points) will be removed from the context periodically, thus the degree of the fragmentation can be reduced and controlled.

5.5 Implementation and Experimental Verification

5.5.1 Architecture and Implementation

This work is treated as an implementation of Generic SLA Manager (GSLAM) in IaaS scenario, which is applied in project SLA@SOI. The GSLAM provides a generic architecture that can be used across different domains and use cases to manage the entire SLA lifecycle, including activities such as SLAs modeling [34], negotiation, provisioning, resources outsourcing, monitoring and adjustment [79]. I name my work as Infrastructure Planning and Optimization Component (IPOC). According to the design, the IPOC receives requests for infrastructure, queries the Infrastructure Service Manager (ISM) for potential provisioning solutions, selects and reserves the optimal one and requests the Infrastructure Provisioning and Adjustment Component (IPAC) to provision the selected plan as appropriate. If local resources cannot satisfy the request (e.g., due to lack of availability or specification discrepancies), the IPOC attempts to outsource from third party providers.

In Figure B.3 of Section B.3 in Appendix., a class diagram is given, where resources can be either CPUs or memory or storage with different properties. Each request contains all the infrastructure resources information for the incoming SLA requests, like CPU configuration and number, memory size, storage, location, image of the VM, QoS terms amongst others. A site (provider) is represented by a request processor that is an instance of the algorithm's implementation as part of an IPOC's functionality.

It handles one or more requests and interacts with the ISM for querying and reserving the resources. Based on the information within the request, the site minimizes inner implementation and outsourcing costs for reasons of competitiveness, while respecting business policies for profit and risk. An algorithm for outsourcing is designed and implemented, using cost and subcontractor reputation as selection criteria; and local resource configurations as a constraint satisfaction problem for acceptable profit and failure risks. Thus, it becomes possible to provide educated price quotes to customers and establish safe electronic contracts automatically.

In Figure B.4 of Section B.4 in Appendix. represents a class diagram for advanvced reservation. We outline how the context plane is constructed and maintained. <> represents the segment between starting point and ending point. <<RequestPoint>> represents the detailed information about the VMs, it is a tentative point in the context at the very beginning. <<ReservationPoint>> is a reserved point, which inherits all the features from <<RequestPoint>> and represents a valid service that will be delivered in the future to the customer. Both points contain a <<VirtualMachineConfig>>. The logic part of the implementation is <<ResourceManagementPlane>>, it could add and delete the point and (or) segment. When the context mismatches the request from the customer with its own resource pool, it tries to move the point and get the alternative solutions.

5.5.2 Experimental Verification

5.5.2.1 Experimental Environment

To evaluate the model with regard to its validity, we establish a simulation scenario with specific functions for increased quality costs, profit, failure probability, etc. The simulation scenario is based on IT & Medien Centrum (ITMC) Linux High Performance Computing (HPC) clusters at the Dortmund University of Technology. Resources under negotiation are VMs with CPU cores, memory and storage. There are 4 types of servers and Table 5.1 illustrates the infrastructure capacity of the main site. The overall hardware infrastructure is: 10 $\times S_1$, 10 $\times S_2$, 10 $\times S_3$, 10 $\times S_4$. Three subcontractors have CPU cores of 610, 650 and 950 individually.

CPU cores are offered in 4 different combinations of clock speed (1 GHz or 2 GHz) and volatile memory (1 GB or 2 GB). Memory and storage are offered in arbitrary quantities, in increments of 1 GB. CPU core prices are given by Table 5.2 in the form of normal distributions (identified by the mean value and variance). Similarly, the price for storage is 2 ± 0.5 per GB. In each simulation run, each provider is

assigned resource price values at random, from these distributions. Their negotiated qualitative characteristics are availability, measured as a percentage of time, and isolation, indicating that the customer's virtual resources have exclusive access to the physical infrastructure that implements them (e.g., a blade server).

Table 5.1: Main site capacity

Server	CPU	Memory	Storage
S_1	16	24 GB	500 GB
S_2	12	16 GB	500 GB
S_3	16	64 GB	500 GB
S_4	16	24 GB	500 GB

Table 5.2: CPU core prices (€)

	1GHz	2GHz
1Gb	50 ± 5	100 ± 10
2Gb	100 ± 10	150 ± 15

Price reductions are given as a stepwise function ranging between 0% and 20%, in steps of 5% at resource quantities 15, 50, 150 and 500. For resource reservation 10% discount is offered and the price reductions due to the deviation of counter-offer in steps of 5% at resource quantities 10, 20 and 30 time units. We do not distinguish between the type of resources, rather we apply the same function both to CPU, memory and storage. The increased costs for higher quality (i.e. function f^i) are a 50% additional cost for isolation (which can only be *true* or *false*), and 10% for each additional unit of availability. Baseline availability is 95%, and maximum is 99%. Default value for isolation is *false*. We use the same function for both resource types. Profit function g^i is given for both resource types as $g^i = 0.3 \cdot C_I + 0.05 \cdot C_E$. That is, the provider also makes a very small profit from outsourced resources. Therefore, price quotes to the customer are provided as $g(C_I, C_E) + C_I + C_E$, minus applicable price reductions.

An initial (artificial) SLA past failure rate is selected to be 20%. Random SLA violations are introduced, in a rate always in accordance with the site's failure rate

for each resource. Failure frequency is then further controlled by the selected values for β; each time we choose such a value, we modify the failure rate to reflect these extra measures we take to safeguard the SLAs. The minimum profit F^* depends on the customer. We have three customer classes; namely, *Gold*, *Silver* and *Bronze*. For Gold customers, F^* is $0.7 \cdot g(C_I, C_E)$; for Silver it is $0.8 \cdot g(C_I, C_E)$; and for Bronze it is $0.9 \cdot g(C_I, C_E)$. We are allowing additional resources (and hence, lower profit) even for bronze customers, as we wish to improve the reputation of the provider. Our target is to reach a failure rate equal to 5% or less, given the small gradual effect of β on the future violations.

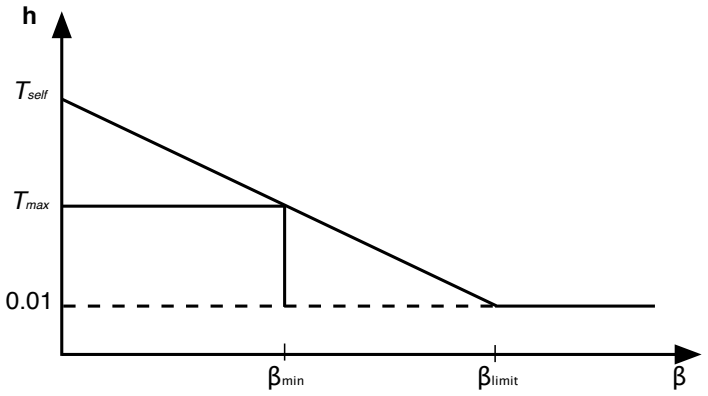


Figure 5.9: Function h in relation to β values

Function h^i takes values between the maximum (statistical) failure rate T^i_{self} and a very small value (chosen to be 10^{-2}), which becomes effective when β^i reaches a value of 0.2 (Figure 5.9). In other words, values of β^i larger than 0.2 (20% more than the normal implementation costs) make no difference to the probability of failure. Finally, we examine the case of four connected providers, one of which is the "main" site, and the other three are subcontractors.

Equation 5.18 can be transformed to be Equation 5.24.

$$\beta^i_{max} \leq \frac{F^i - F^i \times F^*}{C^i} \tag{5.24}$$

We can further transform the Equation 5.19 as:

$$\beta^i_{min} \geq \frac{(T^i_{self} - T^i_{max}) \times 0.2}{T^i_{self} - T^i_{min}} \tag{5.25}$$

5.5.2.2 Generation of Workloads for Advance Reservation

In order to evaluate our advance reservation model, we explicitly setup a pure advance reservation scenario with specific functions to increase the resource utilization by searching alternative solutions and the number of satisfied requests. Thereby, we studied the active VMs of ITMC clusters and generated the workloads by using the following approach:

- There were 3 types of testing workload: the workload with 800 requests, with 1200 requests and with 1600 requests. During SLA negotiation customers can request from 1 to 8 CPU cores, 512 Megabytes (MB) to 16 GB memory and 20 GB to 400 GB storage.

- We assumed there are 365 time units (one day per unit) when the customers may make reservations for VMs within one year. The resource lease period ranges between one week (7 days) and two months (60 days).

- The probability of selecting a certain number of CPU core(s) in the active VMs of ITMC clusters is consistent with Gaussian distribution,

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2}$$

  where $1 \leq x \leq 8$, expected value $\mu = 2$ and variance $\sigma^2 = 1.9$, which means most of the customers prefer 2 cores. The memory and storage are regular and often multiples of number of CPU core(s). They are also consistent with Gaussian distributions. For memory, $0.5 \leq x \leq 8$, expected value $\mu = 2$ and variance $\sigma^2 = 1.9$; For storage, $10 \leq x \leq 50$, expected value $\mu = 20$ and variance $\sigma^2 = 10$.

- As explained in Section 5.4.2, if a service provider lacks resources, the planner has to search an alternative solution. The searching scopes are $\pm$ 0, $\pm$ 10, $\pm$ 20 and $\pm$ 30 time units on the target segment, where $\pm$ 0 means there is no flexibility for searching alternative solutions.

- In order to limit the number of fragments, in our experiment, we set $\mp$ 2 time units as compensation or as free use time. And this should be specified by service provider in reality.

- During SLA negotiation, a customer can send multiple tentative requests to several service providers for selecting the optimal one. As outlined in Section 5.1.1,

by investing additional cost, service provider can not only quote an attractive price but also keep service failure probability as low as possible. Therefore, we suppose that the customer tends to accept our offer to the original request with acceptance rate of 80%, whereas the acceptance rate to an alternative offer floats between 50% and 75%, depending on the degree of shifting steps.

5.5.2.3 Simulation Results

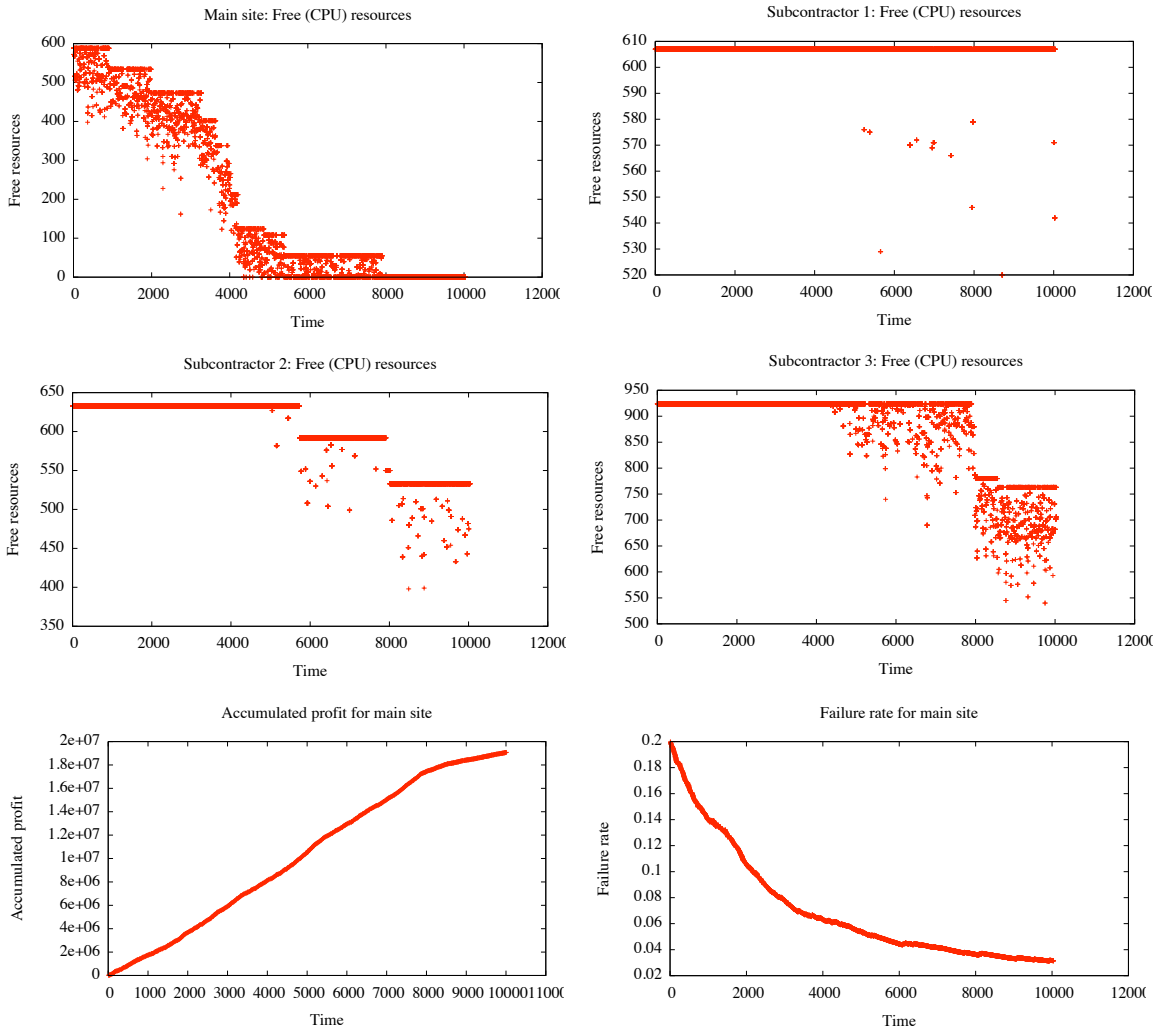


Figure 5.10: Experiment results

For generic SLA business model, Figure 5.10 illustrates the most important results of the simulation. The top four plots show the available resources over time, for the main site and the three subcontractors. We can see that as soon as the main site's resources become depleted, it is mostly the third subcontractor that is being utilized,

as it is the optimal from a price and failure rate point of view. Following, the second and the first are utilized when the third has no more available resources. The bottom two graphs illustrate the accumulated profit over time for the main site, and the values for its failure rate. We can see that the profit keeps increasing even when there are few resources available and the site has to outsource. Also, that the site's reputation (failure rate) is improving significantly over time, starting with an artificial failure rate of 20%, but eventually reaching and surpassing the target of 5%. This decrease of failures was simulated by modifying the initial probability of a failure event, according to the values β was taking over the simulation time.

For pure advance reservation model, requests array [800 requests, 1200 requests, 1600 requests] is evaluated in combination with a searching scope array [± 0, ± 10, ± 20, ± 30], therefore there are in total 12 combinations. The result for each combination is a satisfaction rate. In Table 5.3, we can see that the wider the searching scope available to the alternative solution planner, the more requests that can be satisfied. For 800 requests, the service provider can handle almost all the requests

Table 5.3: Satisfaction ratio of requests with different shifting steps

	Limit on shifting requests in time units			
Workload	± 0	± 10	± 20	± 30
800 requests	95.38%	99.38%	100%	100%
1200 requests	70.42%	86.25%	89.58%	93.67%
1600 requests	55.19%	70.12%	72.06%	72.88%

(95.38%) without finding alternative solutions. For 1600 requests, not every incoming request can be satisfied. For 1200 requests, by searching alternative solutions, almost 90% requests can be satisfied, which is quite close to service provider's capability.

Figure 5.11 is a summary of fragment statistics of all the servers in [1200 requests, ± 10] without controlling fragmentation (red points) and with controlling fragmentation (green points). By controlling fragmentation, there are in average 36 fragments in each server, which are 10 fragments less than the approach without considering fragmentation. Figure 5.12 is a summary of resources utilization of all the servers in [1200 requests, ± 0] and [1200 requests, ± 20] scenarios. We can see that by searching alternative solutions on one hand, significantly more customers requests can be satisfied (19.16% more), on the other hand, service provider has higher utilization of resources.

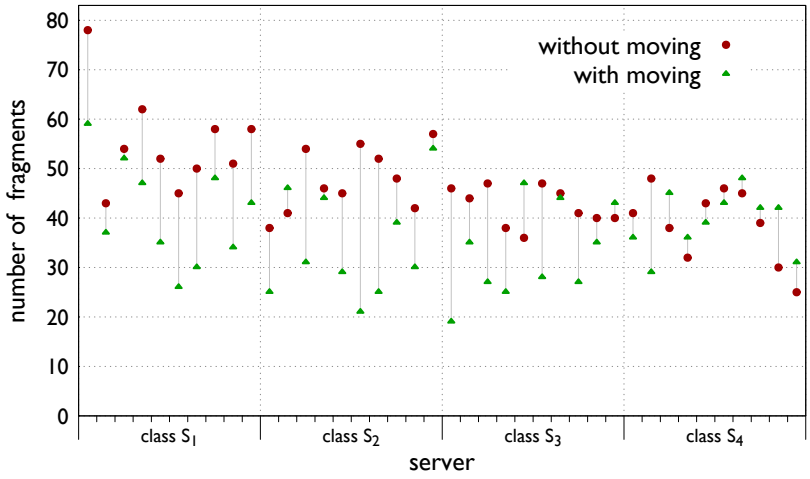


Figure 5.11: Fragment statistic

For example for CPU cores in (a), $(79.55\% - 70.30\%) \times (320\ cores \times 365\ days) = 10804$ core * day. 10804 means that 29.6 more cores can be rented to customers throughout the entire year.

5.6 Discussion

We develope a model for IaaS providers, based on which they can connect resource planning to high-level business decisions, using SLAs as a formalization tool. Our aim is to compute minimum implementation costs as part of price quotations towards customers, in order to remain competitive. Meanwhile, we use profit and SLA violation probability constraints to decide whether the problem can be satisfied at all, and what is the decision space based on which implementation costs can be calculated. Outsourcing via subcontracts is included as part of the decision process, to achieve additional profit but also to sustain customers when local resources are not sufficient.

Besides, a reservation model for resource planning in the context of an IaaS cloud provider is introduced in this chapter. By using computational geometry, a service provider can verify, record and manage the infrastructure resources efficiently during the SLA negotiation and planning phase. Specifically, the service provider can determine the QoS (resource availability and time span) and thereby the feasibility for satisfying the customers' requests can be measured. Alternative solution(s) can be generated as counter offer(s), when the service provider lacks resources. Therefore the provider's reputation can be improved by enhancing his ability to satisfy as many customers as possible leading to higher resource utilization and consequently

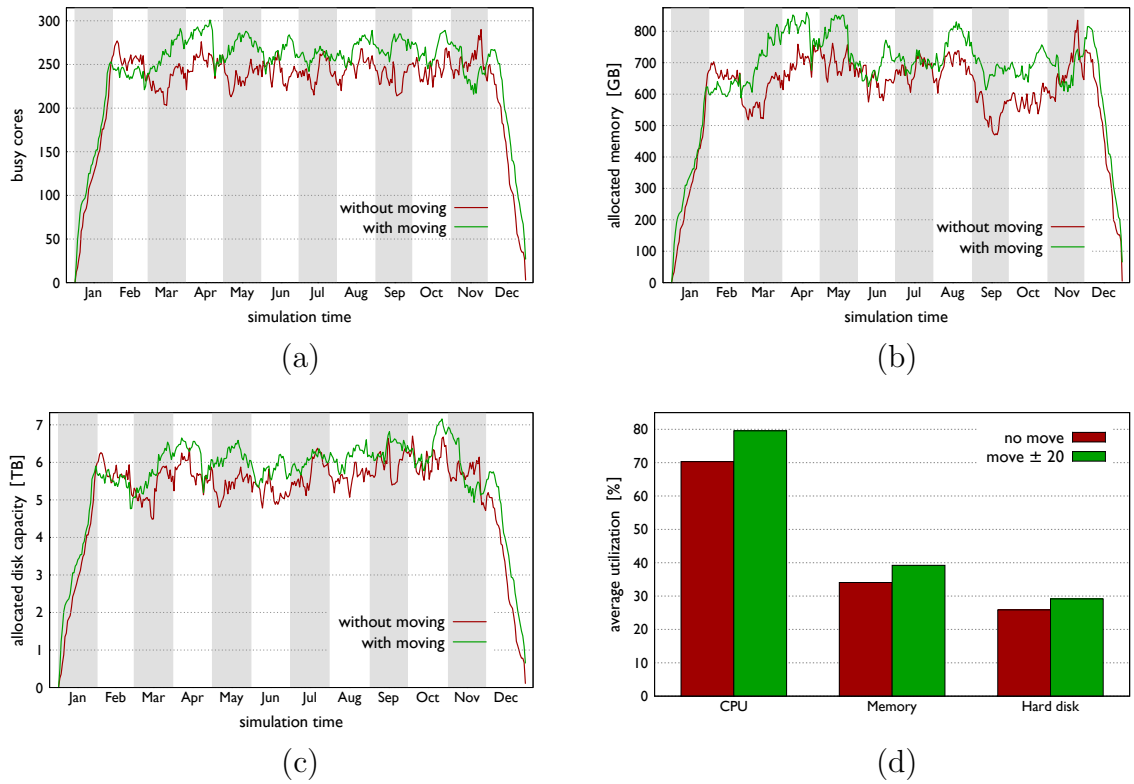


Figure 5.12: Resources utilization of all servers

higher profits. Finally, our simulations prove that the approaches are feasible and reasonable, yielding useful results given the scenario that we chose to implement.

From an optimal SLA representation structure to a feasible SLA offer, finally the SLA negotiation reaches an agreement for both parties. Afterwards, the agreed service will be provisioned accordingly. Having a delivered SLA, the SLA management architecture is responsible for defining and deriving actions (e.g., monitoring, adjusting the services) for getting customers into regular operations during SLA operation phase. Moreover, the SLA management triggers communication with internal and external stakeholders while at the same time safeguarding customer satisfaction [8] in order to avoid the violation, which is still an active area of research in cloud computing. Especially, as we have already derived in Section 1.2 and 2.5, the aforementioned concerns could be translated to be a resource management for multi-domain SLA problem, which is explained more in details in next chapter.

Chapter 6

Multi-Domain Resource Management During SLA Operation

6.1 Introduction

Virtualization technologies enable the live migration of running VMs to achieve load balancing, fault-tolerance and hardware consolidation in data centers. Regarding VM live migration, there are certain approaches already widely utilized. Relying on shared storage, the process of live migration is reduced to only copying memory state and CPU registers from the source host to destinations [80], without transferring the whole VM. In contrast to the pure stop-and-copy strategy of offline VM migration, live migration fine-tunes the migration into several rounds of pre-copying and one last round of stop-and-copy with much less downtime. Nevertheless, based on the impact of VM migration in [81], there are still issues to address. Firstly, the more rounds of iteration in the pre-copying phase, the less data needed to be transferred in the final stop-and-copy phase, and less the downtime. However, the shortened downtime is acquired at the expense of a longer total migration time, which leads to significant adverse influences on the service performance, system load and network [40]. Secondly, the downtime cannot be fully eliminated, because many workloads typically include some memory pages that are updated frequently, named Writable Working Sets (WWS) [80]. Thus, it is wise not to maintain it until the last phase. Moreover, in stop-and-copy, the performance of VM live migration is affected by many factors, such as workload type, hypervisor type amongst others [39].

As such, zero visible downtime is merely a virtual ideal goal. The number of migrations should be controlled, as it has impact on the performance of the other

running VMs in the source and destination hosts, which is mainly proportional to the amount of memory allocated to the VM. Here, we hypothesize that the VM performance violation happens, when the VM experiences 100% CPU utilization. And the performance degradation happens during the VM migrations stage.

For VM consolidation reported in [37] [38] [82], the authors mainly discussed:

- Resource management, either in simulation or some prototype implementations within the common cloud middleware. No SLA management is introduced.
- Using either artificial workloads or partial historical information from the workloads in various projects.
- Using VM live migration to leverage the consolidation and service performance; however, the migration was misused without carefully taking the service availability into consideration.

Therefore, in this chapter, our contributions are the follows:

- A proof-of-concept prototype implementation of OpenStack SLA management (IaaS), which aims to be connected with the PaaS and SaaS layers by providing SLA lifecycle, service customization and automatic scalability.
- By using the workloads information at GWDG, our resource allocation strategies are compared with others existing counterparts in several aspects. Specifically in SLA operation phase, additional resources could be reallocated into the service in order to resolve the violations. Moreover, availability-oriented VM allocation strategies that control VM live migration and leverage the objectives between the service provider and the customer.
- The influence of SLA chain penalties on the multi-domains (i.e. SaaS, PaaS and IaaS) is introduced into the VM consolidation. In Section 6.2, we clarify the manner in which they are mutually influenced in terms of economical aspects during the SLA violation.

6.2 Modeling of SLA Pricing and Penalty

Many works discuss SLA violations, however, modeling the consequences of the violations is rarely seen, namely, SLA penalties. Currently, although some approaches describe penalties [27] [28] [29], they do not satisfy all of the following requirements for formulating complex penalty expressions in a single unambiguous model:

- Presenting the chain effect on penalties among multi-domains due to the violations in the IaaS layer.
- Full flexibility regarding QoS levels concurred agreed and/or achieved, without being constrained (e.g., by pre-specified classes of service).
- Openness and applicability to different domains, without dependence on specific languages or taxonomies.

According to Chapter 5, IaaS service providers are able to compute the minimum implementation costs as part of the price quotations for the customers, in order to remain their competitiveness. Meanwhile, profit and SLA violation probability constraints are used to make a judgment on whether the problems can be satisfied at all, and what is the decision space based on which implementation costs can be calculated. Furthermore, outsourcing via subcontracts was included as a part of the decision process, to achieve additional profits, while still sustain customers when their local resources are not sufficient. Therefore, let us assume an IaaS service i, and an SLA that governs consumption of this service by a certain customer. We have:

$$C^i = C^i_{Imple} + Pr^i \tag{6.1}$$

$$C^i_{Imple} = C^i_I + C^i_E \tag{6.2}$$

$$C^i_I = C^i_{Energy} + C^i_{Utility} \tag{6.3}$$

where the cost C^i of service i is the sum of internal cost C^i_I (i.e. internally utilized resources, energy cost) and external cost C^i_E (i.e. sub-contracted resources) as well as profit Pr^i.

In PaaS, a container includes a set of resources that allow users to run their applications. By delivering such a computing platform (e.g., operating system and program execution environment), many containers can be processed simultaneously on one VM (see Figure 6.1). We assume that on each VM there are n containers.

Therefore, the cost of each PaaS service p is:

$$C^p = \frac{C^i}{n} + Pr^p \tag{6.4}$$

SaaS developers can implement and deploy their applications on a cloud platform (e.g., a container) without the cost and complexity of buying and managing the

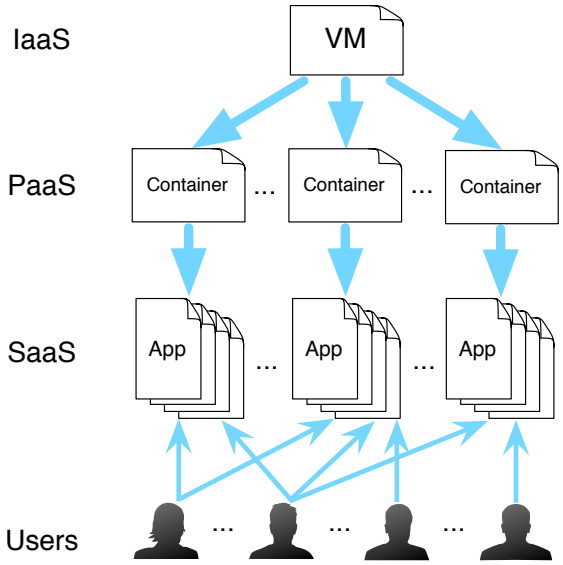


Figure 6.1: Hierarchical structure of the services in different domains

underlying hardware and software layers. Similarly, we also make an assumption that m applications are allocated on each container. Then, the cost of a SaaS service s is defined as following:

$$C^s = \frac{C^p}{m} + Pr^s \tag{6.5}$$

C^p and C^s are valid only under the condition that the payment has no extra implementation costs except the infrastructure and platform environment for service execution.

Meanwhile, an SLA should also contain a set of penalty clauses specifying the responsibilities in case the service providers fail to deliver the pre-agreed QoS terms. Thus, we will use a variation of our previously described penalty model [83] as outlined in Equation 6.6:

$$R_x = \sum_k GW_k \cdot VR_k \tag{6.6}$$

R_x is the penalty ratio associated with the cost of service x, where GW_k is the weight of one specific guarantee being violated, for this specific combination of guarantees. This value can be arbitrarily high. It allows the negotiating customer to express the importance of honoring certain guarantees in this penalty function. VR_k is the violation ratio: the relationship between the achieved quality and planned quality. It indicates the degree of the offset of the quality from the agreed quality of a specific service parameter. Therefore, the penalty of IaaS service i is:

$$P_i^s(QoS_1^s, ..., QoS_t^s) = C^i \cdot R_i \tag{6.7}$$

The penalty of all PaaS services p is:

$$n \cdot P_p^s(QoS_1^s, ..., QoS_t^s) = n \cdot C^p \cdot R_p \tag{6.8}$$

By applying Equation 6.4, we have:

$$n \cdot C^p \cdot R_p = (C^i + n \cdot Pr^p) \cdot R_p \tag{6.9}$$

The penalty of all SaaS services s is:

$$m \cdot n \cdot P_s^s(QoS_1^s, ..., QoS_t^s) = m \cdot n \cdot C^s \cdot R_s \tag{6.10}$$

By applying Equation 6.5, we have:

$$m \cdot n \cdot C^s \cdot R_s = (C^i + n \cdot Pr^p + n \cdot m \cdot Pr^s) \cdot R_s \tag{6.11}$$

The violation of some QoS terms on the IaaS layer will automatically affect the other domains. For example, unavailability of a VM will undoubtedly render its inner PaaS and SaaS services to be unavailable. For all these QoS terms in the three layers, we have $R_i = R_p = R_s = R$. Thus, the extra penalties of the PaaS layer comparing with the IaaS layer is:

$$(6.9) - (6.7) = n \cdot Pr^p \cdot R \tag{6.12}$$

Similarly, the extra penalties of SaaS layer comparing with IaaS layer is:

$$(6.11) - (6.7) = (n \cdot Pr^p + m \cdot n \cdot Pr^s) \cdot R \tag{6.13}$$

Hence, a slight availability violation in IaaS will lead to exponential influences on its associated domains (PaaS and SaaS). An IaaS service provider, in order to be compliant with the SLAs, has to make optimal reaction and adjustment while the service is running. We can further adopt the above SLA penalty model into the ROI Equation 5.13 in Section 5.1.

6.3 Resource Rearrangement

As we introduced in Section 5.1.5, if there is an intersection for Equation 5.18 and 5.19, we name the intersection area as floating area, in which the service provider can offer the service with acceptable failure rate and reasonable profit.

According to our strategy, during the SLA negotiation phase, service provider selects the value of β_{min} for the sake of maximum profit and acceptable failure rate. However, we allow additional resources (and hence, lower profit) for customers, as the

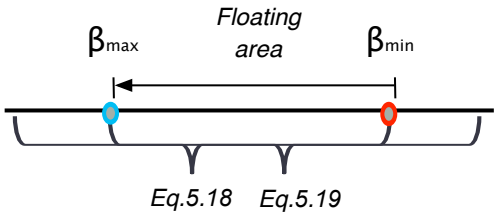


Figure 6.2: Invest additional resource to the service by increasing β in floating area

service provider desires to improve the reputation. More specifically, when the VM service is violated or about to be violated, for instance, extra CPU Millions Instructions Per Second (MIPS) is allocated to the VM in order to resolve the violation. This process is equal to moving the value of β from β_{min} to β_{max} in the floating area (Figure 6.2).

6.4 Resource Selection and Allocation

The energy consumption can be settled by the resource management system deployed in the infrastructure and the efficiency of applications running in the system. It has been shown [84] [85] that an active server with very low CPU utilization consumes between 50 to 70 percent of the power that it consumes when the CPU is fully utilized. To improve the utilization of data center, VM consolidation has been applied and proved to be efficient. This process supports live migrations of VM(s) with dynamic workloads to reduce the number of active physical hosts. Namely, when a host is under-loaded, in order to eliminate the static power, this host will be switched to sleep mode. Consequently, the energy consumption will be reduced sufficiently. On the contrary, when the resource demand increases, which could be the case that a host becomes over-loaded, one or more VMs will be selected and migrated into other host(s) and thus idle hosts will be reactivated again to provide service if necessary.

Thus, during VM live migration, both VM consolidation and service performance can be coordinated. Nevertheless, short downtimes of service migration are unavoidable due to the overheads. Hence, the respective service interruptions in IaaS reduce the overall service availability, which could also be the main cause of the chain effect on the penalties between domains. Here, the term downtime is used to refer to periods when a service is unavailable and fails to provide or perform its primary functions to the customers. The downtime can be further classified as planned and unplanned. Since unplanned downtime, e.g., failure of the system, is complicated and uncertain in a simulation environment, in our work, we only consider the planned downtime for

Algorithm 3 VM Selection-AV

Input: req_availability
Output: selectedVM

```
migratableVms ← getMigratableVms()
totalTime ← 86400 // 24 hours in seconds
vmSize ← migratableVms.getSize()
vm, downtime, totalDowntime, availability ← NULL
sortByAvailability(migratableVms) // descending order
for v ← 1 to vmSize do
  vm ← migratableVms[i]
  downtime ← vm.downTimeEstimator()
  totalDowntime ← vm.totalDowntime
  availability ← 1-(totalDowntime+downtime)/totalTime
  if availability>req_availability then
    selectedVM ← vm
    break
  else
    continue
  end if
end for
selectedVM.updateDownTime()
return selectedVM
```

evaluating the service availability. The downtime that is introduced by VM live migration is a kind of planned downtime. Thus, the availability is formulated as below:

$$Service\ Availability = \frac{T_a}{T_a + T_b} \times 100 \tag{6.14}$$

where T_a is service uptime and T_b is service downtime.

We are focusing on how to manage the number of live migrations so as to control availability according to the established SLA during resource allocation. The optimization of the resource allocation problem in a data center can be executed in two steps: initial selection of VMs that need to be migrated, then the chosen VMs will be placed on the hosts using a VM allocation algorithm.

6.4.1 VM Selection

In Algorithm 3, the input value is the requested availability of customer and the output value is the finally selected VM that will be migrated. Firstly, all the migratable VMs will be selected by removing the VMs that are already in migration. Thus, the selected VMs will be sorted in descending order of their current service availabili-

ties. Then, the availability of each VM in the sorted VMs list will be re-calculated to check whether the availability is still greater than the requested availability when the VM is being migrated. Finally, if such a VM can be found, then we will migrate it and accordingly update its downtime record of this VM. Otherwise, no VM will be migrated. The complexity of the selection part of algorithm is $O(n)$, n being the number of migratable VMs.

Algorithm 4 VM Allocation-AVL

```
Input: optimalHostUtility, selectedVM
Output: allocatedHost
  minimalDiff ← Double.MAX_VALUE
  hosts ← getHostList(); host, hostUtility, diff ← NULL
  for h ← 1 to hosts.size do
    host ← hosts[i]
    if excludedHosts.contains(host) then
      continue
    end if
    hostUtility ← host.getUtilizationOfCpu()
    if host.isSuitableForVm(selectedVM) then
      if hostUtility!=0 &
      overUtilizedAfterAllocated(host, selectedVM) then
        continue
      end if
      diff ← Math.abs(hostUtility-optimalHostUtility)
      if diff < minimalDiff then
        minimalDiff ← diff
        allocatedHost ← host
      end if
    end if
  end for
  return allocatedHost
```

6.4.2 VM Allocation

In Algorithm 4, the inputs are the optimal host utilities and the selected VMs, the outputs are the hosts to where each selected VM will be migrated. First of all, the over-loaded host(s) and the host(s), which is(are) going to be over-loaded after allocating the migrated VM, are not be considered. Then, a host, whose utility is the closest to the $optimalHostUtility$, will be selected. Here, the $optimalHostUtility$ is not a fixed quantity and will be discussed more in Section 6.6. We want to find the relationship between the utilization of the target allocation host and the value of QoS

terms. Specifically, by considering the service availability constraint, our goal is to allocate the VM to a host that induces the least increase of power consumption and service performance violation due to this allocation. The complexity of the allocation part of algorithm is $O(n)$, n being the number of hosts.

6.4.3 VM Live Migration Downtime Estimator

As discussed in Section 6.1, the overall duration and short downtime that are introduced by VM live migration are essential properties while implementing service availability in an SLA. In this section, we introduce a VM live migration downtime estimator into CloudSim [86]. As modeled in [39] and [40], using *migration bounds*, the downtime of VM live migration is defined in lower and upper bounds as follows:

$$mig\_overhead \leq t_d \leq mig\_overhead + \frac{VMSize}{LinkSpeed} \tag{6.15}$$

In order to better estimate the downtime value, the authors summarize four main contributing factors that affect the downtime, namely: available link speed, average page dirty rate, VM memory size and migration overheads. The link speed and page dirty rate are proportional to the access traffic of the server applications in a day. The access traffic experiences the highest capacity at noon and lowest capacity in the morning and late night. Therefore, we assume the probability of determining live migration downtime is consistent with the normal distribution as following:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\mu)^2/2\sigma^2} \tag{6.16}$$

where $0 \leq x \leq 24$ (hour), expected value $\mu = 12$ and variance $\sigma^2 = 1.9$, which means at 12 o'clock the server application usually reaches the highest access traffic. For instance, a VM loading server application workloads has 1024 MB memory and 1 VM per second migration link. As such, the lower and upper bounds of migration are around 314 Millisecond (ms) and 9497.8 ms respectively [40].

6.5 OpenStack SLA Management Framework

As we introduced in Section 5.5.1, an implementation of GSLAM in different scenario could lead to various implementations of ISM, where ISM is the upper level API of cloud infrastructure virtualization middleware, for instance, Tashi [87], OpenNenula [88] and most recently OpenStack [89], CloudStack [90] amongst others. In

2014, IDC predicts that virtually all infrastructure or management software vendors will continue to race toward the new models; a key step will be redeveloping or adapting offerings to run in an OpenStack, CloudStack, or other vendor neutral environment [5]. Thereby, an OpenStack-SLA Manager (SLAM) is presented (Figure 6.3). Relying on OpenStack Nova API Woorea [91] and JCloud [92], the GSLAM is able to implement its corresponding ISM [93] by:

- querying the status of infrastructure during the SLA negotiation, based on which the service provider is able to generate the corresponding offer / counter-offer for the customers;
- providing SLA terms (i.e. pricing, penalty, service availability and performance) monitoring mechanism together with OpenStack;
- deploying the VM allocation strategies within OpenStack Nova so as to maximize the profit and minimize the SLA violation as well as energy consumption;
- creating, customizing and deploying the agreed services;
- reconfiguring and removing the services on demand.

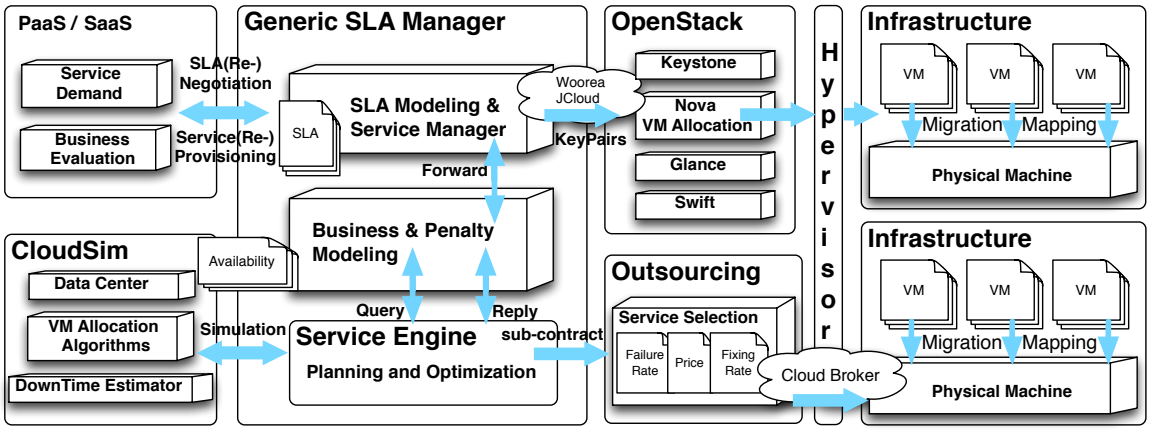


Figure 6.3: Integration of the generic SLA manager and OpenStack

PaaSage aims to deliver an open and integrated PaaS for different (e.g., industrial, e-science and public sector) use cases to support model-based lifecycle management of cloud applications. Using OpenShift, one of the most popular PaaS implementations, it could auto-scale its cloud PaaS framework for Java, Perl, PHP, Python and Ruby applications delivered in a shared-hosting model [94]. PaaS permits many applications offered by multiple development teams to co-exist on the same set of hosts in a safe

and reliable fashion. In addition, the platform offers a variety of opportunities for multi-tenant deployments. Thus, an application that is intended to work for a single organizational unit can also be deployed in such a manner that many organizations or end-users are able to use [95]. Therefore, end-users benefit from the application management (instead of VM level management) while application providers can bring their applications into the PaaS cloud with the minimal effort.

As Figure 6.4 illustrates, OpenShift can be treated as a customer of the OpenStack-SLAM asking for infrastructure support. Our target is to automatically scale up and down the virtual resources (i.e. VMs) for the PaaS domain as needed. The SLAM not only provides the VMs, but it is also able to customize the VMs using pre-defined scripts so as to provide the "OpenShift-ready" VMs in one click. Specifically, let us suppose a PaaS service provider starts SLA negotiation with an IaaS service provider. When the IaaS service provider has sufficient resources, it will send a counter-offer back to the PaaS service provider with a timeout. Once the offer is accepted within the timeout, then the VM will be created, and the SLAM will automatically log into the VM by matching the public key pair with its private key. Finally, the pre-

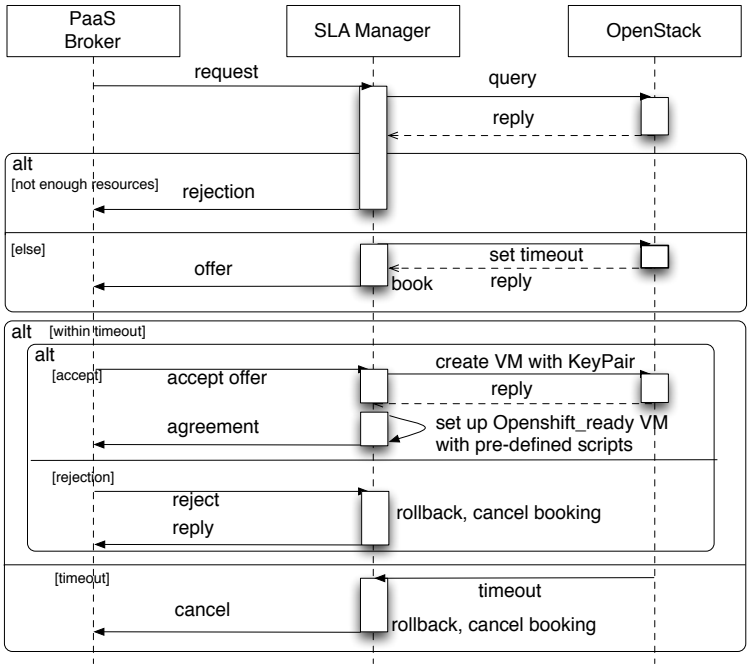


Figure 6.4: Sequence diagram for the negotiation between PaaS and the OpenStack SLA Manager

installed scripts will be executed on the target VM. The execution includes three steps in general, namely:

- Installing the PaaS broker-specific packages.
- Assigning a public Internet Protocol (IP) for the VM.
- Configuration of Mongo database / ActiveMQ / other components associated with the PaaS broker [94].

Thus, the VM can be recognized and controlled by the PaaS broker. Similarly, in case the host is detected under-loaded, the infrastructure can easily be scaled down by removing the VMs. Here, PaaS and IaaS layers are technically interconnected.

6.6 Experimental Results

For the sake of illustrating better our proposed QoS model and its interrelationship with business strategies in Section 7.4, we choose the CloudSim as our simulation platform that is able to model and trace the energy consumption and SLA performance with the automatic host over/under-loading detection, which reduces the preparation of our simulation work to mainly focusing on decision making of the SLA violation and the corresponding adaption [37] (e.g., SLA availability-based VM selection and allocation strategies).

We simulated a data center, including 120 hosts. Each host includes 4 physical physical CPUs (pCPUs), each core with performance of 2000 MIPS, 8 GBs memory and 1 Terabytess (TBs) storage. Virtualization technology allows us to define more virtual CPUs (vCPUs) than the available pCPUs. Usually, we allocate vCPUs onto the VMs. While virtualizing, the expectation is that when one of the VMs, requires increased processing resources, several others on the same host are switched to be idle, which means all the VMs share the MIPS of all pCPUs, of a host. Therefore, the total number of vCPUs across all VMs on a host is potentially greater than the number of pCPUs. We assume the allocation ratio between the vCPU and pCPU is $1.25 : 1$. Therefore, in total, there are 600 vCPUs, available. Based on the cloud infrastructure, we import the real workloads (CPU, utilization) at GWDG in Germany into CloudSim. In this data center, 81 VMs are created. The total simulation time is 24 hours (86400 seconds). Once the cloud environment in CloudSim is setup, it automatically allocates the workloads into the VMs. The interval of utilization measurements is 5 minutes. In this experiment, four types of VM are presented in Table 6.1.

As it was already defined in CloudSim, SLA violation Time per Active Host (SLATAH) is the percentage of time, during which the active hosts have experienced

Table 6.1: VM instance flavor subject to CPU, capacity, cost and profit

Name of instance	**Capacity**	**Cost (€)**	**Profit (€)**
Micro CPU Instance	250 MIPS	4	1.2
Small CPU Instance	500 MIPS	6	1.8
Medium CPU Instance	750 MIPS	10	3
High CPU Instance	1000 MIPS	15	4.5

a CPU utilization of 100%. And Performance Degradation due to Migrations (PDM) is the overall performance degradation by the VMs due to the migrations. As such, SLA performance violation is:

$$SLAV = SLATAH \cdot PDM \text{ [38]} \tag{6.17}$$

6.6.1 Additional Resource Investment

In Section 6.3, by investing additional resource onto the service is able to resolve the SLA violation. In this section, our objective is to experimentally monitor the changes of energy consumption, SLA average violation, the chosen value of β as well as ROI by investing addtional different quantity of CPU resource ranging from 0 to 1000 MIPS with incremental change of 100 MIPS. Finally, we got the following results in Figure 6.5, which illustrate the fact that by investing more resources on the violated service, the total energy consumption increases progressively (Figure 6.5 (a)), and the average SLA violation decreases dramatically (Figure 6.5 (b)). When we consider the profit and failure models, as Figure 6.5 (c) implies that by investing more than 500 MIPS, β_{min} is greater than β_{max}, meaning we cannot achieve enough profits and reasonable low SLA violation at the same time. Similarly, by investing more resources definitely leads to less penalties, but an additional implementation cost will be inevitablly introduced if this investment is more than 650 MIPS (Figure 6.5 (d)). Thus, the value of ROI becomes less than the original ROI (41.8%), meaning the additional investment will not bring more benefits when we take the implementation costs, profits and penalties into consideration. In this scenario, a slight extra investment, e.g., 100 MIPS, leads to a better outcome.

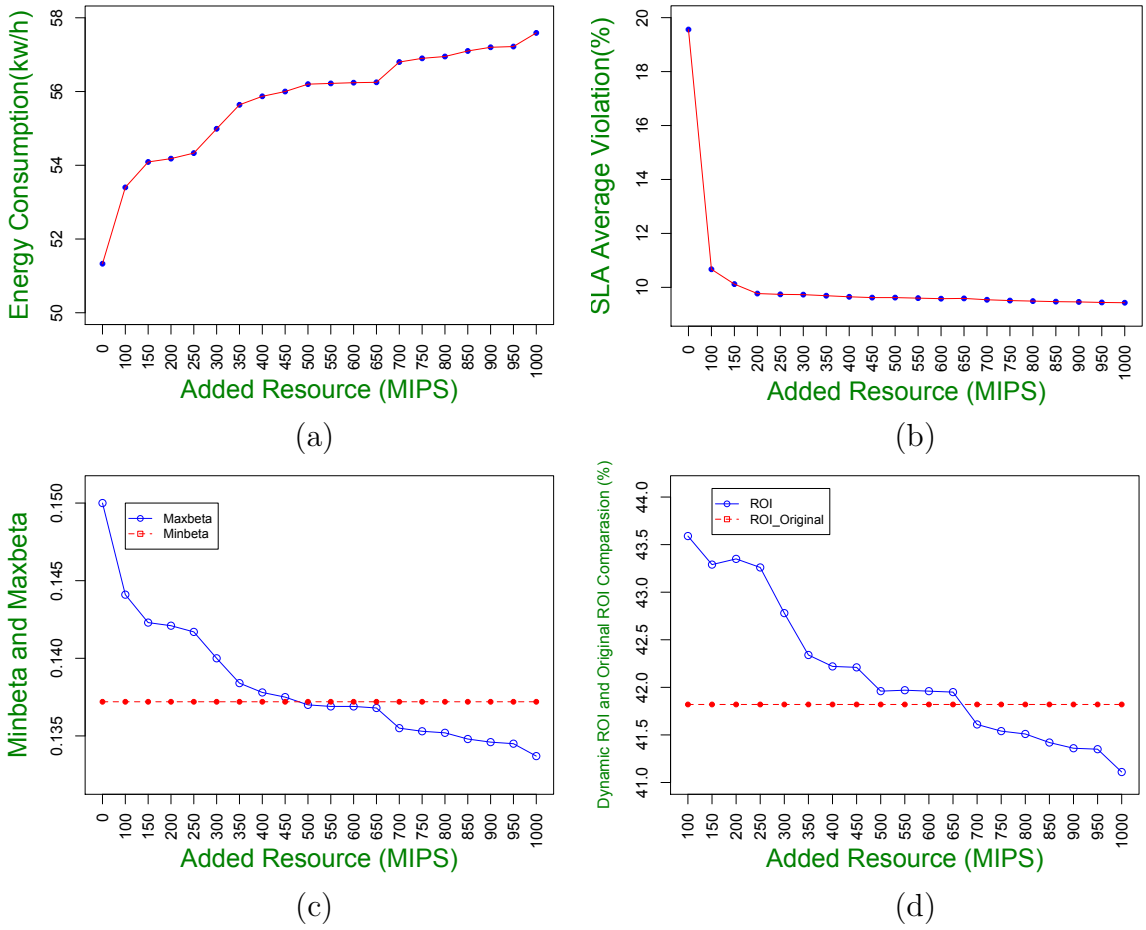


Figure 6.5: (a) Energy consumption, (b) SLA average violation, (c) Relationship between maximum beta and minimum beta, (d) Return of Investment

6.6.2 VM Migration

Actually it is not always practical or even possible to invest additional resource onto the VMs. In this section, we validate the approaches in Sections 6.4.1 and 6.4.2. We consider service performance subject to MIPS. The frequency of the servers' SLAs can be mapped into MIPS ratings. Server CPU utilization is formulated as below:

$$Server\ Utilization = \frac{MIPS_{utilized}}{MIPS_{total}} \tag{6.18}$$

where $MIPS_{utilized}$ is the utilized MIPS of the server and $MIPS_{total}$ is the capacity of the server. For instance, a HP ProLiant ML110 G5 server with 2 cores and each core are allocated 2660 MIPS, i.e. 5320 MIPS capacity per server. Once several SLAs are allocated into the server, at certain point of time, the total utilized MIPS is 5317. Thus, the server utilization is 99.94%. Then in this case we say the server is over-

utilized. And we define that SLA performance violation occurs when a VM cannot get the requested amount of MIPS. This can happen in case when VMs sharing the same host require higher CPU performance that cannot be provided due to the VM consolidation [38].

In case we have to migrate the VMs, by applying the Algorithms 3 into the CloudSim, we want to test how the SLA availability constraint affects the energy consumption and SLA performance. Therefore, the difference between 99.9% and 99.99% is equally divided into 40 intervals, and we set each interval as the *req_availability* of Algorithm 6.4. The results are illustrated in Figure 6.6. In Figure 6.6 (a), when we set SLA availability constraint as 100%, meaning VM live migration is not applied, 88 of 120 hosts are always in active, which means that all VMs have sufficient resources for their applications without SLA performance violation.

However, this causes huge energy consumption (around 85 kilowatt hour (kwh), see upper-right corner of the Figure). Once VM live migration is applied, VM consolidation is always considered in order to save on energy consumption. Thus, for example, with 99.99% constraint, 50 hosts are turned into energy saving mode and the energy consumption decreases dramatically (around 52 kwh).

On the contrary, when the availability constraint is not strict (e.g., 99.9%), the SLATAH value (Figure 6.6 (c)) is relatively low, because as long as an over-load situation is detected, it will be resolved by migrating the VM(s) to the other host(s). Nevertheless, the corresponding PDM value (Figure 6.6 (b)) is very high, because:

- In under-loaded situations, if all the VMs on this host can be migrated to other host(s), the number of VM migrations is increased. However, this leads to "circular flow" for some VMs, meaning they are migrated between the hosts back and forth. Hence, although the server shutting down count increases, energy is actually not saved.

- Extra migrations will definitely cause performance degradation, so the PDM value is also increased.

By applying Equation 6.17, the final SLA performance violation is as illustrated in Figure 6.6 (d). Therefore, from the first experiment, we find that the VM live migration can efficiently reduce the energy consumption of the data center. However, the number of migrations should be balanced in order to achieve the desired service availability and performance requests from the customers.

By applying Algorithm 4 into CloudSim, we strive to find which host will be the optimal destination for allocating the migrated VM(s). The host utility between

0% and 100% is equally divided into 200 intervals, and we set each interval as the *optimalHostUtility* in Algorithm 4. By default, 80% is the threshold utilization. As illustrated in Figure 6.6 (e-f), migrating a VM to a host whose utility is the closest to the threshold utility will lead to the least energy consumption and SLA performance violation. Because otherwise some VMs from the source host will be migrated back and forth until one of them cannot be moved anymore. This not only increases the number of VM migrations unnecessarily and blocks reasonable future migrations due to the availability constraint, but also introduces further energy consumption. As such, in Algorithm 4, we can replace the *optimalHostUtility* with *thresholdHostUtility*.

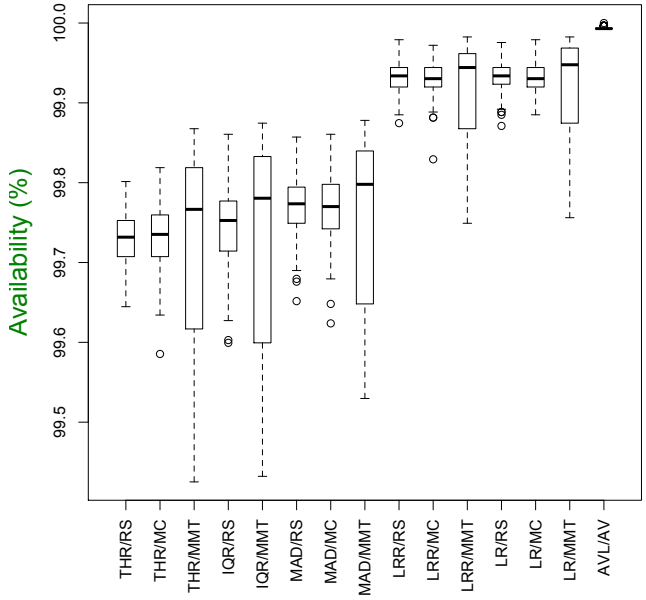


Figure 6.8: Comparison with other strategies in availability

In our final experiment, by taking the results of the experiments above, we set 99.99% as target service availability and 80% as target host CPU threshold utilization during the VM live migration. Using our workloads, we compare the VM allocation algorithms (THR, IQR, MAD, LR and LRR) and the VM selection algorithms (MMT, RS and MC) of CloudSim with our approach, namely AVL/AV. Consequently, the results (Figure 6.8) show that on average the current resource allocation algorithms in CloudSim are able to provide service availability for each VM from around 99.7% to 99.93%. In our approach, the service availability is always kept to be 99.99%, the SLA violation is 0.00155 % and the energy consumption is 54.22 kwh. Whereas, the other approaches in CloudSim introduce around 37 to 51 kwh energy consumption

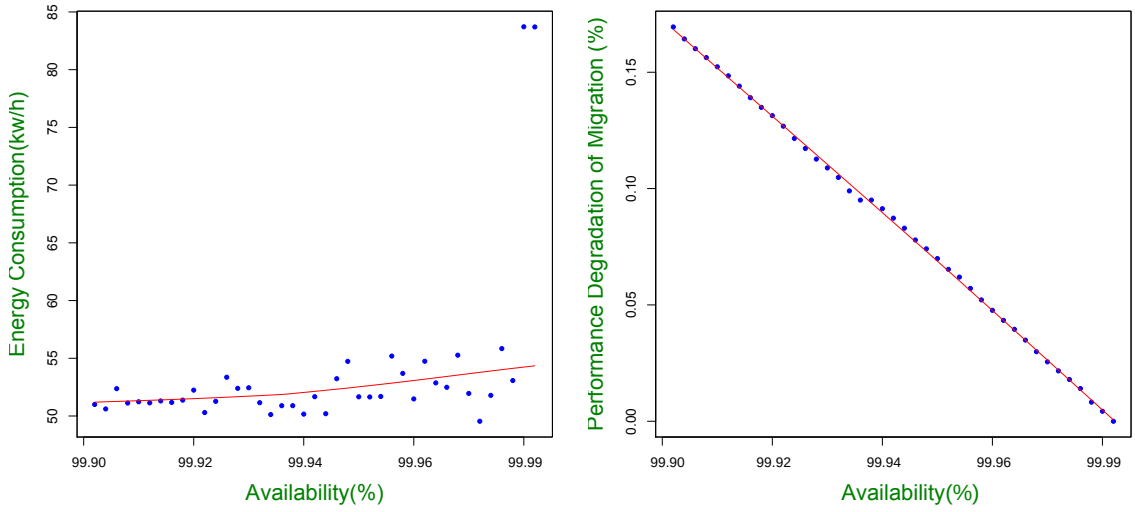
and higher SLA performance violation (Figure 6.9 (a)). Although AVL/AV introduces slightly more energy consumption than the other approaches, using the VM consolidation in general still saves much more energy (85 kwh in non-consolidation case).

The penalty model in Section 6.2 applies when the SLA violation happens. The VM allocation algorithms, using THR, IQR or MAD as VM selection strategy, will lead to availability lower than 99.9%. In this case, the penalty of all three approaches will be increased up to the full cost. Thereby, we selected all the representative approaches to compare with ours in order to find the penalty chain influences in three cloud domains. In Figure 6.9 (b), our approach introduces the fewest penalties. Whereas by using other approaches, penalties are increased exponentially in PaaS and SaaS layers. Especially, the THR approach will return the full cost of the service due to the availability violation.

6.7 Discussion

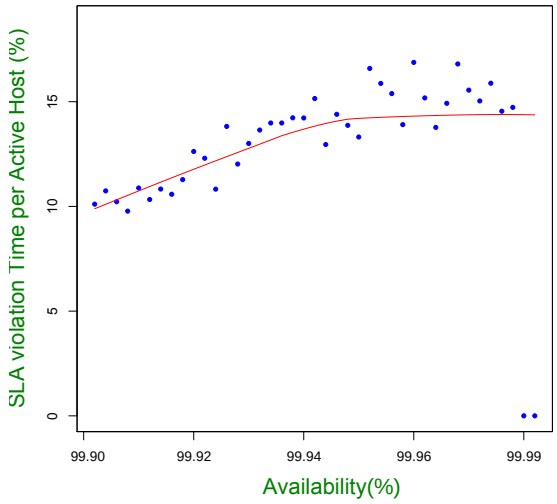
By means of VM live migration technologies, both VM consolidation and service performance can be coordinated. However, short downtimes of services are unavoidable due to the overheads of moving the running VMs. Hence, the respective service interruptions in IaaS reduce the overall service availability and might bring exponential service violation penalties to its associated domains (e.g., SaaS and PaaS). Therefore, in order to provide high service availability and avoid the consequent penalties due to the service violations, in this chapter we present an OpenStack version of the GSLAM. Based on the proposed autonomous SLA violation-filtering framework, by combining IaaS (OpenStack-SLAM) and PaaS (OpenShift) as a use case, applying a multi-domain SLA pricing and penalty model and introducing a resource allocation strategy, we experimentally show that we can manage the VM live migration more efficiently than the current state-of-the-art techniques.

Theoretically, in conjunction with the previous chapters, we have elaborated a utility architecture for business policies and resource optimization based on an structual optimal SLA representation. In next chapter, we take all that a step further to propose a fault-tolerance SLA management framework as an appropriate theoretical model for fine-grained, yet simplified and practical, monitoring of massive sets of SLAs.

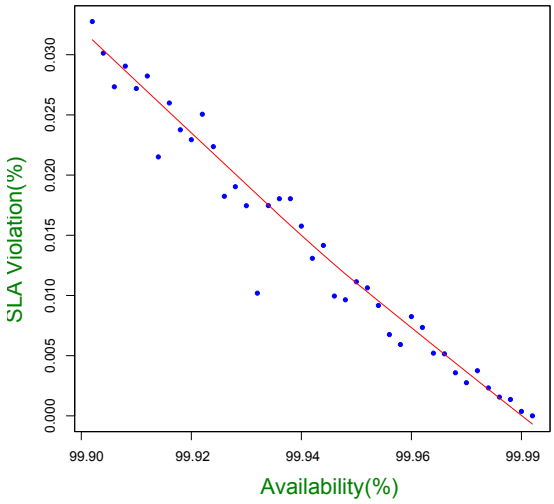


(a)

(b)

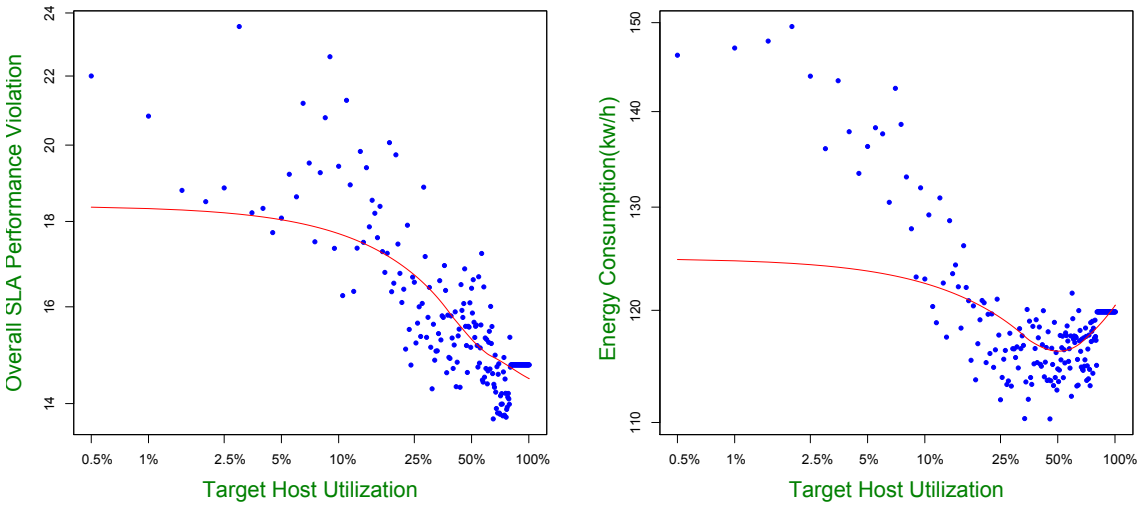


(c)



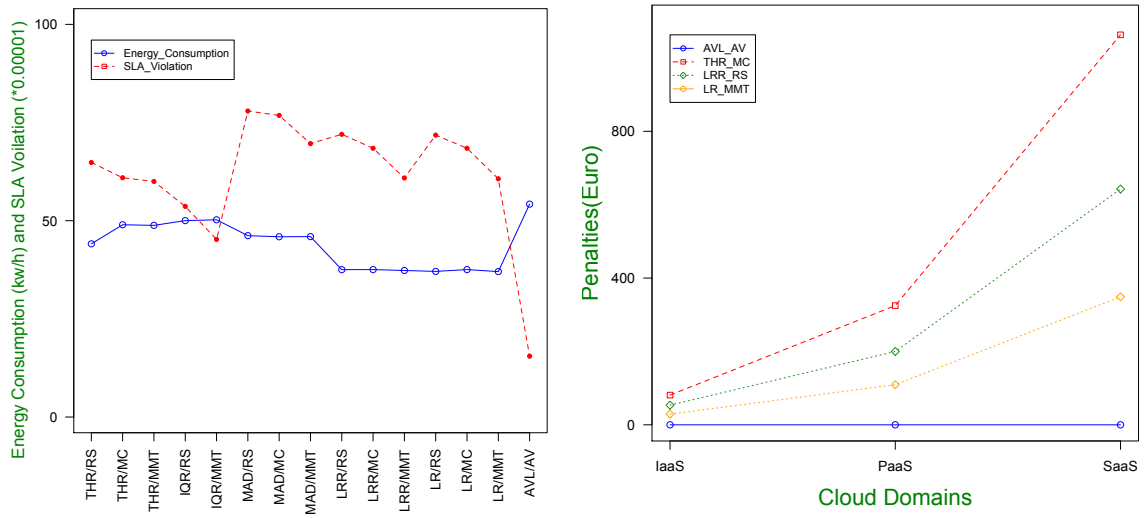
(d)

Figure 6.6: (a-d) Relationship between SLA availability, energy and SLA performance.



(e) (f)

Figure 6.7: (e-f) Optimal target host utilization for allocating the migrated VM(s)



(a) (b)

Figure 6.9: (a) Comparison with other strategies in energy consumption. (b) Comparison with other strategies in SLA violation and penalty.

Chapter 7

Fault Tolerant SLA Management

7.1 Introduction

In this chapter we propose a realistic, modern SLA management system with established theoretical and practical basis. This system separates SLA fault-tolerance concerns into different layers, tied with our optimization and violation prevention strategies. We are using an *actor system* as an implementation basis, achieving compartmentalization and parallelization of SLA management. An actor system builds hierarchies of *actors*, extremely small self-contained and isolated software modules that use strong parent-child relationships to achieve fault-tolerance. In this setting, each actor may have one or more child-actors that are responsible for their own specific functionality. When failure occurs in one of the child-actors, this failure is propagated upwards until it is handled by predefined solutions [96].

7.2 Related Works

Some related works such as [97], [98] and [99] focused on SLA monitoring to detect the violations and corresponding penalty cost without considering the reaction process that avoids the SLA violations.

Many works about SLA monitoring, self-healing and autonomous adaption have recently emerged, such as [100], [101], [102], [103], [104], [105] and [106].

A few related works [105], [106] engaged the monitoring, analysis, planning and execution cycle to react against violations in cloud computing. However, they applied centralized monitoring and reacting. A central system that is designed to monitor all SLAs and react to all detected violations is typically hard to maintain and adapt to different scenarios, and may introduce too much overhead for system scalability and fault-tolerance.

The authors in [100] introduced a self-healing SLA management framework, where the related SLAs communicate with each other hierarchically. And [101] presents a monitoring infrastructure, a general approach for the adaptive multi-source monitoring of SLAs in service choreographies. More specifically, using an event-based monitoring framework (Ganglia), authors introduced an online low level infrastructure adaption mechanism for the sake of avoiding SLA violation. In [102] the authors propose a self-adapting Web-based applications approach by only rearranging the computational resources allocated to the service. And in [103], an automated violations-prevention framework was proposed using event-driven monitoring. Similarly, Emeakaroha et.al [104] proposed CASViD: an application level SLA monitoring and detection framework. In these five papers, the monitoring activities are not centralized; rather the SLA and the service is 1:1 relationship. However, the scope of handling SLA violation is on how to avoid violation propagation between different domains and the adaptation actions for violation handling are tedious. Besides, little attention has been paid to SLA management itself, e.g., how is SLA negotiated, how is monitoring carried out, how to classify and trigger the right event for handling violation.

In some works [104], [109], Java Messaging Service (JMS) [110] was used as communication mechanism together with ActiveMQ [111]; however, JMS implementation requires an external broker, where Akka (the actor-system software used in this thesis) is able to execute without an external broker. Besides, due to the lightweight structure of actors, Akka applications outperform JMS applications in latency and throughput. For instance, one actor only occupies 600 KB memory and 8 GB of memory can include up to 13 million actors [112].

7.3 Decentralization of SLA Management Using an Actor System

In a hierarchical/multi-level SLA monitoring system, it frequently happens that one level is unable to resolve the incident in the first instance and so has to turn to a specialist or superior who can make decisions that are outside the service desk's area of responsibility. This process is referred to as escalation [15].

There are two different types of escalation in ITSM [15]:

- Functional escalation: the support of a higher level specialist is needed to resolve the problem.

- Hierarchical escalation: a manager with more authority needs to be consulted in order to take decisions that are beyond the competencies assigned to this level, for example, to assign more resources onto the service in order to resolve a specific incident.

Therefore, via the concept of escalation, it is possible to separate or decentralize the whole IaaS SLA management concerns into several layers.

7.3.1 Decentralization of IaaS SLA Management

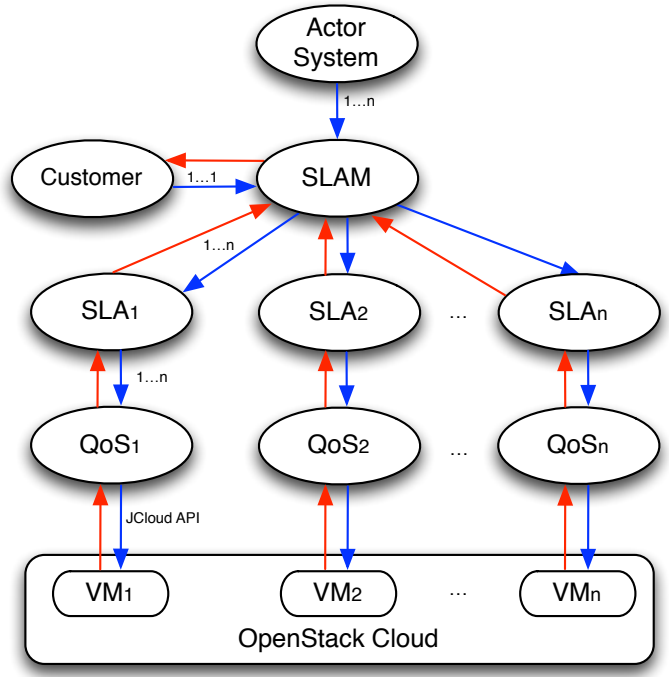


Figure 7.1: Decentralization of IaaS SLA management using actor system

The aforementioned characteristics of escalation can be realized relying on an actor system [112]. In one Java Virtual Machine (JVM), an actor system consists of many actor objects, in which threads are transparently encapsulated. Like everything is an object in Object Oriented Programming (OOP), in actor systems everything is actor. An actor is an entity of computation that communicates with other actors exclusively through message passing. Typically, it encapsulates *state* and *action*. In technical terms, this means processing events and generating responses in an event-driven manner. Every time a message is processed, it is matched against the current list of possible actions of the actor. Thus, action is a function, which defines the strategies to be taken in reaction to incoming messages. As we will argue in this

work, this paradigm is much more suitable for SLA real-time management, because it enables the SLAM to obtain the appropriate high-level view, which allows the management of complex real-time changes of SLA requirements. Therefore, we propose the following setting (see Figure 7.1):

- There is one actor system for maintaining the whole set of SLA, and it includes one or more SLA management contexts.
- Customer and SLA management is a many-to-many relationship.
- By decentralizing the whole SLA management, including SLA negotiation, provisioning, monitoring and adjustment, into hierarchical actors with states and actions, i.e. customer, SLAM, SLA as well as QoS, each actor fulfills its own duty independently.
- One actor, which is assigned to oversee a certain function in the program could split up its task into smaller, more manageable pieces until they become small enough to be handled. Each actor has exactly one supervisor, which is the actor that created it (blue arrow). Therefore, the construction of the actor system is actually the negotiation of SLA. In the end, the whole SLA management actors will be constructed in a top-down manner and shaped as a tree structure.
- During the SLA monitoring and violation detection phase, the corresponding pre-defined strategies could be performed in different layers of actors in the manner of hierarchical escalation. If one actor cannot deal with a certain situation, a corresponding exception message will be forwarded to its supervisor in a bottom-up manner (red arrow).

We believe that via the integration of these basic real-time control mechanisms with the SLA-level policies for reaction inside an actor, complex real-time requirements during SLA monitoring can be met with more flexibility and finer granularity [113]. In the following subsections, we introduce each actor with relevance to its states and actions in detail. To facilitate the introduction, we illustrate each actor with a directed finite state diagram.

7.3.2 Modeling of Actors

We use deterministic finite state machines to mathematically model the actors into a 5-tuple (A, S, s_s, s_e, δ), where:

- A is the action with a finite, non-empty set of input objects, $(a_1, ..., a_n)$.
- S is a finite, non-empty set of states, $(s_s, s_1, ..., s_n, s_e)$.
- s_s is the initial state, an element of S. Thus, $s_s \in S$.
- s_e is the ending state, an element of S. Thus, $s_e \in S$.
- δ is the state-transition function, $\delta(s_x, a_x) \rightarrow s_y$, where s_x and $s_y \in S, a_x \in A$.

In a finite state machine, each of the states will be represented by a rectangular node. Edges with arrows show the transitions from one state to another. Each arrow is labeled with the input that triggers that transition. Inputs that keep the actor in the original state will be represented by a circular arrow. The black node indicates it is the initial state. And the black node with a circle surrounded is ending state.

7.3.3 QoS Actor Finite State Machine

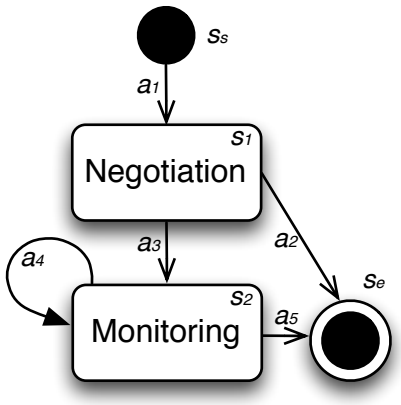


Figure 7.2: Finite state machine diagram of QoS actor

As Figure 7.2 illustrates, the state is changed by invoking a certain action, $\delta(s_s, a_1) \rightarrow s_1$. During SLA negotiation, some non-functional properties of service (i.e. the feasibility of QoS) will be evaluated. The QoS terms can vary depending on the domain; and each SLA actor may contain one or more QoS actors. The negotiation could also be terminated ($\delta(s_1, a_2) \rightarrow s_e$) as long as the timeout expires or one of two negotiating parties withdraws. Once the customer accepts the offer, the QoS actor will turn its state from negotiation to monitoring ($\delta(s_1, a_3) \rightarrow s_2$). In this state, the corresponding QoS will be monitored. If at some point in time a violation is detected, the predefined self-healing action will be activated in order to uphold the negotiated QoS

$(\delta(s_2, a_4) \rightarrow s_2)$. Furthermore, when an SLA is completely fulfilled, its corresponding QoS actor(s) will also be automatically terminated $(\delta(s_2, a_5) \rightarrow s_e)$. We do not define a "violation" state for QoS actors. It might be the case that a QoS actor cannot handle the violation, for instance, the VM is not reconfigurable or the migration is impossible. As such, the QoS actor will inform its parent actor, i.e. SLA actor, via an exception message to fix it.

7.3.4 SLA Actor Finite State Machine

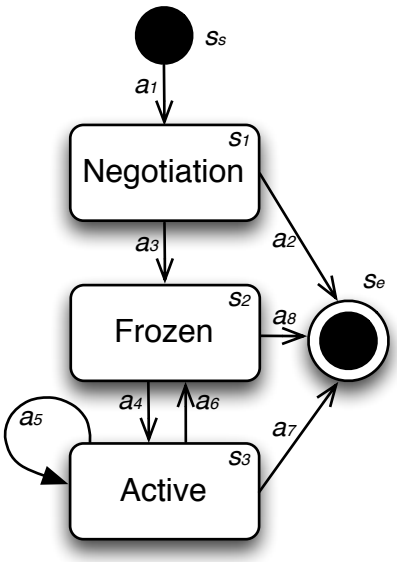


Figure 7.3: Finite state machine diagram of SLA actor

Once an SLAM is established, accordingly one or more attendant SLAs will be created and set in negotiation phase $(\delta(s_s, a_1) \rightarrow s_1)$ (Figure 7.3). Afterwards, all the SLAs are merely potential offers for the customer and they will be automatically turned into frozen state $(\delta(s_1, a_3) \rightarrow s_2)$. Actually, there could be more than one potential solution that is suitable for both the customer and the service provider. Finally, an optimized offer is selected by the SLA manager and forwarded to the customer. Once the offer is accepted, then the selected SLA is turned into the active state $(\delta(s_2, a_4) \rightarrow s_3)$ and the remaining SLAs are kept in a list. While a service is successfully deployed, which means an SLA is running, violations might be detected by its child/children actor(s). In this case, the predefined self-healing action is activated in order to keep the SLA in active state $(\delta(s_3, a_5) \rightarrow s_3)$. For instance, one of the frozen SLA in the list is selected and turned active, the violated SLA actor is frozen and put into the list $(\delta(s_3, a_6) \rightarrow s_2)$. Apparently, the negotiation could be canceled

by any party ($\delta(s_1, a_2) \rightarrow s_e$, $\delta(s_2, a_8) \rightarrow s_e$, $\delta(s_3, a_7) \rightarrow s_e$) in any state of SLA. Again, we do not define a "violation" state for SLA actor. It might be the case that the SLA actor cannot handle the violation, for instance, no alternative solution is available. As such, the SLA actor inform its parent actor, i.e. SLA manager actor, via an exception message to fix it.

7.3.5 SLA Manager Actor Finite State Machine

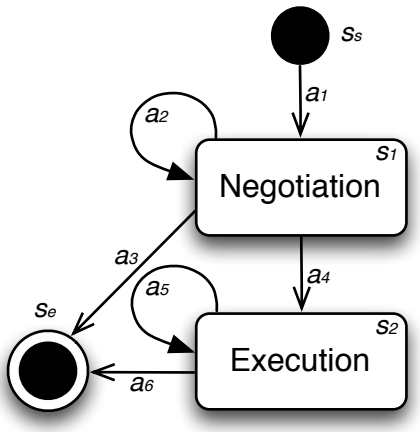


Figure 7.4: Finite state machine diagram of SLA manager actor

Similarly, as Figure 7.4 depicts, when SLA negotiation starts ($\delta(s_s, a_1) \rightarrow s_1$), it may last several rounds ($\delta(s_1, a_2) \rightarrow s_1$), and all of them may require certain iterations. At some point, a final agreement is reached ($\delta(s_1, a_4) \rightarrow s_2$). The negotiation could also be terminated ($\delta(s_1, a_3) \rightarrow s_e$) as long as the timeout expires or one of two negotiating parties withdraws. Furthermore, when an SLA is completely fulfilled, it is automatically terminated ($\delta(s_2, a_6) \rightarrow s_e$). Sometimes, an SLA violation could be avoided by the SLA manager in its execution state ($\delta(s_2, a_5) \rightarrow s_2$). In the SLA manager actor, there is also no "violation" state. It might be the case that the SLA manager actor cannot handle the violation, for instance, no appropriate third party is available for performing outsourcing. As such, the customer actor is informed.

7.3.6 Customer Actor Finite State Machine

Actually, the customer could be an organization, human or even an SLA manager. To facilitate the simulation, we model the customer as an actor (Figure 7.5). Specifically, a customer initiates the SLA negotiation ($\delta(s_s, a_1) \rightarrow s_1$). Two or more parties negotiate an SLAs based on their individual goals. The negotiation may last several

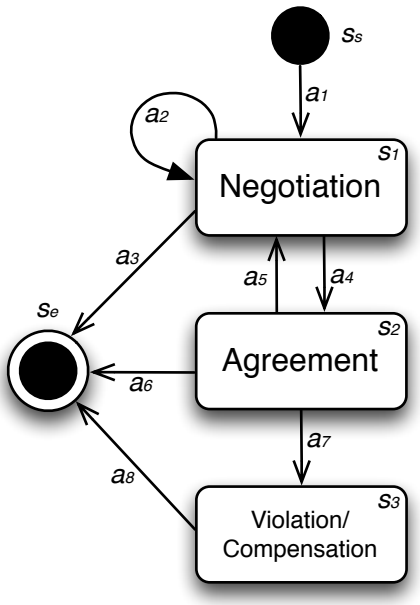


Figure 7.5: Finite state machine diagram of customer actor

rounds ($\delta(s_1, a_2) \rightarrow s_1$) to negotiate offers or counter-offers. The negotiation could also be terminated ($\delta(s_1, a_3) \rightarrow s_e$) as long as the timeout expires or one of two negotiating parties withdraws. After the negotiation, an agreement could be finally reached ($\delta(s_1, a_4) \rightarrow s_2$). Re-negotiating an SLA ($\delta(s_2, a_5) \rightarrow s_1$) could modify its corresponding running service when the SLA is about to be violated. Furthermore, when an SLA is completely fulfilled, it is automatically terminated ($\delta(s_2, a_6) \rightarrow s_e$). In this model, the customer actor is the last station, to which a violation exception broadcasts. SLA violation is finally inevitable if the customer refuses to renegotiate ($\delta(s_3, a_8) \rightarrow s_e$). Should a violation occur, the service provider must fulfill its obligation of compensation given the signed agreement ($\delta(s_2, a_7) \rightarrow s_3$).

7.3.7 Autonomous SLA Violation Monitoring and Filtering

Based on the finite state machines described above, an autonomous SLA violation-filtering framework is presented. In Figure 7.6, four modules are classified and defined individually into four layers.

In this framework, each module has exactly one supervisor, which is the module that creates it. If one module does not have the means for dealing with a certain situation, it sends a corresponding exception message to its supervisor, asking for help. The recursive structure then allows handling failure at the right level. Everything in this framework is designed to work in a complete distributed environment. Thus, all

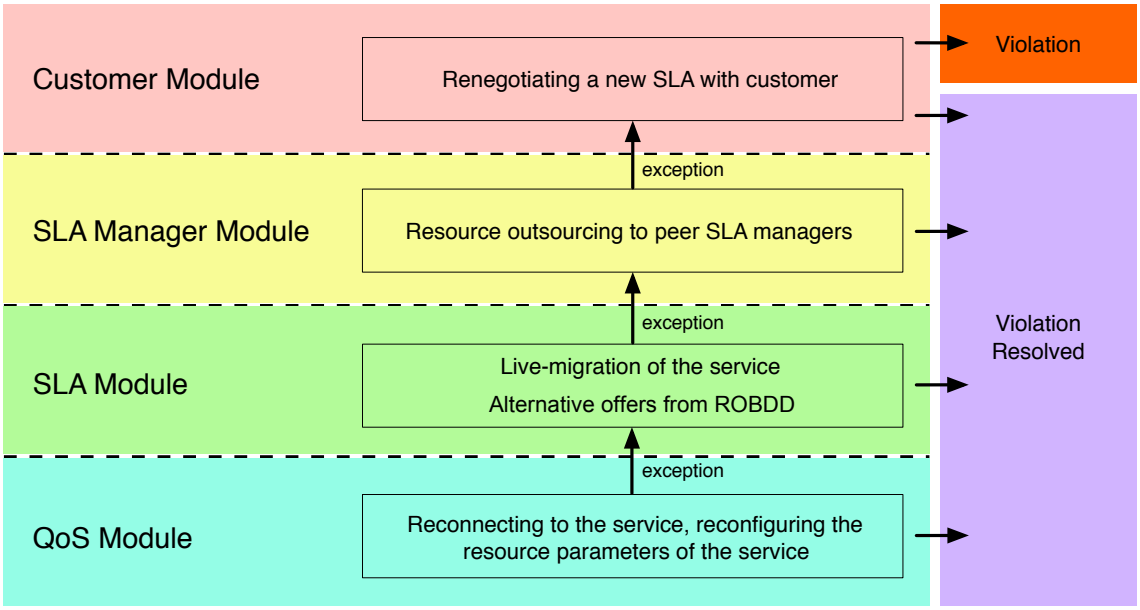


Figure 7.6: Autonomous SLA violation filtering framework

interaction of modules is pure message passing, which is exchanged in a non-blocking, asynchronous and highly performing event-driven manner. In message passing programming, an asynchronous call allows the caller to progress after a finite number of steps, and the completion of the method may be signaled via some additional mechanism (it might be a registered callback or a message). Furthermore, non-blocking means that no thread is able to indefinitely delay others. Non-blocking operations are usually preferred to blocking ones, as the overall progress of the system is not trivially guaranteed when it contains blocking operations [112]. Therefore, the SLA violation could be efficiently eliminated.

Consequently, each module strives to resolve the (potential) violation in its own layer. At the very beginning, when a certain violation (e.g., service performance issue) is detected by QoS module, it tries to fix it by restarting the service or adding extra resources on the running service and without altering the content of the SLA. In this case, the customer does not observe any change and the running SLA is still valid. Otherwise, it sends an exception message to its supervisor (SLA module). In the SLA module, the corresponding exception handlers have already been predefined with all possible solutions in a list. Likewise, if the SLA module cannot find an alternative solution, it forwards the exception to its supervisor (SLA manager module). The SLA manager module outsources to other cloud third party for extending its capacity. Otherwise, a renegotiation could be initialized by the customer module so as

to establish a new SLA without paying the penalty. And there could also be several rounds of renegotiation within a certain time span.

7.4 Modeling of Negotiation Scenario

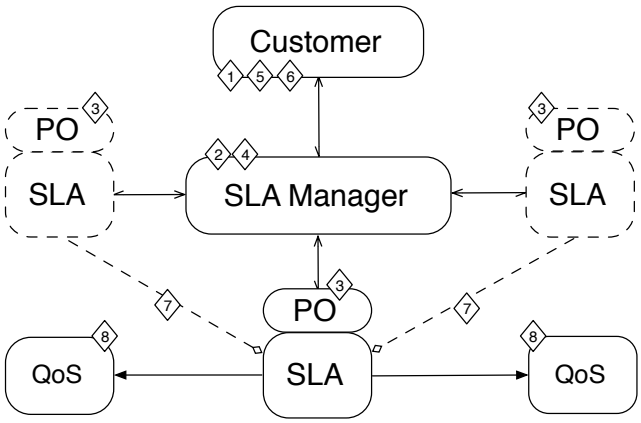


Figure 7.7: Negotiation and provision scenario

In Figure 7.7, the customer may initialize a negotiation with the SLA manager by sending a customized SLA template. An SLA template is a document that describes a set of options for each domain-specific service including functional and non-functional properties (Step 1). In the IaaS scenario, the provider should check whether it is able to offer the required availability and/or performance for the specific service or not. For instance, based on the current load of a host, the feasibility of allocating a VM in the host translates to whether the required number of MIPS is available or not. In case the host is overloaded, it should be ignored during the negotiation.

When an SLA manager is established (Step 2), it will split the planning and optimization (PO) (Step 3) task(s) into one or more attendant SLA actor(s). According to the resource configuration, each SLA actor will start evaluation simultaneously and finally come up with a price quotation.

After the SLA manager collects all the quotations from SLA actor(s), it will select the optimal one and forward it to the customer (Step 4). The latter may refuse the counter-offer (Step 5). Or else, if the offer is accepted (Step 6), this SLA will be turned into active and all the other SLAs will be kept as part of the selected SLA (Step 7). The final selected SLA will create its own QoS actors accordingly (Step 8).

7.5 Modeling of Service Monitoring and Fault Tolerance Scenario

7.5.1 Violation Handling in QoS Module

Two QoS terms are considered, namely service availability and service performance subject to MIPS. Through Nagios we can monitor the real-time service availability,

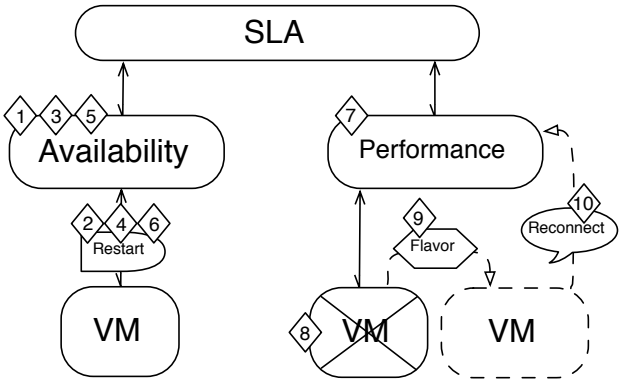


Figure 7.8: Violation handling in QoS module

if a violation is detected, instead of automatically sending an alert email to administrator and let her handle it manually, in this case the QoS actor will receive the exception, then according to the causes of the exception, the corresponding action will be triggered to resolve the issue. We propose that the QoS actor will automatically restart the VM and reconnect to it within a certain timeout (Step 1 in Figure 7.8). Such an operation could be repeated 3 times at the most (Step 2, 4, 5, 6).

For service performance, we fed the real-time resource utilization workloads from Nagios into CloudSim and applied its available resource allocation strategies. Hence, once a host cannot provide enough resources to this VM (Step 7), i.e. less MIPS is allocated to the VM than expected, the QoS actor will try to reconfigure the VM with new appropriate CPU flavor (Step 8, 9), which is also dependent on the current utilization of the host. Finally, a new configured VM will be assigned to this SLA (Step 10).

The resource utilization of each individual service varies according to the time of day. For instance, the peak access may happen around noon and less resource utilization be needed at midnight. Thus, it might be the case that the service provider cannot offer network access of uniform quality as its promises in the SLA that governs service response time. Therefore, in the QoS module, if we cannot allocate more extra

resources on the service or the service is still unavailable after reconnection, VM live migration will be applied in the SLA module.

7.5.2 Violation Handling in SLA Module

If the service is still unavailable after three efforts to reconnect, then the service will be migrated to another suitable host (Step 1, 2 in Figure 7.9). Similarly, when we cannot allocate additional resources, VM live migration will be applied (Step 4, 5), thus in our scenario two types of the VM will be selected to be migrated. Specifically, we will migrate the VM that is unavailable (e.g. server outage) in order to make the VM become available again; and we will migrate the VM with less allocated MIPS in order for the VM to have enough CPU power supplied. Finally, a new VM will be assigned to this SLA (Step 3, 6).

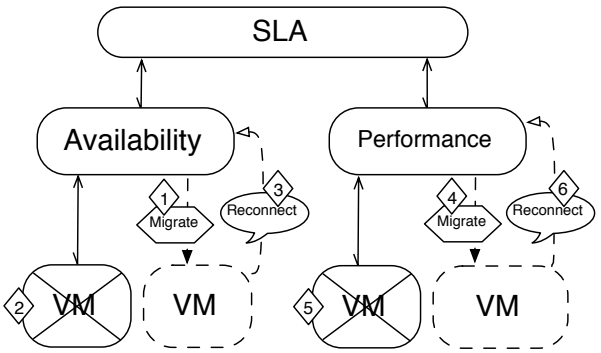


Figure 7.9: Violation handling in SLA module

For VM migration, once we decide which VM will be selected, then the only problem is how to find a suitable server to host the VM by considering the energy consumption and SLA failure rate amongst others. If we manage to find such a host for the migrated VM, then in this layer, we say the SLA violation subject to availability and performance can be resolved. However, short downtimes of services are unavoidable during VM live migration due to the overheads of moving the running VMs. Hence, the respective service interruptions in IaaS reduce the overall service availability and it is possible that the customers expectations on responsiveness are not met. Thus, it can be case that a VM cannot be migrated any more. In this case, the SLA module will report an exception to its supervisor – SLA manager module.

7.5.3 Violation Handling in SLA Manager Module

In the SLA manager module, the ultimate goal of SLA outsourcing is to extend the capacity of the main service provider temporarily without paying penalties due to service violation. In this topic, third parties play a particularly important role in decision making for the intermediate SLA (s), which is (are) not transparent for the end user of the main service provider.

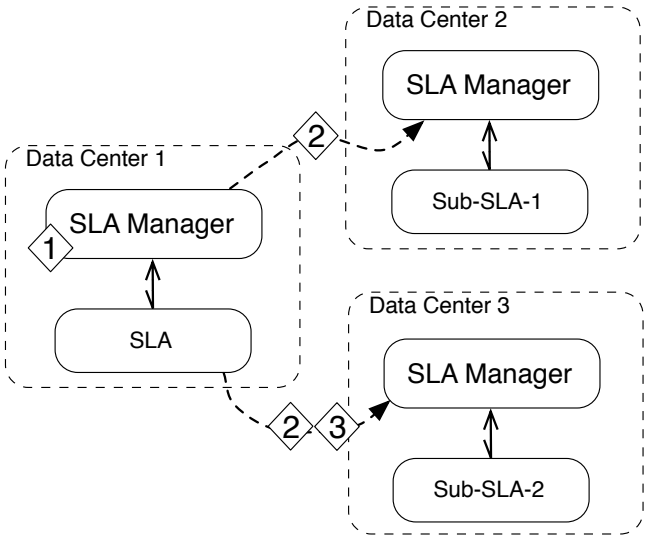


Figure 7.10: Violation handling in SLA manager module

An actor system, using its remote feature could extend the capacity of SLA management (Step 1). In our proposal, one JVM includes one or more SLA managers; different actors in multiple JVMs could communicate with each other by sending message. In this case, remote actor references represent actors, which are reachable using remote communication, i.e. sending messages to them will serialize the messages transparently and send them to the remote JVM. This feature could be seamlessly applied into resource outsourcing strategies. Therefore, each JVM will be represented as a data-center (simulated by CloudSim) located in different place. All the data centers will communicate (start SLA negotiation) over the network (Step 2) in order to extend their resource capacity (Figure 7.10). In case more than one party offers the same service, we will apply the 3-dimensional decision-making strategy, and finally one of them will be selected (Step 3).

7.5.4 Violation Handling in Customer Module

The SLA renegotiation in customer module implies further round(s) of SLA negotiation, during which one or more terms of an established agreement may be changed. A successful renegotiation assesses the existing agreement and replaces it with a new one. The whole interaction is carried out between the customer and the service provider. For example, a user may need to change the amount of CPU time they originally agreed to purchase if they find their computational job is more complex than they originally thought and it requires more CPU time to fully complete [114]. On the other hand, a service provider has to trigger the SLA renegotiation with a new counter-offer in order to adapt to changes in its resource capacity. Since modeling of the behavior or preferences of the customer is out of the scope of the thesis, in our implementation we treat the SLA renegotiation as a new round of SLA negotiation, and we leave the decision to both customer and service provider, which will unquestionably provide such an opportunity for both sides of the negotiating parties that is able to adjust to their new requirements or resource allocation scheme and in the end to avoid the SLA violation and the further penalties. As Figure 7.11 illustrated, the SLA violation could be removed by starting re-negotiation so as to ensure the online reputation of service providers.

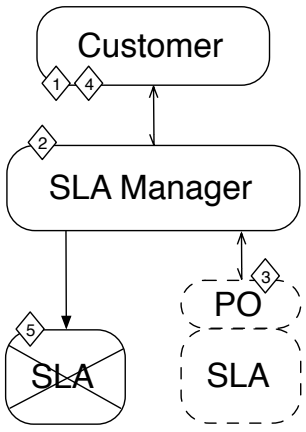


Figure 7.11: Violation handling in customer module

There is a necessity to add SLA renegotiation in SLA management for cloud-based system. This will allow customers and providers to initiate changes in the established agreement, for instance to increase allocated storage due to increased data accumulation rate, or to increase bandwidth capacity due to increased peak traffic at certain times of day. It is evident that changing circumstances leads to the requirement for SLA renegotiation [13].

Both parties in the contract should be allowed to carry out the initiation of renegotiation and both parties should be allowed to cancel an existing contract. The initiation of re-negotiation should be allowed through non-binding enquiries to the other party so that an estimate as to how much it would cost to change the contract can be obtained before committing to a new contract [114].

7.6 Discussion

In this chapter, we propose a realistic, modern SLA management system with the application of actor systems as an appropriate theoretical model for fine-grained, yet simplified and practical, monitoring of massive sets of SLAs. We show that this is a realistic approach for the automated management of the complete SLA lifecycle, including negotiation and provisioning, but focus on monitoring as the driver of contemporary scalability requirements. Our approach separates the agreement to build hierarchies of actors using strong parent-child relationships to achieve fault-tolerance concerns into multiple autonomous layers that can be hierarchically combined into an intuitive, parallelized, effective and efficient management structure.

Chapter 8

Conclusions and Future Directions

In this chapter, I summarize the thesis and discuss its contributions. Beyond that, I investigate the possible research directions, which could extend or refine the results and approaches presented in this thesis.

8.1 Summary

In this thesis, I derive the common challenges and address the specific problems in different phases of cloud SLA lifecycle management. In SLA establishment phase, with SLA representation management, feasibility management and planning optimization strategies.

During the SLA creation phase, by choosing the IaaS SLA template model from SLA@SOI, I propose an approach for representing the SLAs as ROBDDs aiming to facilitate the SLA negotiation. For the sake of further remedying the structural suboptimal problem for SLA ROBDD representation, a ROBDD node optimization algorithm is proposed, where the number of nodes for the ROBDDs can be efficiently reduced to make SLAs occupy less memory space (19.37% reduction). Additionally, eliminating the paths that are semantically redundant via ROBDD path optimization algorithms can decrease the total size of paths in a ROBDD. As such, the SLA automated negotiation is accelerated (24.07% - 27.75% reduction). Furthermore, a novel alternative solution for reducing semantically redundant paths during construction is introduced. I approve that all the options of an SLA term are mutually exclusive. Hence, customizing an SLA term is rewritten as appropriately determining an option specifically for this term. This process is treated as an implementation of an SLA TRS.

Once we have the optimal SLA representation structure, another objective of my approach is to minimize implementation and outsourcing costs to boost competitive-

ness, while respecting business policies optimization for profits and risks. Therefore, I design and implement a greedy algorithm for outsourcing, employing cost and subcontractor reputation as selection criteria and local resource configurations as a constraint satisfaction problem to seek for acceptable profits and failure risks. Thus, it becomes possible to provide educated price quotes to customers and establish safe electronic contracts automatically. Besides outsourcing to third party, for the sake of increasing the utilization of the local resources and satisfying as many customers as possible, I introduce an advance reservation resource planning methodology during SLA negotiation by using computational geometry. As such, the infrastructure resources can be verified, recorded and managed efficiently.

During the SLA operation phase, for resource management in multi-domain SLA problem, I present an SLA management architecture, which is able to combine IaaS and PaaS in an application case that features multi-domain SLA management. By the introduction of a pricing and penalty model that considers such a multi-domain scenario, I apply the developed resource allocation strategies for VM selection and allocation during live migration. I simulate the full scenario and illustrate the suitability of my proposed strategy for the efficient management of VM live migration. Hence the agreed service availability is not violated even without paying the extra penalties and a trade-off between the service provider's objectives and the customers' expected QoS requirements can also be achieved successfully.

Finally, I apply the actor system that matches the service escalation strategy. I propose to decentralize the whole SLA management architecture to build hierarchies of actors using strong parent-child relationships in order to resolve the SLA incident in an event-driven style. Comparing to other event-driven programming languages, the actor system builds hierarchies of actors using strong parent-child relationships to achieve fault-tolerance. In such a self-heal system, each actor may have one or more child-actors that are responsible for their own specific authority and functionality. When failure occurs in one of the child-actors, it will be propagated upwards until it is handled by predefined solutions.

8.2 Outlook

In the future, the structural optimization of the SLA representation can be extended in the following directions. First of all, the TRS could be applied to SLA translation. Thus, a suitable representation and transformation would need to be defined to use term rewriting into this scenario. Secondly, there is a gap in using the canonical form

of the structure for outsourcing and decision-making, related to matching paths from different BDDs and finding out whether they are equivalent so that the outsourced requirements match the available services from sub-contractors.

Regarding the methodology for business policies modeling, further possibilities need to be investigated. I would like to extend the work to include more service types and the QoS constraints for further improving resource utilization. Besides, a strategy for balancing the load of servers so as to maximize the utilization of the whole data center is also worth being considered. For service advance reservation, the computational geometry technology could be applied and explored in virtual resource allocation and reallocation (in case of SLA violation). I would also like to extend this work with proper business models and pricing policies. Finally, a combination of online and offline algorithms for resource re-planning could form future extensions of this work.

During the SLA operation phase, I approve my work in CloudSim environment. In the future, I would like to take this work one step further by performing the cloud emulation tool "Emusim" to automatically extract information from various types of workloads (e.g., CPU intensive, memory intensive, network intensive et cetera) via emulation. I can further use this information to generate the corresponding simulation model. Thus, the results of emulation can be used to prove the correctness and accuracy of the simulation.

In order to make the entire SLA management to be cross-domain applicable, I would like to apply the actor system to multi-cloud structure, i.e. the relationship between IaaS, PaaS and SaaS layers. For instance, the SLAs in SaaS layer are going to be failed if its related SLAs in PaaS layer are violated. Therefore, lacking of an effective relations between dependent SLAs is a vital challenge, which makes the SLA management system inefficient. By using the actor system, the SLAs between domains can communicate with each other in terms of handling SLA translation, composition and violation. Thus, the difficulty of cross-domain SLA management can be released.

Bibliography

[1] T. Vázquez, E. Huedo, S. R. Montero and M.I. Llorente, "Evaluation of a Utility Computing Model Based on the Federation of Grid Infrastructures". Springer-Verlag Berlin Heidelberg, pages 372-381, 2007.

[2] Q. Zhang, L. Cheng and R. Boutaba, "Cloud computing: state-of-the-art and research challenges". Journal of Internet Services and Applications, pages 7-18, 2010.

[3] N. Antonopoulos and L. Gillam, "Cloud Computing: Principles, Systems and Applications" Springer Verlag, 2010.

[4] P. Mell, T. Grance, "The NIST Definition of Cloud Computing". National Institute of Standards and Technology Special Publication 800-145, pages 1-3, 2011.

[5] F. Gens, "IDC Predictions 2014: Battles for Dominance and Survival on the 3rd Platform [Online]". Technical report, International Data Corporation, 2014.

[6] (2014) Amazon EC2 Cloud, [Online]. Available: http://aws.amazon.com/ec2/

[7] (2014) Google App Engines, [Online]. Available: http://code.google.com/appengine/

[8] (2013) SAP Hana, [Online]. Available: http://www.saphana.com

[9] P. Wieder., J. Butler, R. Yahyapour (Eds.), "Service Level Agreements For Cloud Computing" Springer-Verlag, page 3, 2011.

[10] J. Bartlett, D. Hinley, B. Johnson, D. Johnson, C. Keeling, V. Lloyd, L. MacDonald, J. Mather, G. McLaughlin, C. Rudd, D. Wheeldon, R. Young, "Best Practice for Service Delivery". The Stationery Office, 2007.

[11] (2014) Information Technology, Cloud computing, Overview and vocabulary, ISO/IEC 17788:2014(E), [Online]. Available: http://standards.iso.org/.

[12] A. Chazalet, "Service Level Checking in the Cloud Computing Context". IEEE 3rd International Conference on Cloud Computing, pages 297-304, 2010.

[13] A. F. M. Hani, I. V. Paputungan, M. F. Hassan, "Service Level Agreement Renegotiation Framework for Trusted Cloud-Based System". Future Information Technology, Lecture Notes in Electrical Engineering, Volume 276, pages 55-61, 2014.

[14] L. L. Wu, R. Buyya," Service Level Agreement (SLA) in Utility Computing Systems". CoRR abs/1010.2881, page 27, 2010.

[15] (2014) ITIL (IT Infrastructure Library): ITIL Version 3 - Service Improvement, [Online]. Available: http://www.itil-officialsite.com.

[16] (2013) Cloud computing service level agreements. Exploitation of Research Results, [Online]. Available: http://ec.europa.eu/.

[17] Ron, S., Aliko, P. "Service level agreements". Internet NG project, 2001.

[18] (2013) SLA@SOI, [Online]. Available: http://sla-at-soi.eu/.

[19] SLA Management Handbook, Vol. 2, Concepts and Principles, Release 2.5. The TeleManagement Forum, Morristown, New Jersey, USA, 2005.

[20] J. J. Lee, R. Ben-Natan, "What are Service Level Agreements? Integrating Service Level Agreements Optimizing Your OSS for SLA Delivery". Wiley Publishing, Inc., Indianapolis, Indiana, USA, 1st edition, pages 325, 2002.

[21] V. Stantchev, C. Schröpfe, "Negotiating and Enforcing QoS and SLAs in Grid and Cloud Computing". Advances in Grid and Pervasive Computing, Springer Berlin / Heidelberg, pages 25-35, 2009.

[22] I. Brandic, D. Music, S. Dustdar, "Service mediation and negotiation bootstrapping as first achievements towards self-adaptable grid and cloud services". Proceedings of the 6th International Conference on Autonomic Computing (industry session on Grids Meets Autonomic Computing), pages 1-8, 2009.

[23] A. Kertesz, G. Kecskemeti, I. Brandic, "An SLA-based resource virtualization approach for on-demand service provision". Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing, pages 27-34, 2009.

[24] S. Venugopal, X. C. Chu, R. Buyya, "A Negotiation Mechanism for Advance Resource Reservations using the Alternate Offers Protocol". Quality of Service, IWQoS, pages 40-49, 2008.

[25] E. Yaqub, P. Wieder, C. Kotsokalis, V. Mazza, L. Pasquale, J. Rueda, S. Gomez, A. Chimeno, "A Generic Platform for Conducting SLA Negotiations". Wieder. P., Butler, J., Yahyapour R. (Eds.) Service Level Agreements For Cloud Computing, Part 4, Springer-Verlag, pages 187-206, 2011.

[26] (2014) Information Technology, Cloud computing, Reference architecture, ISO/IEC 17789:2014(E), [Online]. Available: http://standards.iso.org/.

[27] J. Kosinski, D. Radziszowski, K. Zielinski, S. Zielinski, G. Przybylski, P. Niedziela, "Definition and Evaluation of Penalty Functions in SLA Management Framework". Fourth International Conference on Networking and Services, ICNS 2008, pages 176-181, 2008.

[28] M. Becker, N. Borrisov, V. Deora, O. Rana, D. Neumann, "Using k-Pricing for Penalty Calculation in Grid Market". Proceedings of the 41st International Conference on System Sciences, page 97, 2008.

[29] O. F. Rana, M. Warnier, T. B. Quillinan, F. Brazier, D. Cojocarasu, "Managing Violations in Service Level Agreementst". Grid Middleware and Services, Springer-Verlag US, pages 349-358, 2008.

[30] (2008) IANOS: An intelligent application oriented scheduling framework for an HPCN grid, [Online]. Available: http://publica.fraunhofer.de/documents/N-94198.html.

[31] (2008) SmartLM, [Online]. Available: http://www.smartlm.eu.

[32] (2009) SLA4D-Grid, [Online]. Available: http://www.sla4d-grid.de.

[33] A. Keller, H. Ludwig, "The WSLA framework: specifying and monitoring service level agreements for Web Services". Journal of Network and Systems Management, pages 57-81, 2003.

[34] K. T. Kearney, F. Torelli, C. Kotsokalis, "SLA: An abstract syntax for Service Level Agreements", GRID2010, pages 217-224, 2010.

[35] (2007) Web Services Agreement Specification (WS-Agreement), [Online]. Available: http://www.ogf.org/documents/GFD.107.pdf.

[36] C. Kotsokalis, R. Yahyapour, M. A. R. Gonzalez, "Modeling Service Level Agreements with Binary Decision Diagrams". 7th International Joint Conference on Service-Oriented Computing (ICSOC2009), Springer-Verlag, pages 190-204, 2009.

[37] A. Beloglazov, J. Abawajy, B. Rajkumar, "Energy-aware Resource Allocation Heuristics for Efficient Management of Data Centers for Cloud Computing". Future Generation Computer Systems, Volume 28, Issue 5, pages 755-768, 2012.

[38] A. Beloglazov, B. Rajkumar, "Optimal Online Deterministic Algorithms and Adaptive Heuristics for Energy and Performance Efficient Dynamic Consolidation of Virtual Machines in Cloud Data Centers". Concurrency and Computation: Practice and Experience, ISSN: 1532-0626, Wiley Press, New York, USA, DOI: 10.1002/cpe.1867, 2011.

[39] F. Salfner, P. Troeger, M. Richly, "Dependable Estimation of Downtime for Virtual Machine Live Migration". International Journal On Advances in Systems and Measurements, volume 5, numbers 1 and 2, 2012.

[40] S. Akoush, R. Sohan, A. Rice, A. W. Moore, A. Hopper, "Predicting the Performance of Virtual Machine Migration". 2010 IEEE International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), pages 37-46, 2010.

[41] (2014) CloudWatch, [Online]. Available: http://aws.amazon.com/cloudwatch/.

[42] (2014) Nagios, IT Infrastructure Monitoring, [Online]. Available: http://www.nagios.org.

[43] (2014) OpenTSDB, The Scalable Time Series Database, [Online]. Available: http://opentsdb.net.

[44] P. S. Weygant, "Basic High Availability Concepts". Clusters for High Availability: A Primer of HP Solutions, Second Edition, Prentice Hall, ISBN-10: 0-13-089355-2, pages 1-39, 2001.

[45] A. Chandra, P. Goyal, P. Shenoy, "Quantifying the benefits of resource multiplexing in on-demand data centers". 1st ACM Workshop on Algorithms and Architectures for Self-Managing Systems, 2003.

[46] C. Zhou, L.T. Chia, B.S. Lee, "DAML-QoS Ontology for Web Services". IEEE International Conference on Web Services, page 472, 2004.

[47] G. Dobson, A. Sanchez-Macian, "Towards Unified QoS/SLA Ontologies". IEEE Services Computing Workshops, pages 169-174, 2006.

[48] G. S. Machado, B. Stillerm, "Investigations of an SLA Support System for Cloud Computing (SLACC)". Praxis der Informationsverarbeitung und Kommunikation (PIK), pages 80-86, 2011.

[49] C. E. Shannon, "A symbolic analysis of relay and switching circuits". AIEE (57), pages 713-723, 1938.

[50] H. R. Andersen, "An Introduction to Binary Decision Diagrams". Citeseer, pages 8-15, 1999.

[51] R. Ebendt, R. Drechsler, "Advanced BDD Optimization". Springer-Verlag, 2005.

[52] J. W. Klop, "Term Rewriting Systems, Amsterdam". The Netherlands: Stichting Mathematisch Centrum, 1990.

[53] F. Baader, T. Nipkow, "Term Rewriting and All That". Cambridge University Press, pages 1-2, 34-35, 1999.

[54] 74LVC1G386, 3-input Exclusive-Or gate, Data Sheet, NXP B.V. (2007).

[55] P. W. C. Prasad, M. Raseen, S. M. N. A. Senanayake, A. Assi, "BDD Path Length Minimization Based on Initial Variable Ordering". Journal of Computer Science". Science Publications, 2005.

[56] R. Drechsler, W. Guenther, F. Somenzi, "Using lower bounds during dynamic BDD minimization". IEEE Transaction on CAD, pages 50-57, 2001.

[57] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams". IEEE/ACM international conference on Computer-aided design, IEEE Computer Society Press, Los Alamitos, pages 8-15, 1993.

[58] M. Ehrgott, "Multicriteria Optimization (2. ed.)". Springer-Verlag, pages 171-195, 2005.

[59] (2007) JavaBDD, [Online]. Available: http://javabdd.sourceforge.net/.

[60] K. Lu, T. Röblitz, P. Chronz, C. Kotsokalis, "SLA-Based Planning for Multi-Domain Infrastructure as a Service". 1st International Conference on Cloud Computing and Services Science, Springer-Verlag, pages 343-351, 2011.

[61] N. W. Paton, M. A. T. de Aragao, K. Lee, A. A. Fernandes, R. Sakellariou, "Optimizing Utility in Cloud Computing through Autonomic Workload Execution". IEEE Data Eng. Bull., pages 51-58, 2009.

[62] H. N. Van, F. D. Tran, J. M. Menaud, "SLA-Aware Virtual Resource Management for Cloud Infrastructures". International Conference on Computer and Information Technology, IEEE Computer Society. pages 357-362, 2009.

[63] H. Wada, J. Suzuki, K. Oba, "DAML-QoS Ontology for Web Services". IEEE Computer Society, Proceedings of the 2009 Congress on Services-I-Volume 00, pages 661-669, 2009.

[64] J. L. Hellerstein, K. Katircioglu, M. Surendra, "A framework for applying inventory control to capacity management for utility computing". Integrated Network Management (IM 2005), pages 237-250, 2005.

[65] B. Sotomayor, S.R. Montero, M.I. Llorente, I. Foster, "Resource Leasing and the Art of Suspending Virtual Machines". 11th IEEE International Conference on High Performance Computing and Communications (HPCC-09), IEEE Press, pages 25-27, 2009.

[66] M. A. S. Netto, K. Bubendorfer, R. Buyya, "SLA-based advance reservations with flexible and adaptive time QoS parameters". ICSOC '07 Proceedings of the 5th international conference on Service-Oriented Computing, pages 119-131, 2007.

[67] J. L. Lucas, C. Carrión, B. Caminero, "Flexible advanced-reservation (FAR) for clouds". 1st International Conference on Cloud Computing and Services Science (CLOSER2011), pages 11-21, 2011.

[68] C. Castillo, G.N. Rouskasb, K. Harfoushb, "Efficient Resource Management using Advance Reservations for Heterogeneous Grids". Intl. Parallel and Distributed Processing Symposium (IPDPS), 2008.

[69] J. Xu, C. Qiao, J. Li, G. Xu, "Efficient burst scheduling algorithms in optical burst-switched networks using geometric techniques". IEEE Journal in Selected Areas of Communication, Volume 22(9):17961811, November, 2004.

[70] (2014) Return on Investment, [Online]. Available: http://en.wikipedia.org/.

[71] (2011) VMware Virtualization Technology, [Online]. Available: www.VMware.com.

[72] (2011) Xen Virtualization Technology, [Online]. Available: http://xen.org.

[73] T. Püschel, D. Neumann, "Management of Cloud Infrastructures: Policy-based revenue optimization". International Conference on Information Systems (ICIS 2009), pages 1025, 2009.

[74] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, L. Stoica, M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing". EECS Department, University of California, Berkeley, 2009.

[75] B. Rochwerger, D. Breitgand, E. Levy, A. Galis, K. Nagin, I. Llorente, R. Montero, Y. Wolfsthal, E. Elmroth, J. Caceres, M. Ben-Yehuda, W. Emmerich, F. Galan, "The RESERVOIR Model and Architecture for Open Federated Cloud Computing". IBM Journal of Research and Development, 2009.

[76] Y. Collette, P. Siarry, "Multiobjective optimization: principles and case studies". Springer Verlag, pages 25-30, 2003.

[77] L. O., Burchard, "Analysis of data structures for admission control of advance reservation requests". IEEE Transaction, pages 413-424, 2005.

[78] M. de Berg, O. Cheong, M. van Kreveld, M. Overmars, "Computational Geometry (3rd revised ed.)". Springer-Verlag, 2008.

[79] M. A. R. Gonzalez, P. Chronz, K. Lu, E. Yaqub, B. Fuentes, A. Castro, H. Foster, J. L. Rueda, A. E. Chimeno, "G-SLAM The Anatomy of the Generic SLA Manager". Wieder. P., Butler, J., Yahyapour R. (Eds.) Service Level Agreements For Cloud Computing, Springer-Verlag, pages 167-186, 2011.

[80] C. Clark, K. Fraser, S. Hand, J. G. Hansen, E. Jul, C. Limpach, I. Pratt, A. Warfield, "Live Migration of Virtual Machines". NSDI'05 Proceedings of the 2nd conference on Symposium on Networked Systems Design and Implementation, pages 273-286, 2005.

[81] F. Hermenier, "Entropy a Consolidation Manager for Clusters". In Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments, pages 41-50, 2009.

[82] A. Corradi, M. Fanelli, L. Foschini, "VM consolidation: A real case based on OpenStack Cloud". Future Generation Computer Systems, 2012.

[83] C. Kotsokalis, J. L. Rueda, S. G. Gomez, A. E. Chimeno "Penalty Management in the SLA@SOI Project". Wieder. P., Butler, J., Yahyapour R. (Eds.) Service Level Agreements For Cloud Computing, Springer-Verlag, pages 116-119, 2011.

[84] A. Greenberg, J. Hamilton, D.A. Maltz, P. Patel, "The Cost of a Cloud: Research Problems in Data Center". Net- works, Proc. ACM SIGCOMM Computer Comm. Rev., vol. 39, no. 1, pages 68-73, 2009.

[85] A. Khosravi, S. Garg, R. Buyya, "Energy and Carbon-Efficient Placement of Virtual Machines in Distributed Cloud Data Centers". 19th Intl Conf. Parallel Processing (Euro-Par 13), Vol. 8097, pages 317-328, 2013.

[86] R. N. Calheiros, R. Ranjan, A. Beloglazov, C. A. F. De Rose, R. Buyya, "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms". Software: Practice and Experience (SPE), Volume 41, Number 1, pages 23-50, ISSN: 0038-0644, Wiley Press, New York, USA, 2011.

[87] (2012) Apache Tashi, [Online]. Available: http://incubator.apache.org/projects /tashi.html.

[88] (2014) OpenNebula, Cloud Infrastructure Management, [Online]. Available: http://opennebula.org.

[89] (2014) OpenStack, Cloud Infrastructure Management, [Online]. Available: http://www.openstack.org.

[90] (2014) Apache CloudStack, Cloud Infrastructure Management, [Online]. Available: http://cloudstack.apache.org.

[91] (2014) OpenStack Nova API Woorea, [Online]. Available: https://github.com/woorea/openstack-java-sdk.

[92] (2014) Apache JCloud, [Online]. Available: https://jclouds.apache.org.

[93] J. Kennedy, A. Edmonds, V. Bayon, P. Cheevers, K. Lu, M. Stopar, D. Murn, "SLA-Enabled Infrastructure Management". Wieder. P., Butler, J., Yahyapour R. (Eds.) Service Level Agreements For Cloud Computing, Springer-Verlag, pages 271-287, 2011.

[94] (2013) Openshift, [Online]. Available: https://openshift.redhat.com/app/.

[95] (2013) Apprenda, [Online]. Available: http://docs.apprenda.com/.

[96] (2014) Erlang programming language, [Online]. Available: http://www.erlang.org/doc.html.

[97] G. Cicotti, S. D'Antonio, R. Cristaldi, A. Sergio, "How to Monitor QoS in Cloud Infrastructures: The QoSMONaaS Approach", Intelligent Distributed Computing VI, Springer, pages 253-262, 2013.

[98] C. Muller, M. Oriol, M. Rodriguez, X. Franch, J. Marco, M. Resinas, A. Ruiz-Cortes, "SLAMonADA: A platform for monitoring and explaining violations of WS-agreement-compliant documents". Principles of Engineering Service Oriented Systems (PESOS), pages 43-49, 2012.

[99] A. Kertesz, G. Kecskemeti, I. Brandic, "Autonomic sla-aware service virtualization for distributed systems". Parallel, Distributed and Network-Based Processing (PDP), pages 503-510, 2011.

[100] A. Mosallanejad, R. Atan, M. A. Murad, R. Abdullah, "A Hierarchical Self-Healing SLA for Cloud Computing". International Journal of Digital Information and Wireless Communications, 4(1): pages 43-52, 2014.

[101] A. Bertolino, A. Calabro, G. De Angelis, "A Generative Approach for the Adaptive Monitoring of SLA in Service Choreographies". 13th International Conference Web Engineering, pages 408-415, 2013.

[102] J. P. Magalhaes, L. M. Silva, "A Framework for Self-healing and Self-adaptation of Cloud-hosted Web-based Applications". IEEE International Conference on Cloud Computing Technology and Science, pages 555-565, 2013.

[103] P. Leitner, A. Michlmayr, F. Rosenberg, S. Dustdar, "Monitoring, Prediction and Prevention of SLA Violations in Composite Services". 2010 IEEE International Conference on Web Services, pages 369-376, 2010.

[104] V. C. Emeakaroha, T. C. Ferreto, M. A. S. Netto, I. Brandic, C. A. F. De Rose, "CASViD: Application Level Monitoring for SLA Violation Detection in Clouds". Proceedings of 36th IEEE International Conference on Computer Software and Applications (COMPSAC), pages 499-508, 2012.

[105] A. L. Freitas, N. Parlavantzas, J. L. Pazat, "A QoS assurance framework for distributed infrastructures". Proceedings of the 3rd International Workshop on Monitoring, Adaptation and Beyond, pages 1-8, 2010.

[106] V. C. Emeakaroha, I. Brandic, M. Maurer, S. Dustdar, "Low level metrics to high level SLAs-LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments". International Conference on High Performance Computing and Simulation (HPCS), pages 48-54, 2010.

[107] (2013) Esper, [Online]. Available: http://esper.codehaus.org.

[108] (2014) Storm, [Online]. Available: https://github.com/nathanmarz/storm.

[109] D. Liu, U. Kanabar, C. H. Lung, "A Light Weight SLA Management Infrastructure for Cloud Computing". 26th IEEE Canadian Conference Of Electrical And Computer Engineering (CCECE), 2013.

[110] (2014) Spring Java Messaging Service, [Online]. Available: http://spring.io.

[111] (2014) Apache ActiveMQ, [Online]. Available: http://activemq.apache.org/.

[112] (2014) Akka documentation, Actor Systems, [Online]. Available: http://doc.akka.io/docs/akka/2.3.2/general/actor-systems.html.

[113] N. Behrooz , S. de Boer. Frank, M. J. Mohammad, "The Future of a Missed Deadline". Coordination Models and Languages, Lecture Notes in Computer Science, Volume 7890, pages 181-195, 2013.

[114] P. M. Hasselmeyer, P. Koller, P. Wieder, "An SLA Re-Negotiation Protocol". 2nd Non Functional Properties and Service Level Agreements in Service Oriented Computing Workshop (NFPSLA-SOC 2008), Ireland, 2008.

Acronyms

API Application Program Interface. 3, 24, 77, 78

AWS Amazon Web Services. 4

BDD Binary Decision Diagram. 24, 28, 32, 35, 37–39, 105

BNF Backus Normal Form. 24

BU Basic Unit. 54–57

CAD Computer Aided Design. 28

CPU Central Processing Unit. 35, 43, 54, 60–62, 64, 67, 69, 70, 74, 80–83, 85, 98, 99, 101, 105

CRM Customer Relationship Management. 4

EC2 Amazon Elastic Compute Cloud. 4, 18, 26, 49

ERP Enterprise Resource Planning. 4

GB Gigabytes. 35, 54, 61, 64, 80

GHz Gigahertz. 35, 38, 61

GSLAM Generic SLA Manager. 60, 77, 78, 86

HEC SAP Hana Enterprise Cloud. 4

HPC High Performance Computing. 61

IaaS Infrastructure-as-a-Service. 3, 7, 10, 11, 17–20, 24, 26, 27, 30, 35, 40, 53, 56, 57, 60, 67, 70–74, 79, 80, 86, 90, 97, 99, 103–105

ID Identifier. 25

IDC International Data Corporation. 3, 4, 78

IEC the International Electrotechnical Commission. 4, 15, 41

IP Internet Protocol. 80

IPAC Infrastructure Provisioning and Adjustment Component. 60

IPOC Infrastructure Planning and Optimization Component. 60

ISM Infrastructure Service Manager. 60, 61, 77, 78

ISO the International Organization for Standardization. 4, 15, 18, 41

IT Information Technology. 4, 15, 20, 41, 57

ITMC IT & Medien Centrum. 61, 64

ITSM IT Service Management. 4, 9, 21, 89

JMS Java Messaging Service. 89

JVM Java Virtual Machine. 90, 100

kwh Kilowatt Hour. 83, 85, 86

MB Megabytes. 64, 77

MIPS Millions Instructions Per Second. 74, 80–83, 97–99

ms Millisecond. 77

NIST the National Institute of Standards and Technology. 2

OBDD Ordered Binary Decision Diagram. 28

OLA Operational Level Agreement. 4

OOP Object Oriented Programming. 90

OS Operating System. 26, 35

PaaS Platform-as-a-Service. 3, 9, 10, 20, 70, 71, 73, 78–80, 86, 104, 105

pCPU physical CPU. 80

PDM Performance Degradation due to Migrations. 81, 83

QoS Quality of Service. 4, 7, 10, 20, 25, 26, 43–46, 52, 57, 60, 67, 71–73, 76, 80, 91–93, 96–98, 104, 105

ROBDD Reduced Ordered Binary Decision Diagram. 27–34, 36–40, 103

ROI Return on Investment. 18, 46, 47, 52, 73, 81

S3 Amazon Simple Storage Server. 4

SaaS Software-as-a-Service. 3, 9, 17, 20, 70–73, 86, 105

SLA Service Level Agreement. 4–13, 15–18, 20, 21, 23–32, 34–37, 39–46, 49–53, 56–58, 60, 62–65, 67, 68, 70–73, 75, 77–105, 123

SLAM SLA Manager. 78, 79, 86, 91, 93

SLATAH SLA violation Time per Active Host. 80, 83

SLO Service Level Objective. 7, 17

SLR Service Level Requirement. 57

SME Small and Medium-sized Enterprises. 19

SOA Service Oriented Architecture. 2

SPI SaaS, PaaS, IaaS. 3, 4, 8

TB Terabytes. 80

TRS Term Rewriting System. 29–31, 33, 37–39, 103, 104

UC Underpinning Contract. 4

UML Unified Modeling Language. 35

UUID Universally Unique Identifier. 25

vCPU virtual CPU. 80

VLSI Very Large Scale Integrated. 28

VM Virtual Machine. 9, 10, 13, 20, 26, 35, 49, 53–61, 64, 69–71, 73–86, 93, 97–99, 104, 123

WS-Agreement Web Service Agreement. 23, 24

WWS Writable Working Sets. 69

XML Extensible Markup Language. 11, 23, 24

List of Tables

4.1 SLA terms and descriptions 31
4.2 Example clauses of an SLA template 40

5.1 Main site capacity 66
5.2 CPU core prices (€) 66
5.3 Satisfaction ratio of requests with different shifting steps 70

6.1 VM instance flavor subject to CPU, capacity, cost and profit 85

List of Figures

1.1 The cloud computing service categories and their characteristics . . . 4
1.2 Evolution of SLA lifecycle management from three steps to six steps . 7

2.1 Requirement diagram 17
2.2 Multi-domain resource provisioning 21

3.1 SLA model high-level overview 28

4.1 (a) The ROBDD of disjunction function, (b) Mutual exclusiveness of disjunction function but with semantically redundant 1-path in ROBDD, (c) Mutual exclusiveness of disjunction function without semantically redundant 1-path in TRS ROBDD, (d) ROBDD after path optimization 35
4.2 The ROBDD with 30 decision nodes and 12 1-paths by applying term rewriting and node optimization 41
4.3 The ROBDD with 21 decision nodes and 12 1-paths by applying path and node optimization 42
4.4 The number of nodes (a), 1-paths (b) and negotiation time (c) statistics of the initial ROBDD (blue), the TRS ROBDD (red) and the ROBDD after running BDD optimization algorithms (green) 43

5.1 Relationship of failure probability and quality factor 50
5.2 β^i_{max} for gaining enough profits and β^i_{min} for keeping failure rate low. (a) without intersection area. (b) with intersection area. 52
5.3 Resource request levels, and subcontractors with different capacities . 54
5.4 Price per unit and failure rates for subcontractors 55
5.5 Representation of a schedule with two reservations as a histogram (a) and using computational geometry (b). 59
5.6 Representation of a schedule with three reservations as a histogram (a) and using computational geometry (b). 59

5.7 Representation of a schedule with three already scheduled reservations and a requested reservation as a histogram (a) and using computational geometry (b). 60
5.8 Finding alternative solutions by moving the request point. The moving is indicated by diagonal arrows starting at **R4** pointing towards the upper left and the lower right, respectively (compare with Fig. 5.7(b)). 62
5.9 Function h in relation to β values 67
5.10 Experiment results 69
5.11 Fragment statistic 70
5.12 Resources utilization of all servers 71

6.1 Hierarchical structure of the services in different domains 76
6.2 Invest additional resource to the service by increasing β in floating area 78
6.3 Integration of the generic SLA manager and OpenStack 82
6.4 Sequence diagram for the negotiation between PaaS and the OpenStack SLA Manager 83
6.5 (a) Energy consumption, (b) SLA average violation, (c) Relationship between maximum beta and minimum beta, (d) Return of Investment 86
6.8 Comparison with other strategies in availability 88
6.6 (a-d) Relationship between SLA availability, energy and SLA performance. 90
6.7 (e-f) Optimal target host utilization for allocating the migrated VM(s) 91
6.9 (a) Comparison with other strategies in energy consumption. (b) Comparison with other strategies in SLA violation and penalty. 91

7.1 Decentralization of IaaS SLA management using actor system 94
7.2 Finite state machine diagram of QoS actor 96
7.3 Finite state machine diagram of SLA actor 97
7.4 Finite state machine diagram of SLA manager actor 98
7.5 Finite state machine diagram of customer actor 99
7.6 Autonomous SLA violation filtering framework 100
7.7 Negotiation and provision scenario 101
7.8 Violation handling in QoS module 102
7.9 Violation handling in SLA module 103
7.10 Violation handling in SLA manager module 104
7.11 Violation handling in customer module 105

B.1 Top-level SLA (Template) classes 130
B.2 UML diagram of BDD implementation 131
B.3 UML diagram of infrastructure planning and optimization component implementation 132
B.4 UML diagram of advanced reservation implementation 133

Part IV

Appendices

Appendix A

SLA Template XML Example

A.1 SLA Template Party

Listing A.1: SLA Template Party

```
<slasoi:UUID>OneVMTypeTemplate</slasoi:UUID>
<slasoi:ModelVersion/>
 <slasoi:Party>
  <slasoi:Text/>
  <slasoi:Properties>
  <slasoi:Entry>
   <slasoi:Key>http://www.slaatsoi.org/slamodel#gslam_epr</
       slasoi:Key>
   <slasoi:Value>http://localhost:8080/services/ISNegotiation?wsdl
   </slasoi:Value>
  </slasoi:Entry>
  </slasoi:Properties>
   <slasoi:ID>ID_OF_PROVIDER_PARTY_GOES_HERE</slasoi:ID>
   <slasoi:Role>http://www.slaatsoi.org/slamodel#provider</
       slasoi:Role>
 </slasoi:Party>

 <slasoi:Party>
  <slasoi:Text/>
  <slasoi:Properties>
  <slasoi:Entry>
   <slasoi:Key>http://www.slaatsoi.org/slamodel#gslam_epr</
       slasoi:Key>
   <slasoi:Value>http://localhost:8080/services/SWNegotiation?wsdl
   </slasoi:Value>
   </slasoi:Entry>
   </slasoi:Properties>
    <slasoi:ID>ID_OF_CUSTOMER_PARTY_GOES_HERE</slasoi:ID>
    <slasoi:Role>http://www.slaatsoi.org/slamodel#customer</
        slasoi:Role>
 </slasoi:Party>
```

A.2 SLA Template Variable Declaration

Listing A.2: SLA Template Variable Declaration

```
<slasoi:VariableDeclr>
 <slasoi:Text/>
 <slasoi:Properties/>
 <slasoi:Customisable>
 <slasoi:Var>VM_CORES_VAR</slasoi:Var>
  <slasoi:Value>
   <slasoi:Value>1</slasoi:Value>
   <slasoi:Datatype>http://www.w3.org/2001/XMLSchema#integer
   </slasoi:Datatype>
  </slasoi:Value>
  <slasoi:Expr>
  <slasoi:CompoundDomainExpr>
  <slasoi:Subexpression>
  <slasoi:SimpleDomainExpr>
  <slasoi:ComparisonOp>http://www.slaatsoi.org/coremodel#
      greater_than
  </slasoi:ComparisonOp>
  <slasoi:Value>
  <slasoi:CONST>
   <slasoi:Value>0</slasoi:Value>
   <slasoi:Datatype>http://www.w3.org/2001/XMLSchema#integer
   </slasoi:Datatype>
  </slasoi:CONST>
  </slasoi:Value>
  </slasoi:SimpleDomainExpr>
  </slasoi:Subexpression>
  <slasoi:Subexpression>
  <slasoi:SimpleDomainExpr>
  <slasoi:ComparisonOp>http://www.slaatsoi.org/coremodel#
      less_than_or_equals</slasoi:ComparisonOp>
  <slasoi:Value>
  <slasoi:CONST>
   <slasoi:Value>16</slasoi:Value>
   <slasoi:Datatype>http://www.w3.org/2001/XMLSchema#integer
   </slasoi:Datatype>
  </slasoi:CONST>
  </slasoi:Value>
  </slasoi:SimpleDomainExpr>
  </slasoi:Subexpression>
  <slasoi:LogicalOp>http://www.slaatsoi.org/coremodel#and
  </slasoi:LogicalOp>
  </slasoi:CompoundDomainExpr>
  </slasoi:Expr>
  </slasoi:Customisable>
</slasoi:VariableDeclr>
```

A.3 SLA Template Agreement Declaration

Listing A.3: SLA Template Agreement Declaration

```
<slasoi:AgreementTerm>
 <slasoi:Guaranteed>
 <slasoi:Text/>
 <slasoi:Properties/>
 <slasoi:State>
  <slasoi:ID>VM_CORES</slasoi:ID>
  <slasoi:Priority xsi:nil="true"/>
  <slasoi:Constraint>
  <slasoi:TypeConstraintExpr>
   <slasoi:Value>
   <slasoi:FuncExpr>
 <slasoi:Text/>
 <slasoi:Properties/>
  <slasoi:Operator>http://www.slaatsoi.org/resources#vm_cores
  </slasoi:Operator>
  <slasoi:Parameter>
  <slasoi:ID>VM_X</slasoi:ID>
  </slasoi:Parameter>
  </slasoi:FuncExpr>
  </slasoi:Value>
  <slasoi:Domain>
  <slasoi:SimpleDomainExpr>
  <slasoi:ComparisonOp>http://www.slaatsoi.org/coremodel#equals
  </slasoi:ComparisonOp>
  <slasoi:Value>
  <slasoi:ID>VM_CORES_VAR</slasoi:ID>
  </slasoi:Value>
  </slasoi:SimpleDomainExpr>
  </slasoi:Domain>
  </slasoi:TypeConstraintExpr>
  </slasoi:Constraint>
 </slasoi:State>
 </slasoi:Guaranteed>
</slasoi:AgreementTerm>
```

Appendix B

UML Diagrams

B.1 UML Diagram of SLA (Template) Model

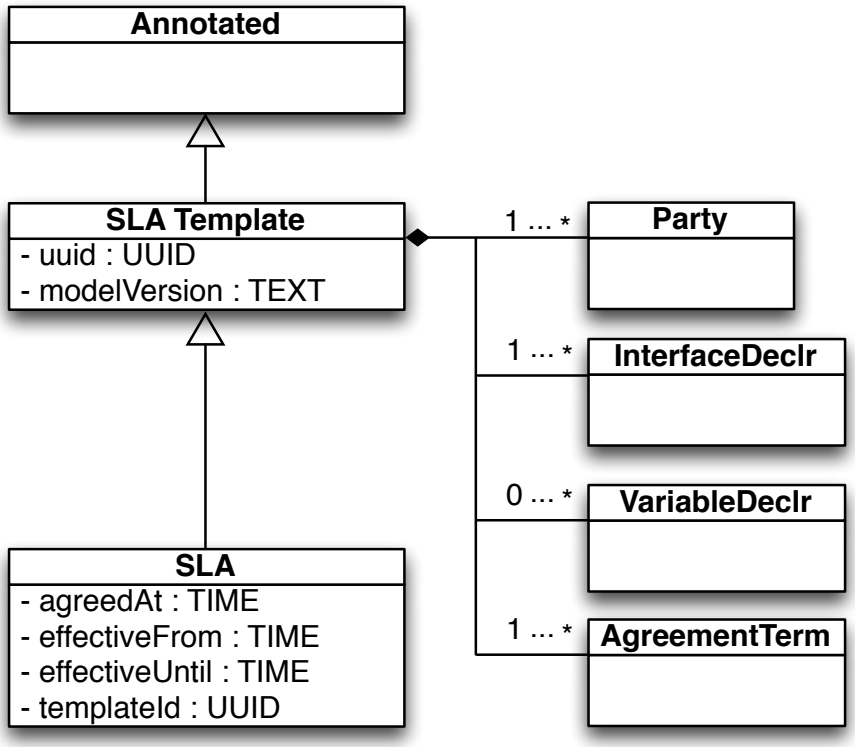


Figure B.1: Top-level SLA (Template) classes

B.2 UML Diagram of BDD Implementation

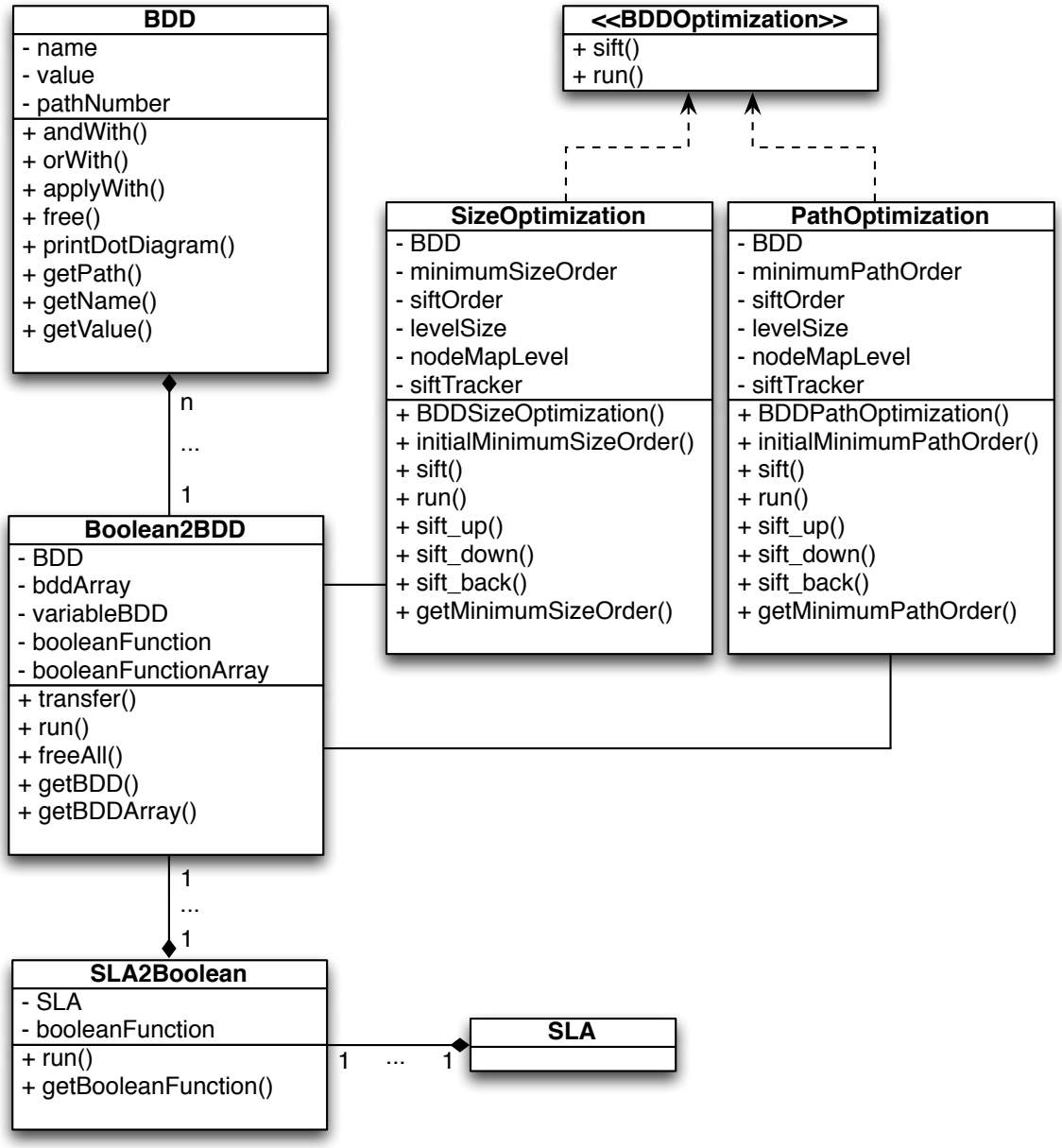


Figure B.2: UML diagram of BDD implementation

B.3 UML Diagram of Infrastructure Planning and Optimization Component Implementation

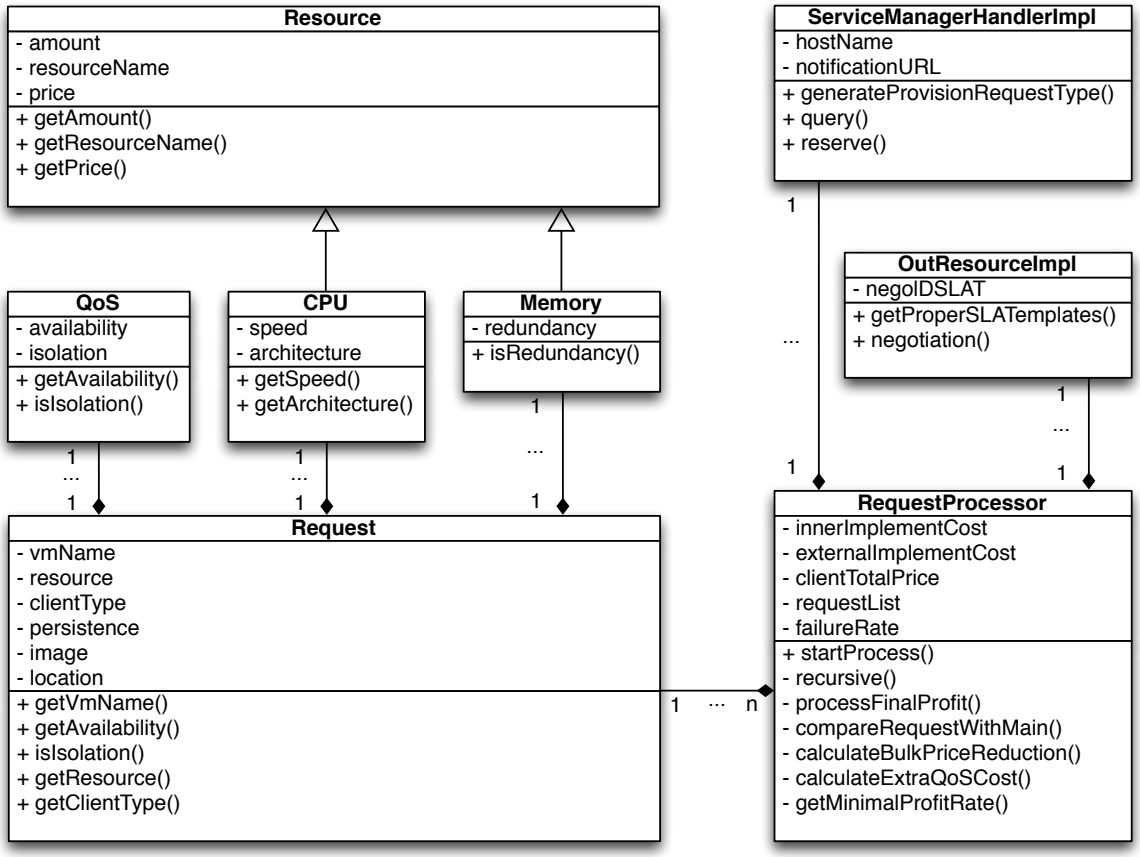


Figure B.3: UML diagram of infrastructure planning and optimization component implementation

B.4 UML Diagram of Advanced Reservation Implementation

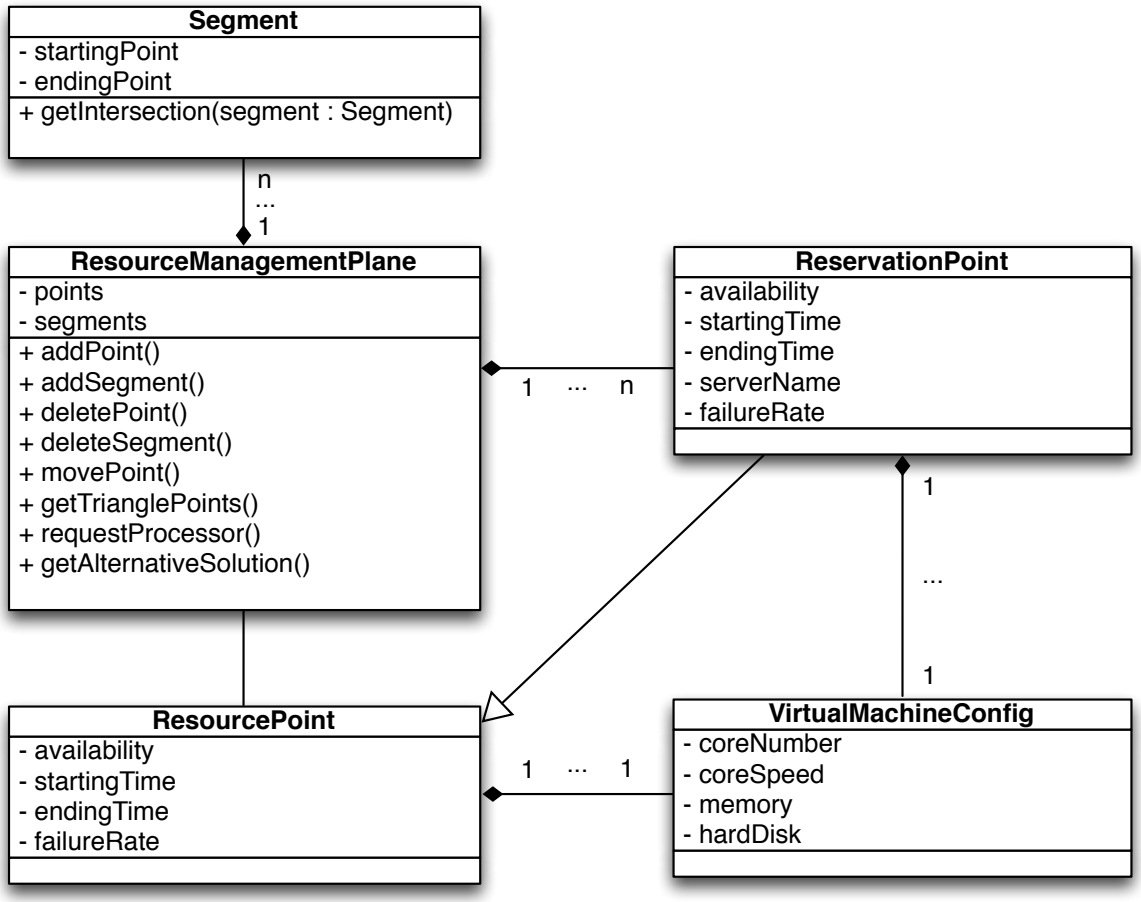


Figure B.4: UML diagram of advanced reservation implementation