UNIVERSITAT POLITÈCNICA DE VALÈNCIA

DSIC
DEPARTAMENT DE SISTEMES INFORMÀTICS I COMPUTACIÓ

SLA-Driven Cloud Computing Domain Representation and Management

Philosophiæ doctor in Computer Science dissertation

Author: Andrés García García
Advisor: Ignacio Blanquer Espert
Advisor: Germán Moltó Martinez

EDITORIAL
UNIVERSITAT POLITÈCNICA DE VALÈNCIA

February 27, 2014

Acknowledgements

I would like to thank my family and friends for the support given in the years of the making of this Ph.D. Thesis.

Summary

Service Level Agreements (SLA), as well as all its concerning facets such as SLA definition language, negotiation, monitoring, etc., have been subject of research for years, but the advent of Cloud computing, and the need of means for defining and ensuring Quality of Service (QoS) levels have greatly increased the interest on these developments. As the size and complexity of Cloud systems increases, the manual management of these platforms becomes a challenging issue. Therefore, the automation of large scale systems management is another promising feature of the SLA-driven autonomic Cloud solutions. Additionally, SLA-driven Clouds need mechanisms to represent, store and retrieve the information related to their particular domain. Usually the domain changes between different platforms, so custom models are built to capture this information and ad-hoc implementations are used to store and retrieve it.

This Ph.D. Thesis contributes to these topics by proposing a generic methodology for the representation of the domain in Cloud solutions. This methodology uses the WS-Agreement specification for capturing and manipulation arbitrary domain information using SLA fragments. SLA fragments are parts of SLA documents that describe a single computational element, and are composed on the fly in response to user request to generate a complete SLA document. This methodology provides the generality, extensibility and flexibility to unify the modeling of the domain in arbitrary Cloud services. A SLA composition algorithm enables a prototype implementation of the methodology in Cloudcompaas, a SLA-driven Cloud framework that manages the complete resource (e.g. Virtual Machines, software, services) lifecycle. This framework features an extension of the WS-Agreement SLA specification, tailored to the specific needs of Cloud computing. In particular, Cloudcompaas enables Cloud providers with a generic SLA model to deal with higher-level metrics, closer to end-user perception, and with flexible composition of the requirements of multiple actors in the computational scene. Moreover, Cloudcompaas provides a framework for general Cloud computing applications that dynami-

cally reacts to changes on Cloud infrastructures to correct QoS level guarantee violations.

The two major contributions of this Thesis are a generic methodology for the description of Cloud services, and the architecture, design and implementation of a SLA-driven Cloud framework. A use case provides a quantitative measure of the utility provided by the methodology from a Cloud user and Cloud provider point of view, reducing the price and increasing the ratio of users served. The effectiveness of the framework is demonstrated through the simulation of several realistic workload profiles, where Cloudcompaas achieves minimum price and maximum utility under highly heterogeneous utilization patterns.

Resumen

Los Service Level Agreements (SLA), así como sus aspectos asociados tales como lenguajes de definición de SLAs, negociación, monitorización, etc., han sido objeto de investigación durante años. Sin embargo, la popularidad del Cloud computing y la necesidad de proporcionar métodos para asegurar los niveles de calidad de servicio (QoS) en la misma han incrementado el interés en este área de estudio. Según aumenta el tamaño y la complejidad de los sistemas Cloud, la administración manual de los mismos se convierte en un reto. Por tanto la automatización de la administración de grandes sistemas es una de las aplicaciones más prometedoras de las SLAs aplicadas al Cloud. Adicionalmente dichos sistemas Cloud tienen la necesidad de representar y almacenar información concerniente a su dominio de aplicación. El dominio de aplicación es único y particular para cada plataforma, y por tanto modelos ad-hoc son utilizados usualmente para cumplir este objetivo.

Esta Tesis contribuye a estos retos proponiendo una metodología para la representación del entorno en plataforma Cloud. Esta metodología utiliza la especificación WS-Agreement para capturar y manipular la información del dominio mediante SLAs parciales. Las SLAs parciales son fragmentos de documentos SLA que son compuestos dinámicamente en respuesta a peticiones de usuarios, generando un documento SLA completo de forma dinámica. Esta metodología proporciona la genericidad, extensibilidad y flexibilidad necesarias para unificar el modelado del entorno en plataformas Cloud arbitrarias. Un algoritmo de composición dinámica permite la implementación de la metodología en Cloudcompaas, un framework Cloud dirigido por SLAs para la administración del ciclo de vida completo de recursos Cloud (e.g. máquinas virtuales, recursos software, servicios). Cloudcompaas incluye una extensión de WS-Agreement expecificamente diseñada para administrar despliegues Cloud. Cloudcompaas proporciona a los proveedores Cloud un modelo genérico de SLAs para la administración de métricas de alto nivel, con la composición flexible de los requisitos de usuario. Adicionalmente, Cloudcompaas propor-

ciona un framework general para la automatización del control de la calidad de servicio de recursos Cloud.

Las dos principales contribuciones de esta Tesis son una metodología genérica para la representatición de servicios Cloud, y la arquitectura, diseño e implementación de una plataforma Cloud dirigida por SLAs para la administración del ciclo de vida completo de recursos Cloud y el control automático del nivel de calidad de servicio de los mismos. Un caso de uso proporciona una medición cuantitativa del benefio obtenido por la metodología propuesta desde el punto de vista de proveedores, incluyendo el número de usuario servidos, y usuarios, incluyendo el precio del despliegue. La efectividad de la plataforma Cloud se demuestra mediante la simulación de varios perfiles de carga de usuarios realistas, donde Cloudcompaas logra minimizar precio y maximizar beneficios bajo distintos patrones de utilización.

Resum

Els Service Level Agreements (SLA), així com els aspectes associats a aquests, com ara llenguatges de definició d'SLA, negociació, monitoratge, etc., han sigut objecte d'investigació durant anys. Això no obstant, la popularitat de la informàtica en núvol (cloud computing) i la necessitat de proporcionar mètodes per a assegurar-hi els nivells de qualitat de servei (QoS) han fet créixer l'interès en aquesta àrea d'estudi. A mesura que augmenten la grandària i la complexitat dels sistemes en núvol, l'administració manual d'aquests esdevé un repte. Per tant, l'automatització de l'administració de grans sistemes és una de les aplicacions més prometedores de les SLA aplicades a la informàtica en núvol. Addicionalment, aquests sistemes en núvol tenen la necessitat de representar i emmagatzemar informació concernent al seu domini d'aplicació. El domini d'aplicació és únic i particular per a cada plataforma, i per tant, habitualment s'usen models ad-hoc per a assolir aquest objectiu.

Aquesta tesi contribueix a donar resposta a aquests reptes proposant una metodologia per a la representació de l'entorn en plataforma en núvol. Aquesta metodologia fa servir l'especificació WS-Agreement per a capturar i manipular la informació del domini mitjançant SLA parcials. Les SLA parcials són compostes dinàmicament en resposta a peticions d'usuaris. Aquesta metodologia proporciona el caràcter genèric, l'extensibilitat i la flexibilitat que calen per a unificar el modelatge de l'entorn en plataformes en núvol arbitràries. Un algorisme de composició dinàmica permet la implementació de la metodologia en Cloudcompaas, un framework en núvol dirigida per SLA per a l'administració del cicle de vida complet de recursos en núvol. Cloudcompaas inclou una extensió de WS-Agreement dissenyada específicament per a administrar desplegaments en núvol. Cloudcompaas proporciona als proveïdors en núvol un model genèric d'SLA per a l'administració de mètriques d'alt nivell, amb la composició flexible dels requisits d'usuari. Addicionalment, Cloudcompaas proporciona un marc general per a l'automatització del control de la qualitat de servei de recursos en núvol.

Les dues principals contribucions d'aquesta tesi són: una metodologia genèrica per a la representació de serveis en núvol; i l'arquitectura, el disseny i la implementació d'una plataforma en núvol dirigida per SLA per a l'administració del cicle de vida complet de recursos en núvol i el control automàtic del nivell de qualitat de servei d'aquests recursos. Un cas d'ús proporciona un mesurament quantitatiu del benefici obtingut per la metodologia proposada des del punt de vista dels proveïdors, incloent-hi el nombre d'usuari servits, i també des dels usuaris, incloent-hi el preu del desplegament. L'efectivitat de la plataforma en núvol es demostra mitjançant la simulació de diversos perfils de càrrega d'usuaris realistes, en què Cloudcompaas aconsegueix minimitzar el preu i maximitzar els beneficis sota diferents patrons d'utilització.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

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 [1]. Cloud computing is currently being used to tackle challenging problems in different application domains, such as industry, science, and government [2][3][4][5]. Cloud and other related technologies and concepts, such as Utility computing or Service-Oriented Infrastructures (SOI) are becoming widespread in the Information and Communications Technology field. Now that the infrastructure providers have a mature market and Cloud standards are progressively emerging, such as Open Cloud Computing Interface (OCCI) [6], Cloud Infrastructure Management Interface (CIMI) [7], Cloud Data Management Interface (CDMI) [8], etc., research in Cloud computing is focusing on improving the access and exploitation of Cloud resources.

Buyya et al. [9] were one of the first to focus attention on the role of Cloud computing to deliver a sustainable, competitive and secure computing utility. They propose Service Level Agreements (SLAs) as the vehicle for the provision and management of resources and the definition of Quality of Service (QoS) guarantees. A SLA is a

formal contract between providers and consumers, which defines the resources, QoS, obligations and guarantees in the delivery of a specific service. In the context of Cloud computing, SLAs are machine readable documents automatically managed by the provider. Therefore SLAs are widely regarded as a key feature for the future development of Cloud platforms. However, the application of SLAs for Grid and Cloud systems has many open research lines. One of these challenges, the definition of Cloud services, lies at the core of the objectives of this Thesis.

Cloud services are the products and solutions that are delivered by a Cloud platform. Cloud services can be described by their static and dynamic features. Static features are the ones that describe the service at deployment time. Common static features are resources (CPU, memory, software) and dependences between resources. Dynamic features are the ones that describe the service at execution time. Common dynamic features are QoS rules and the price of resources.

One of the current challenges in Cloud computing is to provide a unified technique for the definition of Cloud services, including its static and dynamic features. This definition should also be generic and extensible to include arbitrary Cloud services and features.

1.1 Motivation

Research projects have made advances in different facets of SLA for Grid or Cloud systems, focusing only on the development of specific features such as resource provision [10][11], negotiation [12][13], monitoring [13][14], reporting of guarantee violation [15][16], etc. One feature that is common to the development of SLA-driven Cloud platforms is how the resources, dependences and QoS rules are represented [17].

However, the advances in techniques for the definition of Cloud services have been limited. Usually each research project defines Cloud services using ad-hoc representations. Available Cloud standards such

as CIMI [7] and CDMI [8] focus only on the static features of Cloud services (resource definition) and describe a small subset of them.

This Thesis introduces a novel methodology for the representation of Cloud services in SLA-driven Cloud platforms. This approach consists on using SLA fragments that define their features and relationships, and that can be composed to produce complete SLAs. The benefits provided by this approach are the following.

- it provides a general methodology for the representation of Cloud services;
- it is language-agnostic;
- it makes the Cloud platform self-contained;
- it improves the transparency of the SLAs.

This methodology can be applied to any scenario and needs no consideration of ad-hoc solutions. It is formulated in terms of generic SLAs and their general features. It does not restrict itself to any SLA specification, and as such can be used with any of them. Representing Cloud services inside the SLA elegantly integrates all the information needed by the platform in a single system. This integration reduces the dependency on external components, reduces the complexity of the system and improves its flexibility. Finally, this methodology can be used to include in a single document all the information involved in the deployment, providing the user with a transparent vision of the service.

These arguments highlight the benefits of SLAs. However, our methodology proposes using SLA fragments that must be composed into complete SLAs. This raises the question about what are the advantages of SLA fragments that require an algorithm to be composed on-the-fly, rather than using predefined templates. A methodology that uses SLA fragments and composition has the same expressive power than an equivalent methodology that uses complete SLAs. Indeed, it is possible to generate any possible SLA supported by the system a-priori. However, a method that uses predefined SLAs also needs an algorithm

for the generation of these documents. The difference between both models is whether the composition is made dynamically at runtime, or statically at deployment time. On the ground that both representations are essentially equivalent, it can be argued that the convenience of using SLA fragments and composition over using static SLAs is qualitative, not quantitative.

Therefore, the SLA composition approach provides the following advantages over the static one.

- reduces operational and maintenance expenses;
  - SLA template documents are generated on the fly, and hence there is no need to explicitly store them;
  - generation, modification and elimination of elements are simplified. Changes to a single element imply changes to a single template.
- each element is self-contained. Each SLA fragment is defined considering its features and its restriction with other elements;
- improves flexibility. The number of templates available is not restricted to a set of predefined templates, but SLAs can be produced on-the-fly based on user requirements;
- implements a system suitable for the realization of Cloud markets. The ability of defining elements independently of each other, and compose them at execution time provides the foundation of a Cloud market. Different agents can expose their resources in a decentralized fashion and users can search for combinations of resources that satisfy their requirements.

1.2 Objectives

The first objective of this Thesis is to provide a methodology for the representation of Cloud services. The methodology uses SLA fragments to describe the features of a Cloud service, and SLA composition to produce complete Cloud services. The methodology is generic and extensible and can be used to describe arbitrary Cloud resources, QoS rules, etc.

The major contributions of this objective are the following.

I proposing SLAs as an unified representation of Cloud services;

II specification of a methodology for the representation of the Cloud services using SLA fragments;

III design of an algorithm for the composition of SLAs.

The second objective is to support the static and dynamic features of Cloud services using the proposed methodology. This objective implies the design and implementation of a SLA-driven Cloud framework that follows the methodology. To this end, this Thesis implements Cloudcompaas [18], a SLA-driven Cloud computing framework. Cloudcompaas covers all the steps involved on the management of SLAs, from the set-up of the SLA with the final user, feeding the SLA into the Cloud provider and interacting with the manager that allocates the required resources in the infrastructure, to the monitoring of the SLA and performing the necessary actions in order to maintain the QoS levels specified in the SLA.

Cloudcompaas is based on standards, such as the WS-Agreement [19] specification, for defining the SLAs, and on open-source initiatives, such as the WSAG4J [20] framework, for implementing the prototype. In this Thesis, the WS-Agreement specification has been tailored to meet the needs of Cloud computing, and the WSAG4J framework has been extended and adapted to deal with the complete lifecycle of the SLA, as well as with other requirements that are specific to the domain.

The major contributions on this line of work are the following.

I a SLA-driven architecture for the automatic provision, scheduling and allocation of Cloud resources (static features);

II a SLA-driven architecture for the assessment of QoS rules and self-management operations (dynamic features);

III a model of Cloud resources;

IV the implementation of the SLA composition algorithm;

V Cloudcompaas, an open source implementation of a SLA-driven Cloud framework for the management of Cloud services.

The third objective is the evaluation of the advantages of the proposed methodology over other approaches. A set of experiments have been performed to evaluate the utility achieved by the methodology regarding the static and dynamic features of Cloud services compared with other alternatives.

The major contributions on this line of work are the following.

I demonstrate the capabilities of the prototype by the resolution of a use case;

II demonstrate the utility achieved by using the proposed methodology to define Cloud resources versus using fixed resource definition (static features);

III demonstrate the utility achieved by using the proposed methodology to define QoS rules versus not using QoS rules (dynamic features).

1.3 Method

1.3.1 Cloud service representation methodology

The definition of the methodology begins with a survey of the state of the art and recent developments on this field. The aim of this survey is to identify projects that have dealt with resource modeling and SLA composition. These projects will help to outline the proposed methodology. Once the survey is done, the knowledge retrieved from other projects can be used to develop the specification of the methodology. The methodology must be generic enough to account for any case that may appear on its domain of application. Also, it must solve all the problems identified on the first stage. The expected outcome of this task is the specification of a methodology for the representation of Cloud services. This methodology includes the modeling of Cloud resources and the representation of QoS rules.

1.3.2 SLA-Driven Cloud framework

In order to implement Cloudcompaas, it is necessary to review the currently available SLA specifications and languages. Once a SLA specification and language is chosen, it is imperative to study its structure and organization. On this phase, the parts of the specification that are relevant to the Thesis are identified. It is important to define to which extent Cloudcompaas will support the SLA specification. After the SLA specification is chosen, the next step is to define the Cloudcompaas architecture and components design. Supporting SLAs in the system requires the arrangement of the components and their interfaces. After the architecture and design have been specified, a prototype implementation will be developed.

The outcome of this task is the working prototype of a SLA-driven Cloud framework that assesses the complete lifecycle of Cloud resources. The framework models the Cloud domain using the proposed methodology and includes the implementation of the SLA composition algorithm.

1.3.3 Composition algorithm

The modeling methodology requires a composition algorithm that can compose SLA fragments. This algorithm will be tailored to the particular requirements of the modeling methodology, although concepts found during the state of the art survey may be useful. After the algorithm design is done, its implementation must be integrated with the Cloudcompaas framework. The algorithm will be included inside a module of the Cloudcompaas framework. It will also be coupled with a user interface, so that users will be able to retrieve SLAs.

The outcome of this task will be the design and implementation of an algorithm that enables the composition of SLA fragments. This algorithm supports the modeling methodology defined on the previous task.

1.3.4 Experimental evaluation

The last stage consists on the experimental evaluation of the methodology, models and tools developed in the previous steps. A set of experiments has been designed to measure the utility achieved by the different objectives of the thesis.

The outcome of this task is an experimental evaluation of the performance of the proposed methodology and algorithm.

1.4 Use case

This section introduces a use case to illustrate the motivation of Cloud service representation.

A group of developers want to migrate a web service to the Cloud to benefit from the high availability and scalability of this technology. The web service is implemented in Java, so it depends on the Java environment. It also requires a large quantity of memory to run properly. Since the service is stateless, it can easily scale to serve more users.

Developers need a mechanism to define the static (cloud resources, dependences, etc.) and dynamic (QoS rules, restrictions, etc.) features of their deployment in the Cloud.

The static features of this use case are the resources and dependences defined explicitly and implicitly in the problem. These resources are the web service, runtime, infrastructure and the user that deploys it. The dependences are the requirement of the web service regarding the runtime and memory of the machine.

The dynamic features of this use case are the elasticity rules that govern the scaling of the web service.

This use case is elaborated in following sections to introduce the requirements of the problem, implementation details of the model and experimental evaluation of the prototype. This example is used to demonstrate the capabilities of the proposed methodology to define Cloud services.

Chapter 2

State of the art

This chapter introduces the state of the art in several topics relevant to the Thesis. Each section offers an historic perspective leading up to the latest developments on each topic. This discussion is presented from the more general, such as Cloud computing and Service Level Agreements, to the more specific, such as QoS assessment in Cloud computing, Cloud SLAs, Cloud monitoring systems and Cloud service representation.

2.1 Cloud computing

The term Cloud computing appeared for the first time in the year 2001, when John Markoff published an article in the New York Times [21]. Markoff refers to the strategy of Microsoft with the .NET platform as Cloud computing, enabling access to services independently of the client device. However, the first use of Cloud computing with a meaning closest to today definition was on August 2006, in a Google conference [22] when Eric Schmidt said the following.

"[...] there is an emergent new model [...] It starts with the premise that the data services and architecture should be on servers. We call it cloud computing - they should be in a "cloud" somewhere. [...] it

doesn't matter whether you have a PC or a Mac or a mobile phone [...] you can get access to the cloud. [...] the computation and the data and so forth are in the servers."

This sentence describe the fundaments of Cloud computing. Work is done on servers and client connect to an abstract cloud to interact with them, independently of the device. Eric Schmidt cites companies such as Amazon, IBM, Microsoft and Google as Cloud computing providers. On August 2007, John Markoff wrote an article on Cloud computing [23], defining the purpose of this technology as follows.

"[...] moving computing and data away from the desktop [...] simply displaying the results of computing that takes place in a centralized location and is then transmitted via the Internet"

This article again includes clients, servers and Internet connectivity. However, under this loose definition, some authors (such as Princeton Computing in the Cloud workshop [24]) list as Cloud computing P2P programs, large web portals that make use of datacenters such as eBay [25], online games, e-mail servers, etc. This criteria is confusing, and does not help to distinguish Cloud computing as an emerging technology providing innovative solutions.

It is on this context when Amazon introduced Simple Storage Service (S3) and Elastic Compute Cloud (EC2). These two services offer storage and computing capabilities in a pool of Amazon servers. These services are paid in a pay-as-you-go subscription, with a price per GB stored (S3) or hour of running time (EC2). Around that time other paradigmatic Cloud services went live, such as Google Docs [26] and Google App Engine [27]. These services enable a more refined of the Cloud computing definition, identifying it as an evolution of the client-server paradigm.

In August 2008 David Chappell, in a business oriented Technical Report, refers to Cloud computing as follows.

"Cloud platforms. As its name suggests, this kind of platform lets developers write applications that run in the cloud, or use services provided from the cloud, or both."

The novelty of this definition is that the Cloud is not only a client-server platform, but an environment that hosts services on demand. Chappell makes a distinction between applications in the Cloud and applications for the Cloud. Later in the same report Chappell explains the three models he identifies in Cloud computing: Cloud platforms, added functionalities and Software-as-a-Service (SaaS).

- cloud platforms: A Cloud platform is a service that enables building applications in and for the Cloud. Users can connect remotely to a development environment where they can build applications that can be accessed by other users;
- added functionalities: A desktop application communicates with Cloud services to provide new and improved capabilities;
- Software-as-a-Service (SaaS): Software applications that run entirely on the Cloud. Users access these services using lightweight interfaces.

SaaS is not a new concept. It refers to a software distribution paradigm, where applications are distributed as services instead of products. In software, a product must be bought, installed and finally used. In a service approach, software is hosted in a server and accessed through the network and used on demand. SaaS is part of the Utility computing paradigm. Utility computing advocates using computing resources as utilities, just like the power grid.

Up to this point the confusion surrounding the Cloud computing concept is prominent. As Cloud computing pioneers (led by Amazon) begin to create a relevant market volume, many companies and research groups want to capitalize on the hot topic. Some companies begin to market their products as Cloud computing, adapting terms to their interests and generating more confusion in the process. This trend is symbolized by Larry Ellison, CEO of Oracle, states in the Wall Street Journal [28]:

"The interesting thing about Cloud computing is that we've redefined Cloud computing to include everything we already do..."

On this context, the Hype Cycle 2009 report [29] by Gartner Inc. puts Cloud computing on the summit of the cycle. According to the analysts, Cloud computing was on the summit of popularity and expectative, but it would need another 2 to 5 years to get to the Plateau of Productivity, a period when a technology has matured and becomes productive.

Is around this time when the largest Cloud providers start appearing, either as branches or products of existing companies or as totally new ones. The most relevant IT companies begin to offer Cloud solutions, such as Windows Azure [30] (Microsoft), Google App Engine [27] (Google), Network.com [31] (Sun Microsystem), Blue Cloud [32] (IBM), etc.

Nevertheless, efforts were being made in eliminating the confusion surrounding Cloud computing and provide a formal definition. In February 2009 researchers in Berkeley publish Above the Clouds: A Berkeley View of Cloud Computing [33]. On this paper the authors expose the benefits that Cloud computing can provide to enterprises, such as use on demand, no capital investment, pay-per-use, as well as to providers, such as obtaining revenue from unused hardware. The paper also defines elasticity, the capacity of a Cloud system to react quickly and automatically to load changes, and discusses risks, opportunities, tradeoffs and previsions on the future of this technology. This paper is often considered a seminal work on Cloud computing, as the authors provide definitions for many concepts that would acquire relevance in the future. Specifically the authors define Cloud computing as follows.

"Cloud computing refers to both the applications delivered as services over the Internet and the hardware and systems software in the datacenters that provide those services. The services themselves have long been referred to as Software as a Service (SaaS), so we use that term. The datacenter hardware and software is what we will call a Cloud."

According to this definition developers can become SaaS providers while using Cloud platforms. This proposition sets an schematic view of Cloud computing as a layered system. This was in fact proposed

earlier in a less known paper. In December 2008, [34] tried to convey a Cloud computing definition. The methodology consisted in compiling 22 different Cloud computing definitions performed in 2008 and extracting the common elements, such as autonomic provisioning, virtualization, remote access, scalability, pay-per-use, etc. The paper also provides a comparison between Cloud computing and Grid computing. The Cloud computing definition stated by the paper is the following.

"Clouds are a large pool of easily usable and accessible virtualized resources (such as hardware, development platforms and/or services). These resources can be dynamically reconfigured to adjust to a variable load (scale), allowing also for an optimum resource utilization. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customized SLAs"

Beyond the basic definition, the paper also includes a categorization of Cloud computing on three models.

- *IP manage a large set of computing resources, such as storing and processing capacity. Through virtualization, they are able to split, assign and dynamically resize these resources to build ad-hoc systems as demanded by customers, the SPs. They deploy the software stacks that run their services. This is the Infrastructure as a Service (IaaS) scenario*;

- *Cloud systems can offer an additional abstraction level: instead of supplying a virtualized infrastructure, they can provide the software platform where systems run on. The sizing of the hardware resources demanded by the execution of the services is made in a transparent manner. This is denoted as Platform as a Service (PaaS)*;

- *Finally, there are services of potential interest to a wide variety of users hosted in Cloud systems. This is an alternative to locally run applications. An example of this is the online alternatives of*

typical office applications such as word processors. This scenario is called Software as a Service (SaaS).

Up to this point Cloud computing offers were assumed to be provided by a third party hosting services and providing access to computing capabilities. However, Cloud computing technologies provide significant advantages not only to users, but also to providers, such as server consolidation and efficient hardware utilization. These advantages led companies and research groups to become interested in applying Cloud technologies to their own computational resources, giving birth to the concept of private Cloud, opposed to public Cloud. The interest on this new perspective on Cloud computing led to the creation of research projects and tools oriented towards allowing organizations to deploy Clouds on their own computational resources. Such tools include Nimbus [35] (University of Chicago), Abiquo [36] (Abiquo), OpenNebula (Universidad Complutense de Madrid) and Eucalyptus (University of California).

Currently the most relevant and widely accepted definition of Cloud computing is the one provided by the National Institute of Standards and Technology (NIST) [1]. NIST defined Cloud computing as 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. This cloud model is composed of five essential characteristics, three service models, and four deployment models.

NIST lists five essential characteristics that any deployment must include to be considered Cloud. These characteristics are the following.

- on-demand self-service. Consumers must be able to automatically provision computing resources, with no human interaction required from the provider side;
- broad network access. Computing resources are accessed through the network by using standard mechanisms, independently of the client platform;

- resource pooling. Computing resources are pooled together, serving multiple users in a multi-tenant model, reassigning them dynamically based on the demand. These backend operations are transparent to the user, in the sense that these details are generally concealed to them;

- rapid elasticity. Resources can be dynamically provisioned and released, in some cases automatically, scaling up and down rapidly. This ability gives the user the illusion of unlimited capacity, adjusting the provision of resources to the system load;

- measured service. Resource usage can be monitored, controlled and reported at some level of abstraction relevant to the type of resource.

Clouds can be classified according to its service model and its deployment model. The service model defines the nature of the resources delivered and managed by the platform, while the deployment model defines how the platform itself is deployed and managed. NIST defines three basic service models that are traditionally considered the core Cloud computing service model. A variety of other models have been proposed by different players, but most of them are derivations of these three. The basic service models are the following.

- Infrastructure-as-a-Service (IaaS). The provided capability to the user is processing, storage, network and other fundamental computing resources. User has freedom to select the operating system and run arbitrary software on this hardware;

- Platform-as-a-Service (PaaS). The provided capability to the user is a runtime or environment targeted to a particular programming language or applications. User has freedom to deploy and run applications developed using languages, libraries or tools supported by the provider. Although the user does not have control over the underlying hardware configuration (CPU, memory, etc.), he may have control over configuration settings for the runtime;

- Software-as-a-Service (SaaS). The provided capability to the user is a ready-to-use software service hosted by the cloud platform. These services are accessible over the network using a variety of client devices. Users have no control over the underlying hardware or runtime configuration, although services may provide configurable settings.

The last feature of a Cloud platform is its deployment model. The deployment models of Cloud platforms have been a topic of discussion since the early attempts to produce a formal definition. The difficulty to establish the deployments models arise from the ambiguity of the definition of 'public'and 'private' use of resources, particularly in the case where several institutions are involved. NIST attempts to conceal the many variations on these topics by giving four deployment models generic enough to account for any particular instance of a Cloud platform.

- public Cloud: The Cloud platform is provisioned for the use by the general public. It may be owned and managed by a single organization, or a combination of them;

- private Cloud: The Cloud platform is provisioned for the exclusive use by a particular organization. It may be owned and managed by the organization itself, or by a third party;

- community Cloud: The Cloud platform is provisioned for the exclusive use by users from different organizations. It may be owned and managed by one or more of these organizations, by a third party or by any combination of them;

- hybrid Cloud: The Cloud platform is composed by two or more Cloud deployments, which remain independent from each other and communicate exchanging data and applications using standard or proprietary protocols.

In the last years Cloud computing has been identified as a technology of strategic relevance by the European Commission [37]. International initiatives have been established such as standardization committees

[6][38][8][7]. Almost every major player in the IT industry has active Cloud computing products, such as Google [27], Microsoft [30], Amazon [39], IBM [40], HP [41], etc, and a myriad of Cloud computing vendors have invaded the Cloud ecosystem, such as AppFog [42], Apprenda [43], CloudBees [44], Cloudera [45], Engine Yard [46], Heroku [47], Rightscale [48], Cloudscaling [49], GoGrid [50], Joyent [51], Rackspace [52], Savvis [53], Tier 3 [54], Citrix [55], etc. The large corpus of knowledge and expertise regarding Cloud computing is compiled in conferences and journals devoted to this topic as well as several books, such as Cloud Architecture Patterns [56], Web Services, Service-Oriented Architectures, and Cloud computing [57], Cloud computing Architected: Solution Design Handbook [58], The Cloud at Your Service [59], Cloud computing: Concepts, Technology & Architecture [60] and Cloud computing Bible [61].

2.2 Service Level Agreements

Service Level Agreements (SLAs) are formal contracts between service providers and service consumers that state assertions over the Quality of Service (QoS) of the delivery of a certain good. Specifically, SLAs may define the good to deliver, obligations of the parties, guarantees in the QoS level, warranties in case the expected quality is not met, mechanisms to monitor these metrics and so on. Traditionally SLAs have been represented as legal documents that explicitly state some service level guarantees (e.g. 99.99% uptime) and corresponding warranties for its violation (e.g. 10% discount in the service price). [62] propose SLAs as a mean for the definition of requirements and constraints of software resources, and the autonomic management of services. In order to realize this application of SLAs, a formal, machine readable definition of these documents is necessary.

Several specifications exists for SLA definition and management, with different levels of maturity and completeness. The major candidates to form the basis of the Cloudcompaas framework are the most relevant ones, WS-Agreement [19] and WSLA [63]. Other minor, less mature or popular alternatives have also been considered. WS-Agreement

is the SLA specification and a standard proposal of the Open Grid Forum (OGF). WS-Agreement defines itself a Web Services protocol for establishing SLAs between two parties using an extensible XML language. The specification consists of three parts: a schema for specifying SLAs, a schema for specifying SLA templates and a set of operations for managing the SLA lifecycle. Although the specification is defined as a Web Services protocol, its extensible nature has made WS-Agreement the language of choice for the SLA specification in many grid and cloud SLA projects [64]. Additionally, OGF provides WSAG4J, a WS-Agreement framework for the Java language. This framework includes almost all the elements of the specification and simplify the development of platforms adhering to the WS-Agreement specification.

The Web Service Level Agreement (WSLA) is a framework and specification developed by IBM for the definition and monitoring of SLAs in a machine readable format within the domain of web services. The WSLA language defines the parties involved in the SLA, the description of the service that the provider delivers to the consumer and the obligations of the SLA, where the guarantees and constraints of the SLA are defined. The WSLA framework is a tool for the SLA-driven management of the lifecycle of web services, using the WSLA specification. This framework integrates the usage of WSLA with other web services standards such as WSDL.

Other proposals for the definition of SLAs are the SLAng [65] and WSOL [66] languages. Both proposals are XML-based languages whose aim is to define QoS constraints in the domain of web services, and therefore are tightly related to the web services technologies and standards. Unlike the aforementioned proposals, these two specifications only define a language for the expression of QoS levels, but do not account for SLA lifecycle or assessment.

2.3 QoS assessment in Cloud computing

QoS in Cloud computing focuses on providing Cloud platforms with mechanisms to enforce the requirements of Cloud users. These studies have been made on two major delivery models, Cloud services and Cloud adapted workflows. Even though Cloud services are the focus of the offer of the major Cloud providers, the QoS assessment in Cloud workflows represents a more complex scenario that includes stricter requirements. Therefore the techniques and algorithms developed for Cloud workflows can be readily adopted for the Cloud services scenario.

An algorithm to select services that comply with user requirements from a pool of services is introduced in [67]. The paper proposes modeling the QoS parameters of services and enable users to query for services in different pools. The algorithm implements four stages to enable users to select a Cloud services according to QoS constraints. In order to retrieve a service, the Cloud user feeds the system with a description of the requirements, the QoS criteria and the importance of each criterion. The search step selects the services from the different pools that fit the description of the service, the filter step deletes the retrieved services that do not meet the QoS criteria of the user and the rank step sorts services according to the user preferences. The user is then able to select a service from a list, and the system proceeds to allocate the chosen resources. Finally, the update step updates the values of the user preferences for subsequent searches. Therefore when the user performs a new search, the system will rank the services according to the updated user preferences, enhancing the user experience.

A similar solution is proposed in [68] with the substantial difference that it considers chains of services instead of single services. The paper proposes a model in which services, grouped in pools, offers QoS guarantees, and users search for services indicating QoS criteria and preferences. Multiple services can be included in a single request, and the utility achieved by a service composition is the aggregation of the QoS levels of each service. The paper proposes LOEM, a QoS-aware

service composition algorithm, as a mean to achieve the best utility. The problem of finding the optimal combination of services is NP-hard. Therefore the approximation proposed by LOEM is to provide pareto-optimal solutions. The algorithm starts out by filtering out services that do not meet the required QoS criteria or provide poor utility. Then, the algorithm proceeds to select at most h ($0 < h < n$ with n the total number of services) candidate services by using local optimization methods. Finally, a complete enumeration of the combinations of the h candidate services is performed, and a mixed integer programming method is used to select the pareto-optimal solutions. The paper proposes a solution to the problem of finding a service chain that maximizes compliance with the user QoS criteria and maximizes certain utility function. Yet, it does not account for the stricter QoS requirements of workflows, where the QoS criteria are not asserted over individual services, but over the relationship between services.

In [69] the authors propose a methodology to enact workflows in stages, introducing a synchronization barrier between each stage. The synchronization barrier limits the resource load in each stage and thereby allows meeting the QoS levels specified by the user. However, even though it discusses to provide QoS guarantees in a Cloud infrastructure and mentions to dynamically adjust the resources assigned to each stage, these possibilities are not explored.

In [70] the authors perform a deep analysis of the QoS assessment in the delivery of Cloud services, and focus on very specific problems of the Cloud platforms. On the first place the authors propose SLAs as the vehicle for the QoS level specification. This formal document captures the requirements of an actor respect to another, and provides a common framework for the definition of QoS criteria, obligations and penalties. On the second place the authors model the transitive relationship between Cloud providers and Cloud users, and Cloud users as application providers and end-users as application consumers. Even though the existence of this relationship has been cited in [33], the paper analyses its impact in the QoS assessment. The most complex role is the one of the Cloud users, since they need to establish two SLAs, one with the Cloud provider and another with the end-user. On the third place the authors shift the user-centric view of QoS as-

sessment typical of scientific environments to a business centric view. This paper stress the aim of the provider on maximizing the profit while meeting the QoS criteria of users.

The authors propose four scheduling policies for the dispatch of user requests. The four algorithms are start VM, where a new request is attended by starting a new VM; wait, where a new petition is attended by queuing the request in an available resource; insert, where a new petition is attended by inserting the request in the queue of an available resource before existing petitions, with the condition that no SLAs are violated; and penalty delay, that proceeds as insert but allowing the violation of currently scheduled petitions. These four policies define a gradient in the ratio between user satisfaction and profit, from the first one, where almost every petition is attended at a high cost, to the last one where petitions are accepted even if other ones are violated. Based on these four scheduling policies, three admission control algorithms are introduced. An admission control algorithm is composed by several scheduling policies that are applied in order. The policies that achieve the highest profit are applied first, and if the profit is under a defined threshold, the next one is considered. If no policy meets the minimum profit requirement, the user petition is rejected. These three algorithms range from the most conservative, which only uses the wait and new VM policies, to the most aggressive including all four policies.

Finally, in [71] the authors introduce the architecture and working prototype of the WfMS platform for the QoS-aware execution of workflows. The platform captures the QoS criteria for the execution of workflows in SLA documents, and a monitoring system captures information related to these criteria from the running components and publishes it to an index. The architecture is composed by three major components. The workflow manager is the central component in charge of the coordination of the multiple components in the platform. The workflow enactor is the local component deployed in each Cloud infrastructure responsible of the execution of each workflow. The enactor is the component responsible of retrieving and comparing the monitoring data to the QoS criteria defined in the SLA. If the monitoring data asserts the violation of QoS criteria, the enactor generates

an event that is processed by the workflow enactor. The deployment and execution of a workflow in WfMS is composed of three phases.

On the publication phase the developer register services in the system using description documents. For each service the developers define their input and output interfaces, as well as their resource requirements. Using the registered services the developers can proceed to define a workflow document, specifying the involved services, their order of execution and QoS rules. Using this information the system generates a mapping from high level QoS criteria to low level parameters, as retrieved by the monitoring system. On the negotiation phase a user requests a workflow execution from the Cloud provider, using the template generated by the developer. Users can specify high-level requirements and rules according to these templates that will be automatically checked and translated to resource requirements from the Cloud infrastructure. Once evaluated, a cost is returned to the user, and if the offer is accepted, the workflow is deployed and a SLA is established between user and developer and between developer and Cloud provider. On the execution phase the user manually requests the execution of a deployed workflow on the Cloud. When an execution begins the platform allocates resources to the workflow and begins monitoring of the execution and evaluation of the QoS levels. The Evaluator is in charge of guarantee that the QoS level required by the user is met all along the execution. The execution lasts until the workflow ends or until it is manually stopped by the user.

This work differs from previous ones in the fact that it does not support a query operation for the retrieval of services or service chains. Services and workflows are registered by developers and end users manually specify the workflow to execute. On the other hand this work is similar to [70] in the sense that it models a three actor scenario with end-user, developer/cloud user and cloud provider, and establishes SLAs between them. This work improves the previous ones by providing a QoS-aware execution of complete workflows, not only of single services. Moreover it provides a dynamic QoS guarantee assessment. In the other projects the QoS is seen as a static feature that is met only at the dispatch of a Cloud services i.e. required computing power or estimated deadline. The WfMS platform however includes features of

the services and workflow that may change as QoS criteria (e.g. the frame rate of a video streaming application). This feature introduces the need of a dynamic QoS assessment mechanism that ensures that the user requirements are met all along the execution of the service.

2.4 Cloud SLAs

Earlier definitions propose SLAs as a mean for the definition of QoS constraints in electronic services [72]. Other works [62] propose SLA as a mean for the autonomic management of services. More recently these two concepts were brought together and SLA is used both for the definition of requirements and the automatic management of the complete lifecycle of resources. Significant advances have been made in the development of SLA-driven distributed computing systems. In particular, several innovative projects have considered SLA-driven automatic resource management in the last decade.

In [73] an architecture for the provisioning of on-demand virtualized services based on SLA is proposed. The authors define it as "the first attempt to combine SLA-based resource negotiations with virtualized resources in terms of on-demand service provision", and represents a first step in the line of automated SLA-driven Clouds systems. Further works deal with specific facets of SLA management, such as a system for the monitoring of low level metrics in distributed environments and its transformation to high level SLA parameters [74].

More recently [75] proposes an architecture for an SLA-oriented resource provisioning model for Cloud computing. This architecture is realized using the Aneka platform [76], a solution that enables QoS-driven resource provisioning for scientific computations, and provides mechanisms for the definition of deadline constraints and the incorporation of multiple Cloud resources.

Several European projects in the last years are related at different degrees to the SLA-driven management of resources and other topics covered by Cloudcompaas.

Reservoir [77] is a pioneering European project that enables providers to build their own virtualized Cloud infrastructures. Although Reservoir does not cover SLAs and dynamic management of resources, a number of spin-out technologies and derived projects aim to provide these capabilities, such as BonFIRE [78], Optimis [79] and 4CaaSt [80].

BonFIRE is a European project that provides a platform for the federation of Cloud deployments. It enables developers to deploy and manage Cloud services in a unified environment, including service metrics and monitoring. However BonFIRE does not use SLA to represent resources. It does not include QoS assessment, dynamic management of resources or resource scheduling.

Optimis is a European project that enables private Cloud to automatically interact with public Cloud providers, optimizing the usage of resources by means of Cloud federation, cloudbursting, live migration and autoscaling. Optimis performs scheduling operations by deciding the best provider to host resources. Optimis provides a domain-specific extension of WS-Agreement to specify requirements at IaaS level and constraints in Cloud services.

4CaaSt is a European project that provides a platform for the deployment, management and trade of Cloud services. It includes automatic scaling and management of resources, allows providers to federate their resources in a common marketplace and enables users to compose services. However this platform does neither include SLAs for the representation of resources, nor dynamic QoS management, nor scheduling operations. Also it mainly focuses on the PaaS level of Cloud.

SLA@SOI [81] has among one of its aims the implementation of a framework of tools and components that enables the creation of SLA-driven Service Oriented Infrastructures (SOI). As a large scale project, its developments span all the facets of SLA such as a SLA definition language, negotiation, monitoring, violation prediction and detection, etc. SLA@SOI has developed a methodology for the SLA-driven management of infrastructures and services, and encompasses activities such as dynamic service discovery and composition, service monitoring

and assessment, infrastructure planning and optimization etc. However this project does not consider Cloud computing infrastructures as their target platform, and hence it does not account for some specific needs of this field.

Cloud-TM [82] is a European project aimed to provide a data-centric PaaS middleware for the development of distributed Cloud applications. Its two major aims are to ease the development of Cloud applications by providing high level data management abstractions and to provide self-tuning mechanisms that optimize data operations based on user QoS constraints. The SLA system is based on SLA@SOI. However this project does not cover the PaaS and SaaS levels of Cloud computing, and is focused in data-centric Cloud applications, instead of general purpose Cloud computing.

Cloudscale [83] is a European project focused on offering a system to automatically scale Cloud applications with minimal human interaction. Although it covers extensively the upscaling and downscaling of application, this project does not use SLAs for the representation of resources. It does not account for the deployment and scheduling of resources and does not provide a general mechanism for the dynamic QoS management of resources.

PaaSage [84] is a recent European project whose aim is to build an Integrated development environment to enable designers and developers to automatically deploy and optimize Cloud services, provide runtime monitoring and dynamic adaptation, intelligent metadata retrieval, multi provider support, etc. Although this project covers several topics dealing with QoS assessment and dynamic management of resources, it does not use SLAs for the definition of resources and QoS rules nor cover all the levels of Cloud computing.

Finally Contrail [85] aims to federate Cloud resources by providing unified interfaces for accessing resources. It covers all three levels of Cloud by providing IaaS, PaaS and SaaS resources. It addresses several topics regarding SLAs such as architecture for SLA management, dynamic QoS assessment, monitoring, accounting and billing.

This section has presented an analysis of the state of the art in SLA management in Cloud computing environments. Although significant advances have been achieved in this field, there are several issues that require further developments. Particularly most of the presented solutions do not provide a generic and standard representation of resources, the do not account for the automatic provision and scheduling of resources and they do not account for the QoS assessment of resources.

2.5 Cloud monitoring systems

In the hardware monitoring field there is a number of well-established and widely adopted distributed monitoring tools. Nagios [86] is an open source tool for the monitoring of system metrics and network usage. Nagios provides mechanisms for the generation of reports in case a QoS violation is detected or filling information about the state of the infrastructure. Ganglia [87] is a distributed and scalable monitoring system for high performance and high throughput computing systems such as clusters and Grids. Ganglia uses efficient storage and communication methods to minimize the impact in the host node, enabling its use on infrastructures in the range of thousands of nodes. Both Ganglia and Nagios retrieve information from the resources by placing agents on the target nodes that are in charge of retrieving the value of the metrics of interest. It is possible to combine both tools to build a system able to monitor and infrastructure, generate reports and perform actions based on the state of the resources.

The tools seen so far deal with the monitoring of hardware resources in Grid computing or Cluster computing. There are other tools specifically designed for the monitoring of applications in addition to hardware resources. Zenoss core [88] is an open source tool for the monitoring and management of systems (applications, network, servers, etc.) that provides information about the performance and availability of resources. Zenoss offers a specific tool for the monitoring of Cloud resources, scalable to thousands of machines. Other available tools for

the monitoring of resources include Zabbix [89], FireScope [90], Munin [91] and Collectd [92].

Even though these tools are adequate for the monitoring of IaaS Clouds, they fall short at dealing with higher abstraction levels such as PaaS and SaaS. On these cases a "semantic gap" occurs between the representations of resources. PaaS and SaaS Cloud levels handle a different set of metrics than that of the Infrastructure level, since each one operates at a different abstraction level. Hence the expression of QoS requirements for each scenario is different, although related with each other. For instance, the high level QoS parameter "response time" is meaningful only at SaaS level. Yet, the parameter "response time" is related to the low level parameters "CPU load" and "memory load" that happen at IaaS level, and perhaps to other parameters at different abstraction levels. Therefore it is necessary to develop a monitoring system that faces the challenges raised by the Cloud computing paradigm. In order to deal with the aforementioned "semantic gap", three major approximations have been proposed. These approximations, in increasing order of complexity, are described following.

The first approximation, the simplest one, consists on the expression of the QoS requirements using low level metrics [93]. Using any of the existing infrastructure monitoring systems, these metrics are retrieved and the rules evaluated. This approximation is straightforward since both the monitoring system and the QoS requirements use the same parameters. The additional work in this scenario consists on the expression of QoS requirements that correctly model the expected behavior of the application using low level metrics. The definition of these rules requires extensive profiling of the application. The major advantage is that the Cloud platform is simplified, since currently available monitoring solutions are used. The major drawbacks are the extra effort required for the developer on the deployment of service, as each new application deployed requires profiling and the design of ad-hoc QoS requirements, and the little expressiveness allowed by the model.

The second approximation consists on expressing the QoS requirements using high level metrics, using a monitoring system to retrieve

low level metrics and include in the Cloud platform mechanisms that translates low level metrics to high level metrics, such as the LoM2HiS framework [74]. On this approximation developers express the QoS requirements in a natural way using high level parameters close to the domain of services, while the platform automatically translate these rules to the low level parameters monitored by the system. The LoM2HiS framework uses a rule-based approach to perform the translation of low level to high level metrics, applying a set of rules stored in the system. This approach requires the rules to be already defined in the system for every possible service to be deployed, and in fact this method is very close to the service profiling performed in the previous approximation. In [94] the authors propose the use of a knowledge system to leverage this problem, using past experiences of the system to automatically decide and adjust the value of the parameters.

The major advantage of this approach is that developers are alleviated from the effort of profiling their services and define low level rules for each one, focusing instead on expressing their requirements in term of metrics interesting for their domain. The major drawback is that the effort is shifted to the provider, considerably increasing the complexity of the platform due to the inclusion of the translation mechanism, and that the flexibility of this model is yet limited by the relation between high level and low level parameters.

The third approximation, the most complex one, consists on expressing the QoS requirements using high level metrics, and retrieving high level metric values from the monitoring system. On this case, as the first one, both requirements and monitoring information are expressed using the same metrics, and hence the evaluation of the QoS state is immediate. Additionally, as on the second case, developers can easily express their QoS requirements in a natural way using parameters close to their application domain. The complexity of this third approach lies in the monitoring system and the retrieval of high level monitoring information. [95] proposes a multilayer Cloud monitoring framework that retrieves high level metrics by instrumenting the deployed Cloud services. The rationale behind this model is that QoS constraints are usually defined for parameters relevant to each particular service, and only services themselves can retrieve the value of these parameters.

In order to obtain these metrics, the framework includes a publishing mechanism that is used by Cloud services to expose the relevant metrics, and this information is gathered and aggregated together with monitoring information at other levels (for instance hardware monitoring).

The major advantage of this approach is that it provides high expressiveness since it uses monitoring information at different abstraction levels to determine the QoS state. The major drawback is that it requires an effort from both developers and providers. Developers need to modify their services to run on the Cloud, but this may not always be possible as in the case of legacy code. On the other hand providers need to include an appropriate monitoring system to retrieve and aggregate the monitoring information at different levels.

These three options represent the major approximations to the Cloud monitoring field in the current literature. Any of these three options can be adopted in a Cloud deployment to provide information about the state of resources, or they can be combined together to produce a system with enough flexibility to server any possible user request in the most appropriate manner.

2.6 Cloud service representation

SLA@SOI define the domain of the platform [96] as "a large number of heterogeneous software, hardware and service elements inter-related through a complex set of relationships and dependencies". SLA@SOI utilizes the domain information for the planning and deployment of resources. The domain information is captured using a custom UML model of their use cases, which stores key information. Even though SLA@SOI modeling methodology enables autonomic distributed systems to store and retrieve information and establish relationship between elements, this method is not naturally extensible to other distributed environment such as Clouds, and cannot be generalized to other domains.

Some early works propose models representing and capturing this information, even if they do not explicitly refer to the domain modeling. [97] defines a hierarchy of elements and composition relationships. For instance, in this model a service can be composed by a set of resources, such as a web server and a database system. Further works introduce the concept of different entities and SLAs that define their relationship. [98] models a domain as a set of collaborating entities. A scheduler is able to collocate a set of resources from different providers, establishing a different SLA with each pair.

More recently, [99] proposes a model of the context similar to the domain modeling of SLA@SOI. The authors define the context as the information that represents the interaction between users and services, and argue that changes in the context produce changes in these relationships and hence these changes must be dynamically reflected in the SLA.

Therefore, most of the research projects use ad-hoc representations for modeling the domain. However, any system managing the complete lifecycle of cloud resources will need mechanisms to represent and retrieve information for the different stages of the process.

SLAs are more convenient to represent domain information than ad-hoc representations, since they provides a generic methodology that is language-agnostic, self-contained and transparent. However using SLAs requires covering a wide set of potential combinations of resources, entities and relations. A methodology that uses SLA fragments and composition of SLAs provides additional qualitative advantages, such as reduced operation and maintenance cost, as well as improved flexibility. This approximation to the domain modeling also enables the representation of individual independent elements in the domain, and provides the foundation for advanced applications such as Cloud markets.

Chapter 3

Design

This chapter introduces the design of the elements that realize the Cloud service representation and management methodology. Specifically, this chapter introduces the model of the resources used in Cloudcompaas, the design of the SLA fragments that are the foundation of the Cloud service representation methodology and the architecture of Cloudcompaas.

3.1 Representation of Cloud services using SLA

The first step to design the representation of Cloud services is the definition of the elements that populate the domain. Once the elements have been defined, the next step is to describe the relationships between them. For this purpose we capture the information of these relations using SLA. A simple approach is to model these relations as SLAs between pairs of elements. This approach has several drawbacks.

- it potentially needs a very large number of SLAs. The number of SLAs grows exponentially with the number of elements;

- it requires high maintenance and operational costs. The addition of new elements requires the generation and update of many SLAs;
- its flexibility is low. Static relations between fixed elements fail to capture changes in a highly dynamic environment.

This approach is unpractical for a public cloud involving many types of resources and a large number of services and users. For instance, using this approach, for each new Virtual Service registered in the framework, an SLA that combines that service with each available hardware configuration should be created. Also, changes in a Virtual Service would imply changes in each SLA.

In order to capture the information of the domain using SLAs and solving the aforementioned problems, our methodology proposes describing each element as a SLA fragment, and using a SLA Composition algorithm to produce complete SLA templates. An SLA fragment is a document that includes only a subset of the elements that define an SLA. Our methodology includes the following three elements for each SLA fragment.

- Service Terms, which describe static features. For instance, a VM describes its hardware configuration;
- Guarantee Terms, which describe the dynamic features. For instance, Quality of Service assurance to the consumer, penalties and rewards;
- Creation Constraints, which describe the constraints in the instantiation of each element, and its relationship with other elements. For instance, a service can impose some restrictions so that it can only be executed on machines with a certain amount of memory.

This methodology provides a novel and unique approach to this problem, and enables building arbitrary Cloud services. The next sections define in detail the resource model defined in Cloudcompaas for the

representation of resources, and the model of SLA fragments defined for the representation of Cloud services.

3.2 Resource model

The resource model defines the type of resources that will be represented in Cloudcompaas. Most of SLA specifications provide an extensible language for the definition of SLA documents that must be complemented by a domain-specific representation.

The resources represented by Cloudcompaas were derived from the uses case described in 1.4. Generalizing the use case requirements, we can define resources as belonging to the IaaS, PaaS, SaaS or user level.

3.2.1 IaaS

The data model of the Infrastructure level is depicted in Figure 3.1.

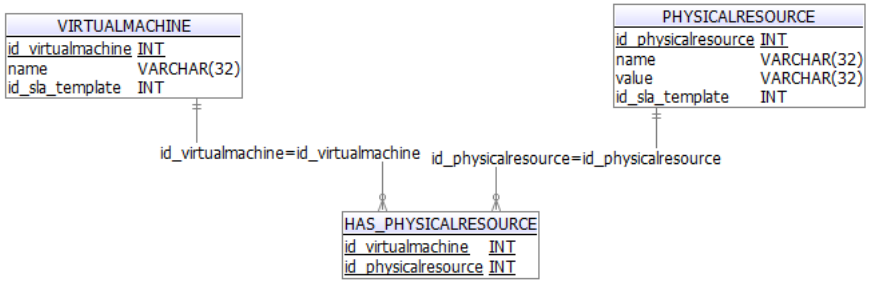

Figure 3.1: Infrastructure model for Cloudcompaas.

The basic building block of the IaaS level is the Virtual Machine. A Virtual Machine (VM) is the aggregation of hardware resources, or simply a hardware configuration.

This model has the following requirements.

- several Physical Resources of the same kind can exist, each one with a different value;
- the system can store Physical Resources not related to any VM;
- a VM can have an unlimited number of resources;
- the model comprises at least one resource from one of the following types: Cores, Memory, Network and Architecture.

The restrictions that our resource model set on a VM are the following.

- a VM must have exactly one resource of the following types: Cores, Memory, Network and Architecture;
- a VM cannot have two resources of the same type.

3.2.2 PaaS

The data model of the PaaS layer is depicted in Figure 3.2.

The basic building block of the PaaS level in Cloudcompaas is the Virtual Container [100]. A Virtual Container is a software stack composed by a hierarchy of components. The four level hierarchy model has been designed based on the requirements of the use cases.

This model has the following requirements.

- a hierarchy of software dependencies;
- a Virtual Container can have multiple Virtual Runtimes, and a Virtual Runtime can be associated to different Virtual Containers;
- a Virtual Runtime can have multiple Software Resources, and a Software Resource can be associated to different Virtual Runtimes;

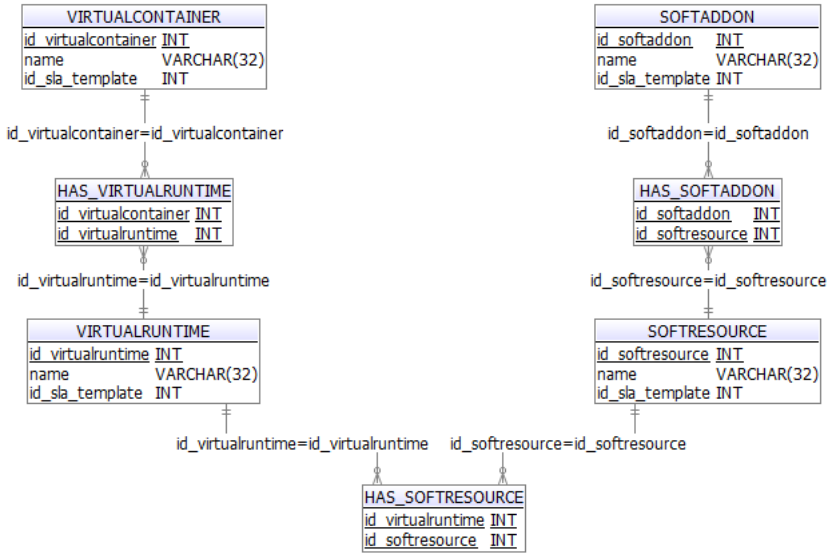

Figure 3.2: Platform model for Cloudcompaas.

- a Software Resource can have multiple Software Add-ons, and an Add-on can be associated to different Software Resources.

The restriction of the model is that a Virtual Container cannot have two Virtual Runtimes, Software Resources or Software Add-ons of the same type. Finally, a Virtual Container must have at least one Virtual Runtime.

3.2.3 SaaS

The data model of the SaaS level is depicted in Figure 3.3.

The building block of the SaaS level is the service, which has been renamed as Virtual Service in the data model. This name is coherent with the nomenclature used through this section, and it also helps to discern the specific services running inside a Cloudcompaas deployment from Cloud Services.

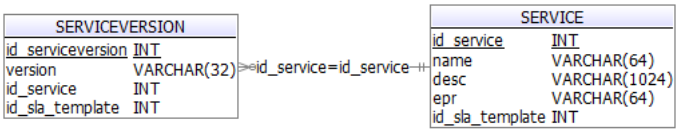

Figure 3.3: Service model for Cloudcompaas.

This model has the requirement that a Virtual Service can have multiple versions.

The restrictions of the model are.

- a Virtual Service must have at least one Service Version;
- a Virtual Service can have an unlimited number of Service Versions;
- a Service Version is related to exactly one Virtual Service.

3.2.4 Users

The data model of the user is depicted in Figure 3.4.

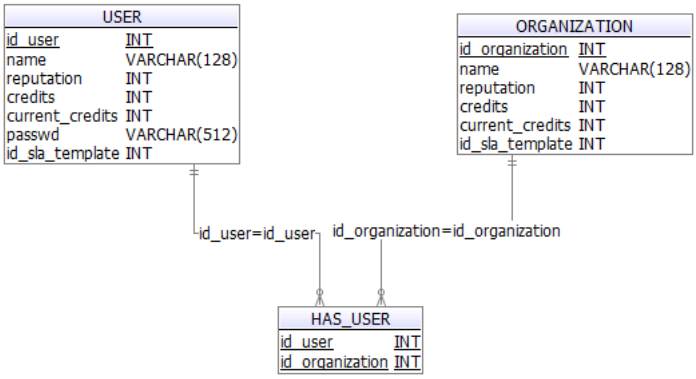

Figure 3.4: User model for Cloudcompaas.

This simple model is utilized to represent users in the system. The element Organization is introduced to offer the possibility of representing simple user associations, similar to the Grid concept of Virtual Organizations.

In this model a User must belong to at least one Organization. A User can belong to several Organizations.

3.2.5 SLA fragments, templates and instances

This section provides examples on how SLA templates, fragments and instances are implemented in Cloudcompaas.

The scenario models the use case of section 1.4. The web service chosen for the implementation is jLinpack [101], a Java implementation of the Linpack service. The jLinpack service has a dependency on the Java runtime. Moreover, due to its high memory requirements, the developers encourage using this service in machines with a high amount of memory.

Therefore the three basic Cloud resources involved in the scenario are a VM, a runtime Java and a jLinpack service. According to the methodology, these resources are represented using SLA fragments. An SLA fragment is an XML document that represents a Cloud resource following the WS-Agreement directives. Listing 3.1 shows the SLA fragment that represents the jLinpack service.

```
1  <Service Name="jLinpack">
2   <ServiceVersion Name="1.0"/>
3   <ServiceDescription>Java linpack implementation.</
        ServiceDescription>
4   <CreationConstraints>
5    <Item Name="vm">
6     <xs:enumeration value="large"/>
7    </Item>
8    <Item Name="java">
9     <xs:enumeration value="Java"/>
10   </Item>
11  </CreationConstraints>
12 </Service>
```

Listing 3.1: SLA fragment for the jLinpack service

The XML document represents the jLinpack service and the restrictions that it has with respect to other resources. These restrictions are represented with the WS-Agreement schema for CreationConstraints. These constraints represent the restrictions that the service must run in a VM of size large, and that the service must run in a VM that includes the runtime Java.

Listing 3.2 shows the SLA fragment for the Java runtime. The runtime is represented in a straightforward manner, since it includes no dependences or options. Listing 3.3 shows the SLA fragment for the VM large. The VM fragment defines the hardware configuration of a large VM in Cloudcompaas. The VM fragment includes the value of the four basic components of a VM in the Cloudcompaas domain, memory, cores, network and architecture.

```
1  <VirtualRuntime Name="Java"/>
```

Listing 3.2: SLA fragment for the Java runtime

```
<VirtualMachine Name="large">
 <PhysicalResource Name="Memory">
  1024
 </PhysicalResource>
 <PhysicalResource Name="Cores">
  2
 </PhysicalResource>
 <PhysicalResource Name="Network">
  2
 </PhysicalResource>
 <PhysicalResource Name="Architecture">
  x86_64
 </PhysicalResource>
</VirtualMachine>
```

Listing 3.3: SLA fragment for a large VM

These are the explicit Cloud resources involved in the scenario. However there is also a implicit set of elements needed to deploy the resources. First, the offer should include the OS that will run on the VM. Second, the Cloudcompaas domain model requires runtimes to be included in an enclosing Virtual Container. Lastly, the user that is deploying the resources must also be included in the SLA. Listing 3.4, 3.5 and 3.6 shows the SLA fragments of these elements.

```
<OperatingSystem>
 <OSId>103</OSId>
 <OSName>linux</OSName>
 <OSVersion>10.10</OSVersion>
 <OSFlavour>ubuntu</OSFlavour>
 <Hypervisor>kvm</Hypervisor>
 <Username>cloudcompaas</Username>
 <Disk>9GB</Disk>
</OperatingSystem>
```

Listing 3.4: SLA fragments for the OS

```
1 <VirtualContainer Name="BasicUbuntuServer"/>
```

Listing 3.5: SLA fragments for Virtual Container

```
1 <User Name="angarg12">
2  <UserCredits>9999</UserCredits>
3 </User>
```

Listing 3.6: SLA fragments for the user

The user also wants to include a QoS rule that dynamically scales the number of running service replica depending on the CPU load of the VM. These rules are already included in Cloudcompaas, and are defined as SLA fragments. Listing 3.7 shows the SLA fragment for an upscaling QoS rule.

```
<GuaranteeTerm Name="UPSCALE" Obligated="ServiceProvider">
 <ServiceScope ServiceName="SAMPLE"/>
 <QualifyingCondition>
  MAX_REPLICAS gt ACT_REPLICAS
 </QualifyingCondition>
 <ServiceLevelObjective>
  <KPITarget>
   <KPIName>CPUAVG</KPIName>
   <CustomServiceLevel>
    list.avg(CPUPERC) le 90
   </CustomServiceLevel>
  </KPITarget>
 </ServiceLevelObjective>
 <BusinessValueList>
  <Penalty>
   50
  </Penalty>
  <Reward>
   10
  </Reward>
 </BusinessValueList>
 <VariableSet>
  <Variable Name="MAX_REPLICAS"/>
  <Variable Name="ACT_REPLICAS"/>
  <Variable Name="CPUPERC"/>
 </VariableSet>
</GuaranteeTerm>
```

Listing 3.7: SLA fragment for the upscaling QoS rule (guarantee term in the WS-Agreement schema)

The QoS rule definition follows the WS-Agreement schema Guarantee Term. The Qualifying Condition represents a condition that must be met before evaluating the rule. On the Upscale rule, the Qualifying Condition is that the number of VM running currently is less than the maximum allowed. Following, the KPI Target defines the target that the QoS rule wants to meet. On the Upscale rule, the KPI Target is defined as $list.avg(CPUPERC)$ *le* 90. This code represents that the average CPU load of running VM should be less than 90%. Whenever this formula evaluates to true, the expected QoS level is being delivered. When this rule evaluates to false, the QoS is considered violated

(i.e. the average CPU load is greater than 90%). Lastly, the Business Value List defines the frequency of the evaluation of the rule as well as the price paid by providers (penalty) and users (reward) when the QoS rule is violated or fulfilled, respectively. On the Upscale rule, the evaluation interval is defined as 2 minutes ($PT2M$). The price paid by the provider to the user when the QoS rule is violated is 50 credits. The price paid by the user to the provider for the QoS rule whenever it is fulfilled is 10 credits.

These are the SLA fragments involved in the scenario. These SLA fragments are composed in response to a user request to build a complete SLA template. Listing 3.8 shows the SLA template resulting from the composition of these fragments.

```
<Template>
 <Name>SAMPLE1</Name>
 <Context>
  <ServiceProvider>AgreementResponder</ServiceProvider>
  <TemplateId>_L-2-3_H-5_I-5_A-1_D-4_C-0_E-0</TemplateId>
  <TemplateName>SAMPLE1</TemplateName>
 </Context>
 <Terms>
  <All>
   <ServiceDescriptionTerm>
    <Metadata>
     <Replicas>
      1
     </Replicas>
    </Metadata>
   </ServiceDescriptionTerm>
   <ServiceDescriptionTerm>
    <User Name="angarg12">
     <UserCredits>9999</UserCredits>
    </User>
   </ServiceDescriptionTerm>
   <ServiceProperties>
    <VariableSet>
     <Variable Name="MAX_REPLICAS"/>
     <Variable Name="ACT_REPLICAS"/>
     <Variable Name="CPUPERC"/>
    </VariableSet>
   </ServiceProperties>
```

```
<GuaranteeTerm Name="UPSCALE" Obligated="ServiceProvider">
 <QualifyingCondition>MAX_REPLICAS gt ACT_REPLICAS</
     QualifyingCondition>
 <ServiceLevelObjective>
  <KPITarget>
   <KPIName>CPUAVG</KPIName>
   <CustomServiceLevel>list.avg(CPUPERC) le 90</
  CustomServiceLevel>
  </KPITarget>
 </ServiceLevelObjective>
 <BusinessValueList>
  <Penalty>
   50
  </Penalty>
  <Reward>
   10
  </Reward>
 </BusinessValueList>
</GuaranteeTerm>
<ServiceDescriptionTerm>
 <Service Name="jLinpack">
  <ServiceVersion Name="1.0"/>
  <ServiceDescription>Java linpack implementation.</
      ServiceDescription>
 </Service>
</ServiceDescriptionTerm>
<ServiceDescriptionTerm>
 <VirtualMachine Name="large">
  <OperatingSystem>
   <OSId>103</OSId>
   <OSName>linux</OSName>
   <OSVersion>10.10</OSVersion>
   <OSFlavour>ubuntu</OSFlavour>
   <Hypervisor>kvm</Hypervisor>
   <Username>cloudcompaas</Username>
   <Disk>9GB</Disk>
  </OperatingSystem>
  <PhysicalResource Name="Memory">
   1024
  </PhysicalResource>
  <PhysicalResource Name="Cores">
   2
  </PhysicalResource>
```

```
69      <PhysicalResource Name="Network">
70       2
71      </PhysicalResource>
72      <PhysicalResource Name="Architecture">
73       x86_64
74      </PhysicalResource>
75     </VirtualMachine>
76    </ServiceDescriptionTerm>
77     <ServiceDescriptionTerm>
78      <VirtualContainer Name="BasicUbuntuServer">
79        <VirtualRuntime Name="Java"/>
80      </VirtualContainer>
81     </ServiceDescriptionTerm>
82   </All>
83  </Terms>
84  <CreationConstraints>
85   <Item Name="vm">
86    <xs:enumeration value="large"/>
87   </Item>
88   <Item Name="java">
89    <xs:enumeration value="Java"/>
90   </Item>
91  </CreationConstraints>
92 </Template>
```

Listing 3.8: SLA template for the scenario

The resulting SLA template combines all the SLA fragments in a single WS-Agreement document. The template has an id that uniquely identifies the template in the system. The template also has a name that can be customized by the user. These values are grouped in the context, which is a section specific of the template schema. The template also includes a new term metadata, which specifies the number of replicas to deploy. The default value is 1, and can be customized by the user. The remaining of the document consists on the SLA fragments of the Cloud resources.

After retrieving a SLA template, users can customize the different values and produce a SLA offer. Listing 3.9 shows an offer generated by a user based on the previous template.

```
1 <AgreementOffer>
```

```
<Name>jLinpack server</Name>
<Context>
 <ServiceProvider>AgreementResponder</ServiceProvider>
 <TemplateId>_L-2-3_H-5_I-5_A-1_D-4_C-0_E-0</TemplateId>
 <TemplateName>SAMPLE1</TemplateName>
</Context>
<Terms>
 <All>
  <ServiceDescriptionTerm>
   <Metadata>
    <Replicas>
     <LowerBound>2</LowerBound>
     <UpperBound>10</UpperBound>
    </Replicas>
   </Metadata>
  </ServiceDescriptionTerm>
  <ServiceDescriptionTerm>
   <User Name="angarg12">
    <UserCredits>9999</UserCredits>
   </User>
  </ServiceDescriptionTerm>
  <ServiceProperties>
   <VariableSet>
    <Variable Name="MAX_REPLICAS"/>
    <Variable Name="ACT_REPLICAS"/>
    <Variable Name="CPUPERC"/>
   </VariableSet>
  </ServiceProperties>
  <GuaranteeTerm Name="UPSCALE" Obligated="ServiceProvider">
   <QualifyingCondition>MAX_REPLICAS gt ACT_REPLICAS</
       QualifyingCondition>
   <ServiceLevelObjective>
    <KPITarget>
     <KPIName>CPUAVG</KPIName>
     <CustomServiceLevel>list.avg(CPUPERC) le 90</
    CustomServiceLevel>
    </KPITarget>
   </ServiceLevelObjective>
   <BusinessValueList>
    <Penalty>
     50
    </Penalty>
    <Reward>
```

```
43        10
44       </Reward>
45      </BusinessValueList>
46     </GuaranteeTerm>
47     <ServiceDescriptionTerm>
48      <Service Name="jLinpack">
49       <ServiceVersion Name="1.0"/>
50       <ServiceDescription>Java linpack implementation.</
             ServiceDescription>
51      </Service>
52     </ServiceDescriptionTerm>
53     <ServiceDescriptionTerm>
54      <VirtualMachine Name="large">
55       <OperatingSystem>
56        <OSId>103</OSId>
57        <OSName>linux</OSName>
58        <OSVersion>10.10</OSVersion>
59        <OSFlavour>ubuntu</OSFlavour>
60        <Hypervisor>kvm</Hypervisor>
61        <Username>cloudcompaas</Username>
62        <Disk>9GB</Disk>
63       </OperatingSystem>
64       <PhysicalResource Name="Memory">
65        1024
66       </PhysicalResource>
67       <PhysicalResource Name="Cores">
68        2
69       </PhysicalResource>
70       <PhysicalResource Name="Network">
71        2
72       </PhysicalResource>
73       <PhysicalResource Name="Architecture">
74        x86_64
75       </PhysicalResource>
76      </VirtualMachine>
77     </ServiceDescriptionTerm>
78     <ServiceDescriptionTerm>
79      <VirtualContainer Name="BasicUbuntuServer">
80       <VirtualRuntime Name="Java"/>
81      </VirtualContainer>
82     </ServiceDescriptionTerm>
83    </All>
84   </Terms>
```

```
85 </AgreementOffer>
```

Listing 3.9: SLA offer for the scenario

A SLA offer is very similar to a template. The only differences are that Creation Constraints are dropped, and that the user is able to modify the customizable values of the template. On this scenario, the user modified the name of the offer to jLinpack server, and changed the number of replicas from the default value fixed to 1 to a range of replicas between 2 and 10. Notice that if the user changes any of the non-customizable values, the offer would be rejected as invalid. Once a SLA has been accepted, Cloudcompaas creates a new SLA instance and returns its identifier to the user. Users can retrieve the state of their SLA instances from Cloudcompaas using this id. Listing 3.10 shows the state retrieved for the previous SLA.

```
1  <AgreementProperties>
2   <Name>jLinpack server</Name>
3   <AgreementId>56</AgreementId>
4   <Context>
5    <ServiceProvider>AgreementResponder</ServiceProvider>
6    <TemplateId>_L-2-3_H-5_I-5_A-1_D-4_C-0_E-0</TemplateId>
7    <TemplateName>SAMPLE1</TemplateName>
8    <InitializationTime>2013-04-13 12:36:59.351</InitializationTime>
9   </Context>
10  <Terms>
11   <All>
12    <ServiceDescriptionTerm>
13     <Metadata>
14      <Replicas>
15       <LowerBound>2</LowerBound>
16       <UpperBound>10</UpperBound>
17      </Replicas>
18     </Metadata>
19    </ServiceDescriptionTerm>
20    <ServiceDescriptionTerm>
21     <User Name="angarg12">
22      <UserCredits>9999</UserCredits>
23     </User>
24    </ServiceDescriptionTerm>
25    <ServiceProperties>
26     <VariableSet>
```

```
27      <Variable Name="MAX_REPLICAS"/>
28      <Variable Name="ACT_REPLICAS"/>
29      <Variable Name="CPUPERC"/>
30     </VariableSet>
31    </ServiceProperties>
32    <GuaranteeTerm Name="UPSCALE" Obligated="ServiceProvider">
33     <QualifyingCondition>MAX_REPLICAS gt ACT_REPLICAS</
           QualifyingCondition>
34     <ServiceLevelObjective>
35      <KPITarget>
36       <KPIName>CPUAVG</KPIName>
37       <CustomServiceLevel>list.avg(CPUPERC) le 90</
       CustomServiceLevel>
38      </KPITarget>
39     </ServiceLevelObjective>
40     <BusinessValueList>
41      <Penalty>
42       50
43      </Penalty>
44      <Reward>
45       10
46      </Reward>
47     </BusinessValueList>
48    </GuaranteeTerm>
49    <ServiceDescriptionTerm>
50     <Service Name="jLinpack">
51      <ServiceVersion Name="1.0"/>
52      <ServiceDescription>Java linpack implementation.</
           ServiceDescription>
53     </Service>
54    </ServiceDescriptionTerm>
55    <ServiceDescriptionTerm>
56     <VirtualMachine Name="large">
57      <OperatingSystem>
58       <OSId>103</OSId>
59       <OSName>linux</OSName>
60       <OSVersion>10.10</OSVersion>
61       <OSFlavour>ubuntu</OSFlavour>
62       <Hypervisor>kvm</Hypervisor>
63       <Username>cloudcompaas</Username>
64       <Disk>9GB</Disk>
65      </OperatingSystem>
66      <PhysicalResource Name="Memory">
```

```
67        1024
68       </PhysicalResource>
69       <PhysicalResource Name="Cores">
70        2
71       </PhysicalResource>
72       <PhysicalResource Name="Network">
73        2
74       </PhysicalResource>
75       <PhysicalResource Name="Architecture">
76        x86_64
77       </PhysicalResource>
78      </VirtualMachine>
79     </ServiceDescriptionTerm>
80     <ServiceDescriptionTerm>
81      <VirtualContainer Name="BasicUbuntuServer">
82       <VirtualRuntime Name="Java"/>
83      </VirtualContainer>
84     </ServiceDescriptionTerm>
85     <ServiceReference Name="References">
86      <EndpointReference>
87       158.42.104.74
88      </EndpointReference>
89      <EndpointReference>
90       158.42.104.75
91      </EndpointReference>
92     </ServiceReference>
93    </All>
94   </Terms>
95   <AgreementState>
96    <State>Observed</State>
97   </AgreementState>
98   <GuaranteeTermState termName="REPLICAS">
99    <State>Fulfilled</State>
100  </GuaranteeTermState>
101  <ServiceTermState>
102   <State>Ready</State>
103   <Metadata>
104    <Replicas>
105     2
106    </Replicas>
107   </Metadata>
108  </ServiceTermState>
109  <ServiceTermState>
```

```
110   <State>Ready</State>
111   <User Name="angarg12">
112    <UserCredits>7799</UserCredits>
113   </User>
114  </ServiceTermState>
115  <ServiceTermState>
116   <State>Ready</State>
117   <Replica Id="158.42.104.74">
118    <LastUpdate>2013-04-13 12:41:42.311</LastUpdate>
119    <CPU_USED_PERC>0.3193057147535615</CPU_USED_PERC>
120    <MEM_USED_PERC>48.582985</MEM_USED_PERC>
121    <IN_NET>735.0</IN_NET>
122    <OUT_NET>0.0</OUT_NET>
123   </Replica>
124   <Replica Id="158.42.104.75">
125    <LastUpdate>2013-04-13 12:41:42.311</LastUpdate>
126    <CPU_USED_PERC>0.536918326854220</CPU_USED_PERC>
127    <MEM_USED_PERC>66.762424</MEM_USED_PERC>
128    <IN_NET>821.0</IN_NET>
129    <OUT_NET>0.0</OUT_NET>
130   </Replica>
131  </ServiceTermState>
132 </AgreementProperties>
```

Listing 3.10: SLA state for the scenario

A SLA state document includes the definition of Cloud resources from the SLA. It also adds information regarding the SLA instance statistics and monitoring information. The SLA state includes the id number of the SLA instance that it belongs to. The context is update to include the timestamp of when the SLA instance was created. At the end of the service terms, the service reference schema is included. This WS-Agreement schema is used to represent the endpoint references of the deployed Cloud resources. On the scenario, this schema includes the endpoint references of the deployed VM, namely their IP addresses. At the end of the document, the SLA state, guarantee term state and service term schemas include the state of the Cloud resources and their monitoring information. The SLA state describes the state in which the SLA instance is at that particular moment. On the scenario, the state is Observed, which means that the SLA is currently active. The SLA state describe whether a QoS rule is violated or fulfilled. The

service term state describes whether a Cloud resource is ready or not ready (e.g. if it is currently being deployed), and contains its monitoring information, if applicable. On this scenario, the VM resources include monitoring information regarding the CPU, memory and network usage, while the Virtual Container and the Service terms only include the state. Two particular cases are the User and the Metadata. The user term state includes the current number of credits of the user. If the number of credits reaches 0, the SLA is decommissioned, since the user cannot pay for it anymore. The metadata term includes the current number of replicas of the Cloud resources.

3.3 Architecture

Cloudcompaas is a distributed framework built of interrelated components that perform specific tasks, based on the architecture presented in [100], depicted in Figure 3.5. This section describes the components of this architecture and their role in the allocation and management of Cloud services.

3.3.1 SLA Manager

The SLA Manager is the entry point to Cloudcompaas. The SLA-driven nature of the framework implies that every interaction among components is performed by means of SLAs. Therefore any external interaction must pass through this component. The SLA Manager can build documents, check offers for correctness and register new SLAs. The four basic operations supported are search, create, query and delete.

The search operation enable users to retrieve SLA templates according to different criteria. The create operation sends an SLA offer to the component. It checks that the offer complies with the SLA template. If this operation fails, the offer is rejected. If the offer is well defined, the SLA Manager sends it to the Orchestrator to schedule its deployment. If this operation fails, for instance because no free resources are available, the offer is rejected. After an SLA has been accepted and its

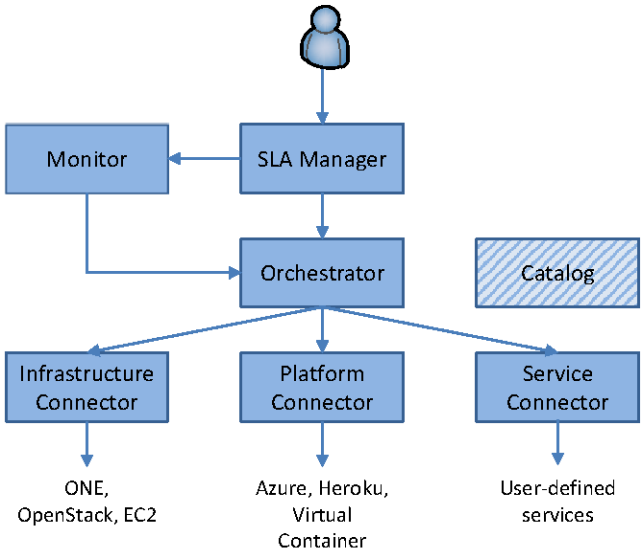

Figure 3.5: Cloudcompaas architecture.

resources have been allocated, the SLA Manager registers the SLA in the Monitor component. The query operation enable users to retrieve the state of SLAs that they have sent to the framework (including the rejected ones). The delete operation allows users to deallocate the resources associated with an SLA and stop its monitoring. The SLA Manager checks if the appointed SLA is currently active and if the user has rights to delete it before interacting with the Orchestrator and the Monitor to delete the SLA.

3.3.2 Monitor

The Monitor component performs the assessment of the dynamic features of SLAs. For each SLA, this component schedules the retrieval of the monitoring data of the resources and QoS rules. This data is used to evaluate the rules and check for SLA violations.

If a QoS rule violation is found, the Monitor apply corrective actions to restore the state of the system and eliminate the violation. These actions usually are specific to the domain of application.

The Monitor also performs accounting and billing operations. It charges the user for the resources consumed periodically, and applies discounts, penalties to each party when the QoS violations occur.

The Monitor depends on the SLA Manager to register and unregister SLAs when they start and finish they lifecycle, respectively.

3.3.3 Orchestrator

The Orchestrator is the central component of the framework and acts as a global coordinator. When a new SLA is accepted by the SLA Manager, a deployment request is sent to the Orchestrator. This component keeps a global view of all the available Cloud backends, and performs the scheduling of the Cloud service based on the SLA requirements and the available resources. The Orchestrator manages the deployment process by delegating the allocation operations to a set of IaaS, PaaS and SaaS connectors. Hence, the scheduling procedure selects the Cloud backend that will deploy the resources. The Orchestrator performs a sequential process for the allocation of resources. It communicates with the Infrastructure Connector, Platform Connector and Service Connector to deploy IaaS, PaaS and SaaS resources on this order, and feeds each level with information retrieved from the previous one. Using this procedure, the Orchestrator can deploy resources from one level on top of resources from the lower levels of the hierarchy.

3.3.4 Infrastructure connector

The Infrastructure Connector is the component in charge of deploying the infrastructure resources in a specific IaaS Cloud backend. Hence the main task of this component is to allocate required infrastructure and configure the resources according to the SLA specification. Since the Infrastructure Connector component can be implemented for

several different Cloud providers, a plug-in approach has been used. The basic Infrastructure Connector component provides a uniform interface to Cloudcompaas, including the SLA deployment, SLA undeployment and SLA adjustment operations. Several plug-ins provide a specific implementation for each target Cloud provider.

When a plug-in receives an incoming request, it must check if the Cloud deployment can serve that request. A provider may be unable to serve a request either if there are no free resources available or if the provider does not support at least one of the resources specified in the request. Once an SLA has been approved, the plug-in translates the SLA representation of resources to the back-end specific representation of resources and requests the resource allocation. When the resources have been successfully allocated, the plug-in retrieves its identifier, reference endpoint and other relevant information.

Finally, the plug-in performs the configuration of the deployed resources. The configuration step includes the automatic operations needed for the correct behavior of the system such as the injection and execution of a monitoring agent or the setup of a guest user account on the Virtual Machines. Once all operations have finished, the Infrastructure Connector returns the retrieved information (such as the endpoint references) to the Orchestrator.

3.3.5 Platform connector

The Platform Connector receives the SLA representation of a Virtual Container and translates it to the specific representation of the PaaS backend. Then it deploys and configures the Virtual Container. Once the resources are allocated, the endpoint reference of the Virtual Containers and other relevant information is retrieved. As a particular case, the SLA can specify that the Virtual Containers must be hosted in the Virtual Machines deployed previously. In this case, the plug-in must retrieve the endpoint references of the Virtual Machines and contextualize them with the Virtual Container software stack if needed.

3.3.6 Service connector

The Service Connector receives the SLA representation of a Virtual Service and translates it to the specific representation of the SaaS backend. Then it deploys and configures the Virtual Service. Once the resources are allocated, the endpoint reference of the Virtual Services and other relevant information is retrieved. As a particular case, the SLA can specify that the Virtual Service must be hosted in the Virtual Containers deployed previously. In this case, the plug-in must retrieve the endpoint references of the Virtual Container and deploy the Virtual Service on it.

3.3.7 Catalog

The Catalog implements an Information System, by means of a distributed and replicated database accessible through a RESTful API. The other components use this Information System for retrieving and storing a variety of information, such as SLAs, SLA templates, and runtime and monitoring information.

Chapter 4

Implementation

This chapter provides insight at the particularities of the implementation of the software developed in the Thesis. It provides an analysis of the SLA Composition algorithm, a key component for the implementation of the SLA composition methodology. This section also introduces the implementation of Cloudcompaas and the components involved in the SLA-driven resource lifecycle management, including the scheduling and management of Cloud services.

4.1 SLA Composition

This section provides details on the implementation of the SLA composition methodology. It begins providing a formal statement of the SLA composition problem. Next, it introduces the SLA composition algorithm, including a basic implementation of the algorithm. Finally it describes a set of optimizations to improve its performance.

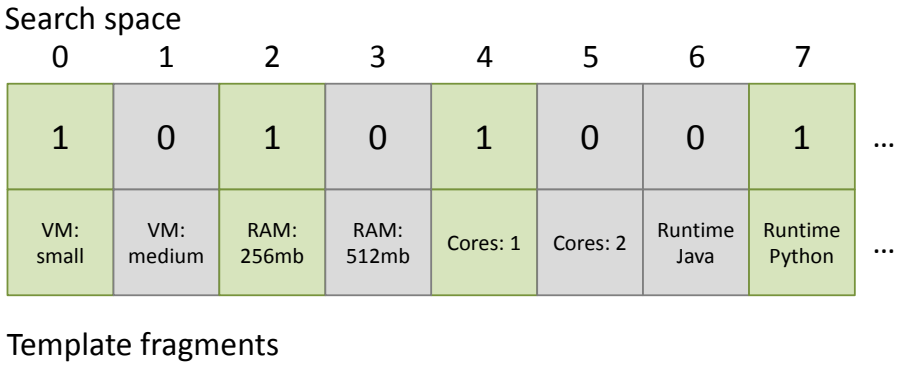

Figure 4.1: The solution space is represented by an array of booleans. Each index value indicates whether the indexed template is added or not to the solution.

4.1.1 The SLA Composition problem

In regular SLA-driven platforms, SLA templates are stored explicitly, and users can retrieve a list of the available templates or perform queries. In our proposed scenario SLA templates are generated on the fly in response to user requests.

The SLA Composition problem can be characterized as a decision problem over the space of template fragments. A problem instance is a set of restrictions that the target template must met, and is represented as a Boolean expression composed by variables and AND, OR, NOT operations. A solution to the problem is a combination of template fragments that fulfill the proposed restrictions, and is represented as an array of Boolean values that indicates the template fragments which are combined together, as depicted in Figure 4.1. A solution is valid if and only if the provided values satisfy the restrictions of the problem.

This characterization is a reduction of the SLA Composition problem to the Boolean satisfiability (SAT) problem. SAT is a well-known instance of NP-complete problem [102]. Many decision and optimization problems can be reduced to SAT, and this reduction is used as a

proof of the NP-completeness of a particular problem instance [103]. Based on this reduction, the NP-completeness of the SLA Composition problem is proven. The general complexity of NP-complete problems is $O(2^n)$, where n is the number of SLA templates in the system.

However this linear characterization does not represent the nature of the problem. The hierarchical organization of the elements in Cloudcompaas is recursive for some elements. For instance, a SLA must decide the set of Cloud resources that complies with a set of restrictions. When selecting the VM resource, the problem consists on determining which VM hardware configuration complies with the set of restrictions. Therefore the problem is applied recursively for each Cloud resource, where a decision problem is solved at each node of a tree hierarchy (e.g. a hierarchy of VMs, Runtimes, Software...) with a cost of $O(2^n)$.

4.1.2 The SLA Composition algorithm

Following the general overview of the SLA Composition problem, a brute force SLA composition algorithm, depicted in Algorithm 1, has been designed. This algorithm sets the grounds for the proposed solutions to this problem. The brute force algorithm for SLA Composition consists of two major steps, the branching step and the sub-problem

resolution step. The algorithm explores the solutions in the search space filtering out those that do not meet the restrictions of the model.

```
Data: target, searchspace, solutionspace
Result: solutionspace
1  if searchspace.end then
2  |  if meetRestrictions(target) and isValid(target) then
3  |  |  solutionspace.add(target);
4  |  end
5  |  return;
6  end
7  if isTerminal(searchspace.current) then
8  |  // do not add the item
9  |  compose(target, searchspace.next, solutionspace);
10 |  // add item
11 |  newtarget = merge(target, searchspace.current);
12 |  compose(newtarget, searchspace.next, solutionspace);
13 else
14 |  // spawn new combinatorial problem
15 |  subsearchspace = generateSearchSpace(searchspace.current);
16 |  compose(searchspace.current, subsearchspace, subsolutionspace);
17 |  merge(searchspace, subsolutionspace);
18 |  compose(target, searchspace.next, solutionspace);
19 end
```

Algorithm 1: Brute force implementation

The algorithm receives as input a target item, a search space and a solution space. The search space is the set of template fragments considered for the current problem. The solution space is the set of combined template fragments that are valid solutions to the problem. The target is the template being currently analyzed for a solution. In the characterization of the problem, the solutions are represented as arrays of Booleans which define the templates used in the composition. However, for the actual SLA Composition algorithm, this representation is cumbersome, and also do not properly represents the data involved in the process. A structure called Composition Item is used as replacement of the solution array depicted in the previous

section. Instead of representing a candidate solution as an array of Booleans where each index indicates whether the corresponding fragment is added or not to the target, candidate solutions are represented by the corresponding template. This structure stores four elements of data, a template fragment, an id, a length value and a Boolean that determines whether the item is a terminal element or not.

Terminal elements represent elements that do not produce new fragments. For instance, a Physical Resource Memory: 256MB represents a single element that cannot be combined any further with other elements. Non-terminal elements represent elements that can be combined with others by solving a combinatorial instance, and produce a new set of template fragments. For instance, a VM element may be combined with several Physical Resources, producing a set of possible hardware configurations for that VM.

The composition algorithm is naturally recursive. The base case occurs when the search space is empty. In the base case, the target template is evaluated, and if it is correct according to the constraints, then it is added to the solution space. The recursion step occurs whenever the search space is not empty. In the recursion step, an element from the search space is selected and processed. The behavior of the algorithm depends on whether the processed element is terminal or non-terminal.

Figure 4.2 illustrates a step-by-step example of an execution of the SLA composition brute force algorithm. Non-terminal elements (NT) represent the internal nodes of the search tree (e.g. VM). Terminal elements (t) represent the leaf nodes of the search tree (e.g. physical resources such as memory, cores, etc.). A NT element is composed by an aggregation of t elements. Each arrow in the tree represents a combinatorial problem for a NT element over a search space of t elements.

The algorithm begins with an empty SLA template, and a search space populated by the non-terminal elements NT0 and NT1 (1). The procedure starts considering the first element in the search space (NT0) to be added to the empty template. When the algorithm tries to

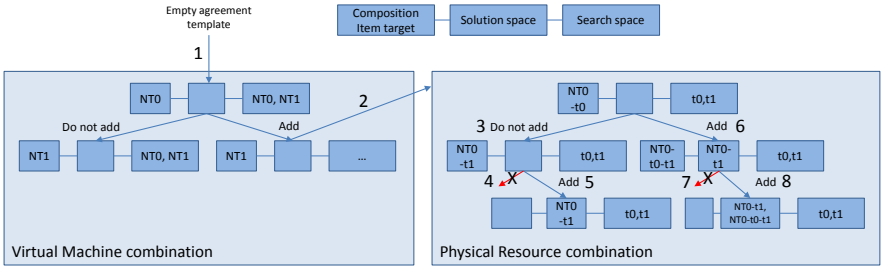

Figure 4.2: Detail on how a VM combinatorial problem spawns a physical resource combinatorial problem.

add NT0 to the template, it notices that it is not a terminal element, and then spawns a new combinatorial problem (2). The newly spawn combinatorial problems aim is to generate all the terminal elements derived from a non-terminal one, NT0 on this case. The combinatorial problem will explore a search space populated by two terminal elements, t0 and t1, trying to generate all the valid combinations of NT0. First the new combinatorial problem explores the branch where t0 is not added to NT0 (3). Then, it explores the branch where t1 is also not added to NT0 (4). However, this operation fails. A combination operation fails because it violates some restriction of the domain data model. For instance, one restriction that could cause the failure of step 4 is that an NT element must include at least one t element. The algorithm proceeds by the alternative branch in the composition procedure, and adding t1 to NT0 (5). This operation succeeds and the resulting template (NT0-t1) is added to the solution space.

The algorithm continues the depth-first exploration of the tree by considering adding t0 to NT0 (6). This operation succeeds and then the next steps in the algorithm considers whether to add the next element of the search space (t1) to the partial solution found so far (NT0-t0). The first branch explores the possibility of not adding t1 and therefore produces NT0-t0 as a final solution (7). However this step fails and that template is not added to the solution space. Lastly, the algorithm adds t1 to the partial result (8) and produces NT0-t0-t1. This operation succeeds, that is, this configuration of elements is

correct according to the data model, and hence this template is added to the solution space.

After the combinatorial problem ends, the NT0 template from the parent problem has been substituted by NT0-t1 and NT0-t0-t1. NT0, which was a non-terminal element, has been substituted by all the possible valid combinations of NT0 configurations. This allows the algorithm to treat these templates as terminal elements and continue exploring the search space to produce new solutions.

4.1.3 Optimizations of the algorithm

The brute force SLA Composition algorithm has a complexity of $O(2^n)$ for n templates. Problems of such complexity are considered as not tractable by brute force techniques. Programming techniques have been developed to tackle some of these problems. This section explains three major optimization techniques applied to the brute force algorithm. These techniques rely on two principles to reduce the complexity of the problem, using heuristics and focusing on special cases. Heuristics are used to produce good solutions in a fraction of time of the original algorithm, even tought it doesn't guarantee finding an optimal or correct solution. Focusing on special cases may allow solving some instances of an NP-Complete problem in polynomial time.

Branch and bound

Branch and bound is a general algorithm for finding the optimal solution to a variety of problems, especially combinatorial optimization problems. This algorithm consists on the systematic enumeration of all candidate solutions, where large subsets of fruitless candidates are discarded by using an estimator of the candidate solutions quality. The objective of the algorithm formulated as a combinatorial problem is finding feasible solutions in the solution space. The algorithm explores a search tree with all the possible template fragments combinations, where invalid combinations are discarded from the tree. In the SLA composition problem the number of fragments included in a

template, defined as its 'length', can be used as an estimator of the quality of the solution. The reason is that solutions that involve less template fragments provide the required functionality using fewer resources. This estimator can be used to prune branches for 'longer' templates. The technique of pruning by size reduces the execution time by exploring a subset of the solution space.

The branch and bound algorithm stops the search as soon as the 'shortest' possible answers are found, applying backtracking to explore other alternatives if no valid solution is found. Backtracking consists in resuming the exploration of a combinatorial problem already computed, in order to retrieve the next set of solutions. It occurs when the selected shortest candidate items do not provide valid solutions. The computational cost of the worst case scenario of the branch and bound algorithm is the same as the base algorithm, since it may be possible for the branch and bound method to explore the complete search space before finding a solution. However the experimental average cost of the algorithm is much lower than the worst case scenario.

Dynamic programming

The branch and bound algorithm works well for the resolution of a single combinatorial problem. However, as described before, the SLA Composition problem is recursive at some elements, solving several combinatorial problems on the process. At this level is where the Dynamic Programming paradigm is applied.

Dynamic Programming is a method for solving complex problems. The problems candidate to be solved by this paradigm must include two features, optimal substructure and overlapping subproblems. Optimal substructure refers to the possibility of obtaining an optimal solution for a problem based on the optimal solution of its subproblems. Overlapping subproblems refers to the redundant nature of the recursive structure of the problem, where the same problem instances are solved several times.

The SLA Composition problem exhibits both features.

- optimal substructure: In order to find a valid template composed by the fewer number of fragments, the fragments must be valid and composed by the fewest number possible themselves;

- overlapping subproblems: For each pair of element to be composed in the SLA structure, the product set of both elements instances is the set of subproblems to solve. If any of those elements itself can be composed with another set of items, a recursive instance of the problem is produced.

Dynamic Programming methods can be used to speed up the execution of the SLA Composition problem. Algorithm 2 shows a script that implements the behavior of the algorithm. Every time a non-terminal node is found in the search space, a global cache is checked for the solution of the combinatorial problem spawn from that node. In case the subproblem has been solved previously, the stored solution is used. Otherwise, a new subproblem is spawn and solved, and the solution is stored in the global cache for future reference. This procedure is called memoization.

```
Data: target
Result: solutionspace
1 solutionspace = getCache(target);
2 if solutionspace == null then
3 |   searchspace = generateSolutionSpace(target);
4 |   for item ∈ searchspace do
5 |   |   compose(item, searchspace, solutionspace);
6 |   end
7 |   putCache(target, solutionspace);
8 end
9 return solutionspace;
```

Algorithm 2: Memoized version of the recursion step in the SLA composition algorithm

Ad-hoc optimizations

Even though generic programming techniques such as branch and bound and programming provide a substantial improvement in the performance of the algorithm, the SLA Composition problem exhibits certain features that may be exploited to increase the efficiency in the calculations.

The first optimization is implicit pruning by semantic restrictions. Even though the brute force approach generates all the combinations of templates and checks their validity just before adding them to the solution space, some templates can be identified as invalid much earlier and removed from the search space, reducing the space to explore. One example of such restrictions is that no template may have more than one VM. Using this restriction is possible to prune the search space of fruitless candidates improving the performance of the algorithm.

The second optimization exploits the structure of the data to reduce the number of operations to perform. The brute force approach produces the Cartesian product of two sets of items every time a new combinatorial subproblem is spawn, producing, for instance, every combination of Operating System and Software components. However, in a real scenario not every Software is available to every Operating System. By capturing this information in the data model of the algorithm, it is possible to reduce the search space of combinatorial subproblems by limiting the number of combinations explored.

The third optimization combines both preceding techniques. It consists in reducing the search space based on the semantic restrictions of the model and the organization of the data. For instance, the semantic restriction that one Virtual Service can include only a single Service Version can be exploited to reduce the combinatorial problem on Services. Instead of producing every possible combination of Service Versions, that has a cost of $O(2^n)$, the algorithm explores each Service Version only once, with a cost of $O(n)$.

The last optimization consists on pruning the search space by means of user restrictions. In the brute force algorithm, the user restrictions

are used only to filter out the solutions at the end of the calculation. However it is possible to embed these restrictions into the search process. For instance, if the user specifies that a particular VM must be included in the solution, the algorithm directly includes that template to the base template, and start off the search procedure from a non-empty template. As users provide more restrictions, the algorithm explores a smaller subset of the search space.

4.2 Cloudcompaas Framework

The Cloudcompaas framework implements the architecture present in section 3.3, including a module for each component in the architecture. Cloudcompaaas implements a SLA-driven framework for the complete lifecycle management of Cloud services. The framework manages both the static (e.g. resources deployment and scheduling) and dynamic (e.g. QoS management) features of Cloud services. The following sections describe concrete implementation details concerning relevant aspects of the framework.

4.2.1 Interactions and flow of control

An interaction with Cloudcompaas begins when a user sends a request to the SLA Manager. The operations available to users are searching for SLAs, creating a SLA, retrieving the state of a SLA or terminating a SLA. An interaction diagram for each one of these functions is show following.

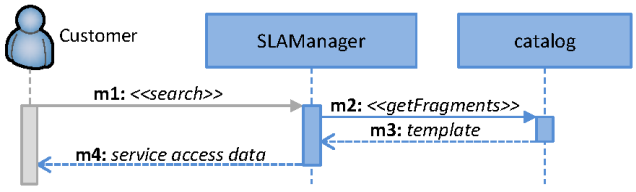

Figure 4.3: Interaction diagram for the search operation

Figure 4.3 shows the interaction diagram for the search operation. The interaction begins with a user sending a search request to the SLA Manager component. The SLA Manager retrieves a set of SLA fragments from the catalog according to the user requests, performs the SLA composition algorithm and returns the resulting template back to the user.

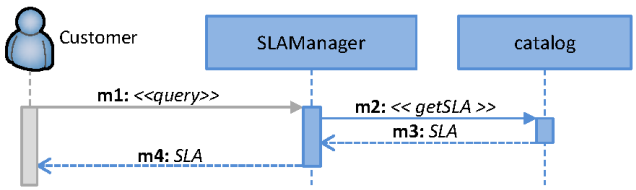

Figure 4.4: Interaction diagram for the retrieve operation

Figure 4.4 shows the interaction diagram for the retrieve operation. The interaction begins with a user sending a retrieve request to the SLA Manager component. The SLA Manager retrieves the indicated SLA from the catalog and returns it back to the user.

Figure 4.5 shows the interaction diagram for the create operation. The interaction begins with a user sending a create request to the SLA Manager component. The SLA Manager verifies that the SLA offer is correct, and sends a register request to the Monitor component. This request is asynchronous since the monitoring process continues as long as the SLA is active. Following, the SLA Manager sends a deploy request to the Orchestrator. The Orchestrator coordinates the deployment of the Cloud resources defined in the SLA communicating with the corresponding connectors. The order in which the communication occurs is relevant, since resources at different levels may depend on one another. The Orchestrators begins sending a deploy operation to the Infrastructure Connector. The Infrastructure Connector communicates with a specific Cloud back-end (such as OpenNebula, OpenStack or EC2) to deploys the VM resources specified in the SLA, and returns back to the Orchestrator the state of these resources (i.e. their IP addresses). The Orchestrator continues sending a deploy operation to the Platform Connector, that deploy the resources (Vir-

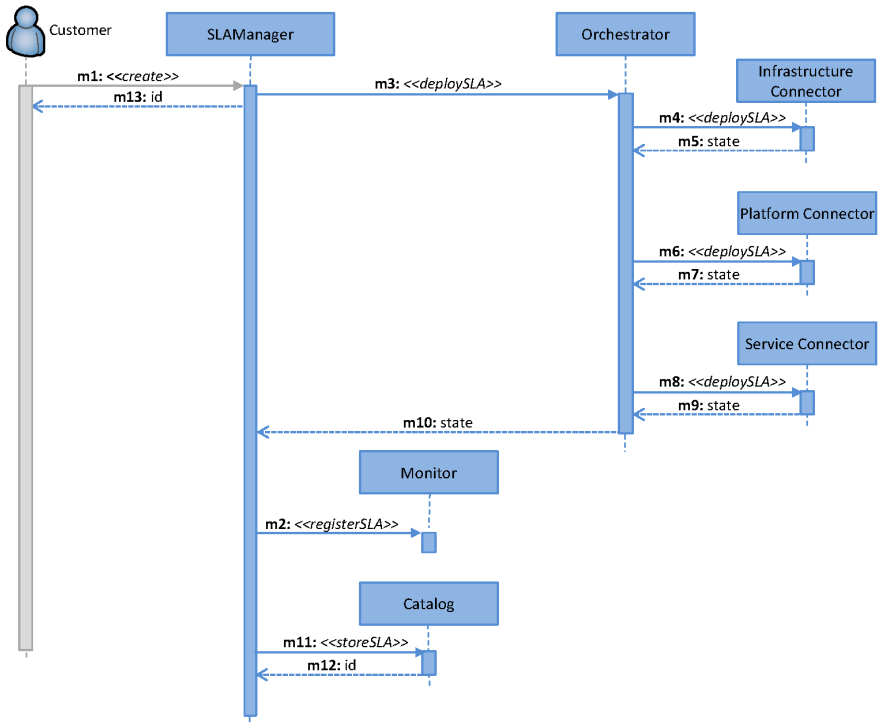

Figure 4.5: Interaction diagram for the create operation

tual Containers, software components) in a Cloud back-end (such as Azure, Heroku or VM deployed by the Infrastructure Connector). The Platform Connector sends back to the Orchestrator the state of the platform when done. Finally, the Orchestrator sends a deploy operation to the Service Connector. The Service Connector deploys a user defined services in a PaaS Cloud (such as AppEngine or a Virtual Container deployed by the Platform Connector) or allocates a service from a Cloud back-end. The Service Connector sends back to the Orchestrator the state of the services. The Orchestrator combines the state of the Cloud service and sends it back to the SLA Manager. Finally the SLA Manager updates the SLA with the state and stores the SLA in the Catalog. The Catalog sends back to the SLA Manager the identifier of the newly created SLA, and the SLA Manager sends back the id to the user.

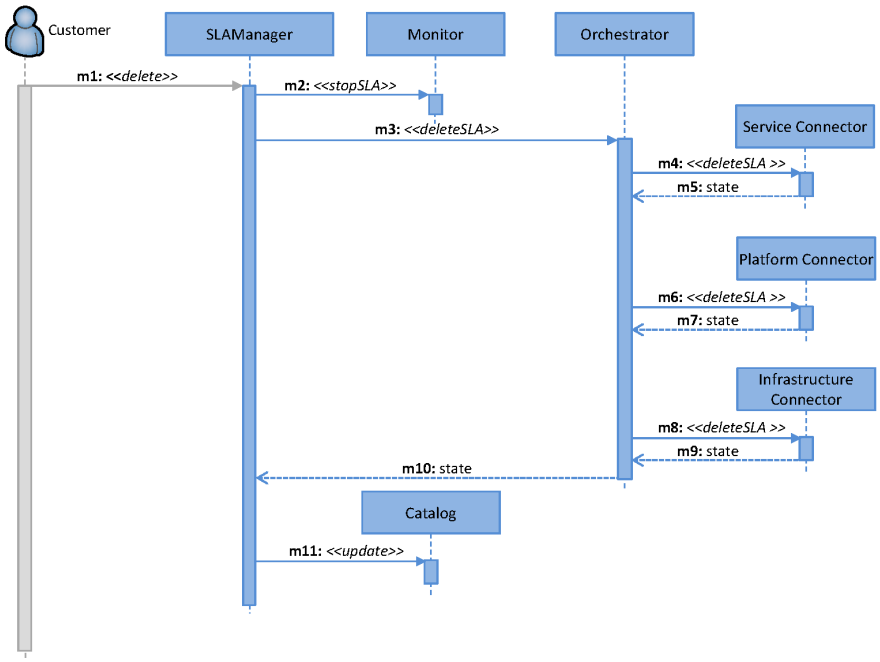

Figure 4.6: Interaction diagram for the terminate operation

Figure 4.6 shows the interaction diagram for the terminate operation. This is the only asynchronous operation, since it returns no value. The interaction begins with a user sending a terminate request to the SLA Manager component. The SLA Manager verifies that the user has rights to terminate the SLA, and sends a stop request to the Monitor component. The Monitor stops the monitoring cycle for the SLA and deletes it. Following, the SLA Manager sends a delete request to the Orchestrator. The Orchestrator communicates with the Connectors in order to undeploy and delete the resources. The Orchestrator sends a delete request to the Service Connector, Platform Connector and Infrastructure Connector on this order. Each connector is in charge of deleting the resources associated with the SLA at that level. Finally, the SLA Manager modifies the value of the SLA and sends an update request to the Catalog.

Interfaces and communications

The interfaces and communication system are designed according to the REST principle [104]. REST (Representational State Transfer) is a design principle where the interactions are performed by means of the manipulation of representation of resources. These manipulations are done using four basic HTTP operations GET, PUT, POST and DELETE. A service that adheres to the REST principles is RESTful. The interfaces of the components of the framework follow RESTful paradigm. As the WS-Agreement specification provides a reference SOAP interface for the implementation of the standard, a reference REST interface for WS-Agreement is defined in [105]. Using these guidelines, the interfaces of the components are designed to maintain a uniform semantic for the operations in the framework.

Scheduling

The role of the Orchestrator is to schedule and coordinate the deployment of the SLAs. A relevant decision in the design of the framework is how the scheduling process will be done. The major actors in the deployment process are the Infrastructure Connectors, each one interacting with a specific Cloud back-end, and the Orchestrator, coordinating the actions of different Infrastructure Connectors. The scheduling process can be performed by the Orchestrator component, by the Infrastructure Connector or by both at different extents.

Scheduling at Infrastructure Connector level is simpler. The Infrastructure Connector scheduling operation decides whether a back-end can deploy or not an SLA. On the other hand, the component only has a limited view of the complete system, and cannot take global scheduling decisions that optimize resource utilization.

Scheduling at Orchestrator level has the greatest flexibility regarding resource usage. The Orchestrator has a global view of the system, the available resources, Infrastructure Connectors state, etc. Using all the available information a wide variety of scheduling schemas and algorithms can be implemented, following different criteria as needed

(maximize user satisfaction, minimize providers cost, etc.). On the other hand, managing that amount of information and coordinating disparate components greatly increases the complexity of the operations of the module. A mixed approach consists on limiting the vision the Orchestrator has of the whole system and support the scheduling process at both component levels. Using this schema, the Orchestrator makes global scheduling decisions, and delegates the rest of the process to the selected Infrastructure Connectors. The Infrastructure Connector then performs local scheduling decisions.

Metadata representation

The domain model for Cloudcompaas includes four elements, namely Infrastructure, Platform, Service and User, and their relationships. When this model is translated to a domain-specific language that is expressed in a WS-Agreement document, these elements and their restrictions and requirements are represented as Service Description Terms, with each Term including all the information regarding each element. However, some information is not captured in this model, specifically implementation information commonly described as Metadata. The Metadata term, unlike the other four terms, does not describe resources in the system, but captures information needed for the management of the SLA. Introducing this fifth term to the Cloudcompaas SLA schema, the framework captures all the information needed for the proper management of resources.

4.2.2 Components Implementation Details

This section provides insight at the details of the implementation of the framework modules.

SLA Manager

The SLA Manager provides a RESTful implementation of the WS-Agreement operations. The SLA Manager extends the WSAG4J framework to provide additional dynamic resource management capabilities. The full extent of the modifications performed to the framework are presented in Section 4.3.

Orchestrator

The Orchestrator coordinates the components and the deployment and management of SLAs. When a SLA deployment operation is received, the Orchestrator forwards the request to an appropriate Infrastructure Connector. An Infrastructure Connector is appropriate if it can support a given SLA. The Orchestrator determines appropriate Infrastructure Connectors by matching the SLA requirements with the Connector capabilities.

The Orchestrator retrieves the information returned by the Infrastructure Connector (usually the endpoint reference of the newly deployed VMs). If the SLA defines a Virtual Container, the Orchestrator forwards the request to a Platform Connector and includes the information retrieved from the previous component. This communication flow allows the Platform Connector to communicate with the newly deployed VMs and proceed with the installation and configuration of the required software.

The Orchestrator retrieves the information returned by the Platform Connector, and if the SLA defines a Virtual Service, repeats the operation forwarding the request to a Service Connector, passing the information retrieved from the previous components.

SLA undeployment operations are processed in a similar fashion but in reverse order. The Orchestrator begins by forwarding the request to the Service Connector, Platform Connector and Infrastructure Connector on this order. The rationale is to allow services to be deallocated in order before shutting down the host VM, allowing users to

finish ongoing interactions and performing a clean shutdown. The Service Connector sends a shutdown operation to the deployed services, waiting for them to cleanly allow users to finish the current interaction before stopping the service. The Platform Connector sends a shutdown operation to the Virtual Container, that waits for current operations to finish and stops the Container. Finally, the Platform Connector sends a shutdown operation to the VMs and performs the deallocation of the resources.

Infrastructure Connector

The Infrastructure Connector is the component in charge of connecting Cloudcompaas to different IaaS Cloud backends. The Infrastructure Connector component provides a common interface, communications and security systems. A plug-in system implements the operations needed to deploy SLAs in each different Cloud provider.

Infrastructure Connector also performs local scheduling labors. In the current implementation these operations are straight forward. The plug-in first determines the minimum set of resources offered by the provider that comply with the SLA document, and checks if there is enough capacity to deploy the resources. If this operation succeeds, the plug-in allocates and contextualizes the services; otherwise the operation fails.

The Infrastructure Connector main operation consists on translating the WS-Agreement representation of resources of Cloudcompaas to the domain specific representation of resources of the underlying Cloud platform. This operation must be manually implemented for each plug-in.

For instance, the OpenNebula plug-in translates the Cloudcompaas representation of VM of Listing 4.1 to the ONE representation of VM of Listing 4.2. The plug-in may use a common parser to extract the key information from the SLA fragment of a VM (such as CPU cores or memory). However, it needs to implement a function that builds the ONE representation of VM from the parsed information. This func-

tion must account for the limitations and restrictions of each Cloud platform, for instance if a specific resource is not supported.

```
<VirtualMachine Name="large">
 <OperatingSystem>
  <OSId>103</OSId>
  <OSName>linux</OSName>
  <OSVersion>10.10</OSVersion>
  <OSFlavour>ubuntu</OSFlavour>
  <Hypervisor>kvm</Hypervisor>
  <Username>cloudcompaas</Username>
  <Disk>9GB</Disk>
 </OperatingSystem>
 <PhysicalResource Name="Memory">
  1024
 <PhysicalResource Name="Cores">
  2
 </PhysicalResource>
 <PhysicalResource Name="Network">
  2
 </PhysicalResource>
 <PhysicalResource Name="Architecture">
  x86_64
 </PhysicalResource>
</VirtualMachine>
```

Listing 4.1: Cloudcompaas representation of a VM

```
CPU = 2
MEMORY = 1024
OS = [ ARCH = "x86_64"]
DISK = [
IMAGE_ID = 103
]
NIC=[NETWORK_ID=2]
```

Listing 4.2: OpenNebula representation of a VM

Infrastructure Connector plug-ins also manage the implementation details of each specific Cloud platform and hide them from Cloudcom-

paas. For instance, OpenNebula assigns to each VM an id. This id is independent from the Cloudcompaas id for the SLA. Therefore the Infrastructure Connector must keep track of which OpenNebula VM ids correspond to the VM instances of each SLA transparently to Cloudcompaas.

Catalog

The Catalog module implements a distributed database that stores information such as SLAs, instances, monitoring information, etc. The Catalog is used by the other modules to store and retrieve this information. The current implementation of the Catalog module includes an embedded HSQLDB [106] database that stores the data model of the framework. A RESTful interface enables authorized users to manipulate the information.

The RESTful interface enables components to perform the four basic operations (GET, POST, PUT and DELETE) on a piece of data. Each piece of data is referenced by a URL. In order to operate on different data, modules must manipulate them one at a time.

Resources point to a URL, so if you want for instance to store information on different tables, you need to create different resources. URL is the endpoint representation of the table-item that you want to manage.

The interface is determined by the URLs used to access the catalog. All the URLs begin by BASEURL, that is the endpoint of the Catalog. TABLE is the string identifier of the table to be accessed (e.g. monitoring_information). ITEM is the numeric identifier of an element in a table. search is a reserved keyword that defines a search operation. The search operation is transferred as a querystring, of the following form.

```
search?field=value&field2=value2...
```

Each type of operation is performed as follows.

GET

Retrieves data from the Catalog. The data is retrieved in XML format.

`URL = BASEURL/{TABLE}`

Returns the id of all the items on the table `{TABLE}`.

`URL = BASEURL/{TABLE}/{ITEM}`

Returns the XML representation of the item with id `{ITEM}` in the table `{TABLE}`.

`URL = BASEURL/{TABLE}/search`

Returns the id of the items that match the search criteria.

POST

Stores a piece of data in the Catalog, creating a new entry in the database. The field information is sent in XML format in the HTTP request. The XML file can be easily generated by a properties file, using field-value pairs to specify the value of each field of the new item in the database. This operation returns a small XML fragment containing the id of the new item.

`URL = BASEURL/{TABLE}`

Creates a new item in the table `{TABLE}`, filling the fields with the information included in the request.

PUT

Replaces an existing piece of data in the Catalog. No information is returned.

`URL = BASEURL/{TABLE}/{ITEM}`

Modifies the item `{ITEM}` in table `{TABLE}` by replacing the values as indicated in the request.

`URL = BASEURL/{TABLE}/search`

Modifies all the items in `{TABLE}` that match with the search criteria by replacing the values as indicated in the request.

DELETE

Deletes a piece of data from the catalog. No information is returned.

`URL = BASEURL/{TABLE}/{ITEM}`

Delete the item `{ITEM}` from the table `{TABLE}`.

`URL = BASEURL/{TABLE}/search`

Delete all the items in the table `{TABLE}` that match with the search criteria.

Other implementation considerations

This section briefly introduces some general considerations of particular relevance about the implementation of Cloudcompaas. The use of REST decouples the components and unifies the interfaces using standard technologies for the exchange of information. The communication mechanisms between components is provided as a set of operations in a library included in every Cloudcompaas module. The library includes commands to perform REST operations on target URLs or component type, and operations to easily manipulate the XML documents exchanged.

Components of the platform need to maintain a group view. The group view consists on the knowledge of the endpoint reference of other components so that requests can be sent to the appropriate module as needed. This problem is solved by registering all running components in the Catalog. The communication library is then able to query the Catalog and retrieve a list of the available components. This solution however raises the problem of how does the communication system retrieve the endpoint reference of the Catalog in the

first place. The solution to this problem consists on the inclusion of a little bootstrapping step at the beginning of the execution of every component. At startup, the modules set up the communication library with the endpoint reference of at least one running Catalog that is provided beforehand. The library proceeds to register the current component in the Catalog and then retrieves the list of currently running components, resuming the execution of the module once it is ready to exchange messages.

The last relevant implementation consideration is the monitoring system that feeds the Catalog. This system is left open for third party software to be plugged in the system. For the current implementation, a custom service that periodically updates the catalog with the value of relevant metrics has been developed. Once a VM is deployed, the contextualization step sends to the service the endpoint reference of the Catalog components and begins its execution.

4.3 Dynamic Cloud resources management

The Monitor module of the SLA Manager performs the dynamic assessment of the QoS rules from active SLAs. The three basic operations of the Monitor are updating the SLA terms state, checking the guarantees state and performing self-management operations. SLAs registered in the Monitor are set to be updated every certain period of time, commonly defined as monitoring cycle. At each cycle, the Monitor performs these three operations in order. The monitoring continues until one of the following conditions becomes true:

- the SLA is completed. This condition is met if the SLA is defined for a certain period of time, or when it is defined on the basis of particular objectives (e.g. associated to an individual experiment or execution), which must be completed;
- the SLA is terminated by the consumer;

- the SLA is rejected by the provider. An accepted SLA can be rejected by the framework at any time, although this form of termination may involve penalties to the service provider.

The update operation consists on the retrieval of the status of the SLA terms from the Catalog component. Derived or external values, such as the current time and date, are also computed on this step. The check step uses the values retrieved in the previous step to determine the status of the guarantee terms. The monitor evaluates the formulas of the guarantee terms and sets the value of the guarantees to either Fulfilled or Violated. The self-management step performs the assessment operations based on the outcome of the guarantee check. For each guarantee evaluated as Fulfilled, the Monitor performs the billing operation, charging the user for the service. For each guarantee evaluated as Violated, the Monitor performs corrective actions aimed to restore the proper functioning of the service. Corrective actions are domain-specific functions that act on the configuration of the resources. This step effectively implements the QoS assessment capabilities. The WSAG4J framework has been used as the basis for the development of the SLA Manager component. This open-source framework has been extended with new components and operations to fulfil the needs of Cloudcompaas. Figure 4.7 shows the structure of the modules that have been modified in WSAG4J to perform the dynamic management of the SLA.

The first major modification to WSAG4J is that the guarantee evaluation and the SDT monitoring have been decoupled. In the original implementation of WSAG4J, SDT monitoring is scheduled at fixed periods of time, with the guarantee evaluation done right after it. However, this behavior lacks from the flexibility needed to implement the WS-Agreement specification, which allows service providers to define the monitoring cycle for each guarantee term. Therefore, the monitoring of the state and the evaluation of the guarantees must be performed independently.

In order to perform these operations, the *CloudcompaasMonitor* (1) schedules the execution of a *ServiceTermJob* (2a) and a *GuaranteeTermJob* (2b) for each SLA. These two classes are in charge of updating

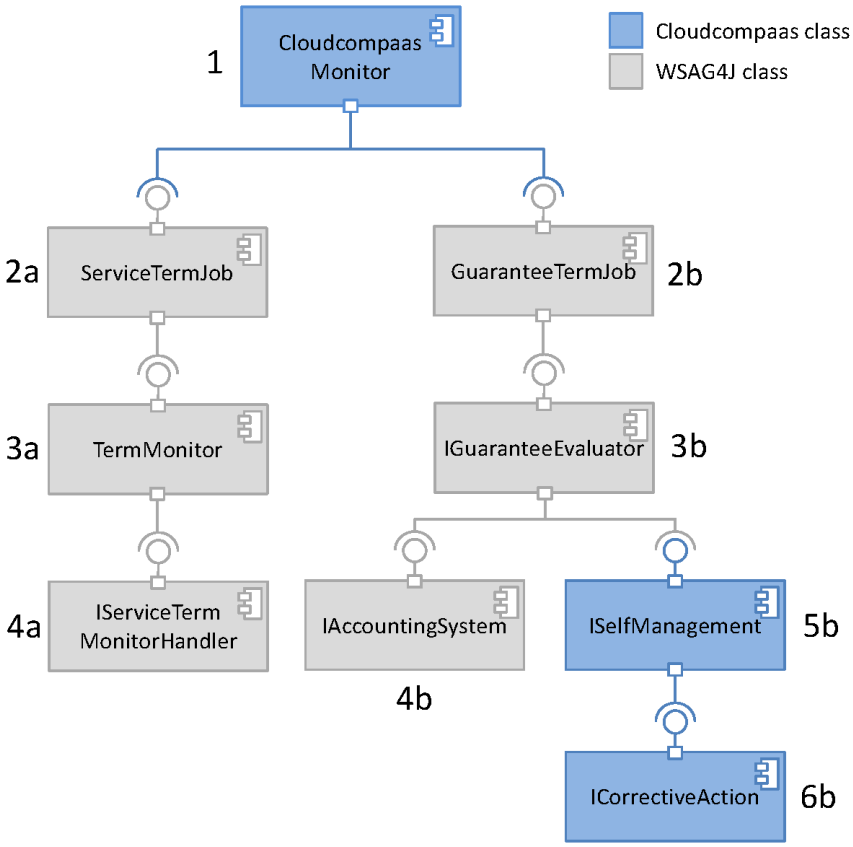

Figure 4.7: Monitor architecture

the monitoring information and evaluating the QoS rules for an SLA, respectively. They exchange information by storing and retrieving the monitoring information in a shared SLA instance, and using synchronization mechanisms to avoid race conditions.

The *ServiceTermJob* executes a *TermMonitor* (3a) at each monitoring cycle, which updates the state of the SDT. The *TermMonitor* updates the state of each individual SDT by executing each one of the available *ServiceTermMonitoringHandler* (4a), which are individual handlers designed to update the different SDT. For instance, the *VirtualMachineMonitoringHandler* is the handler that gathers the monitoring

information that concerns to the virtual machines, and decides the SDT state, accordingly.

The *GuaranteeTermJob* (2b) executes an *IGuaranteeEvaluator* (3b), which updates the state of each guarantee term of the SLA. Similarly to the original implementation of WSAG4J, the *IGuaranteeEvaluator* uses an *IAccountingSystem* (4b) to issue rewards and penalties.

One of the major limitations of the original WSAG4J approach, where SDT monitoring is scheduled at predefined times, is that in WSAG4J the same guarantee term cannot define different monitoring cycles for different business values (for instance, a 5 min interval for reward, and a 1 min interval for penalty). Cloudcompaas defines a group of *GuaranteeTermJobs* per SLA, using a possibly different monitoring cycles per job defined by the templates.

The second major modification to WSAG4J is the introduction of a self-management component in the monitoring process. In the original implementation, the *IGuaranteeEvaluator* issues penalties or rewards by means of the *IAccountingSystem*. However, Cloudcompaas needs autonomic decision making based on the SLA terms.

This capability is introduced by the *ISelfManagement* (5b) component. This component is instantiated when the *IGuaranteeEvaluator* evaluates that a guarantee state is violated and it performs the required operations to restore the state of the violated guarantee.

Each *ISelfManagement* groups a set of *ICorrectiveAction* (6b), and interfaces with the matchmaking system. The *ICorrectiveAction* provides the domain-specific implementation of the actions required to restore a violated guarantee. These actions may range from very generic, general purpose actions to application specific actions. In [107] the authors discuss the dynamic adaptation of Cloud resources, defining a hierarchy of actions to perform for different typical Cloud scenarios. Several important contributions have been made to this field in the recent years, such as the usage of a knowledge system to decide the corrective actions to execute [108].

Implementing the WS-Agreement specification requires that the guarantee states to be checked at each monitoring cycle, issuing rewards and penalties, and applying corrective actions (when needed). However, corrective action could take longer to execute than the monitoring cycle of the associated guarantee, which may cause the accumulation of corrective actions. For instance, an auto-scaling guarantee is assessed every 2 minutes. Its corrective action consists on the deployment of new virtual machines. The deployment could take more than 2 minutes to complete and when the guarantee is checked again it will still be evaluated as violated, and another corrective action will be issued. This problem may also appear when the correction of different guarantees lead to the same corrective action. The issue of the accumulation of corrective actions has a negative impact in a SLA-mediated system, because introduces an unnecessary overload and may cause violations of additional guarantees.

In Cloudcompaas, this issue has been addressed by using a cooldown approach. To this end, the *ISelfManagement* defines a cooldown for each corrective action. Every time a new request is received, the corrective action is registered as 'ongoing'. When the system receives a request for a corrective action, which is currently ongoing, then it ignores the request. Additionally, after the cooldown expires, the ongoing action is deleted from the registry.

Beside these two major modifications, other smaller changes have been made in order to increase the flexibility of the WSAG4J framework and to adapt it to the particular needs of a multi-tenant environment. For example, Cloudcompaas has modified the language used for expressing conditions in WSAG4J. One of the significant contributions to the language is the introduction of array variables and math operations as possible values for the KPI of the SLO of the guarantees. Using these variables, complex values can be expressed, such as the mean CPU % usage of all deployed virtual machines. Another example is the possibility of using expressions instead of literals in the value of rewards and penalties. This change allows SLAs expressing dynamic prices for the resources. For instance, in Cloudcompaas, it is possible to express the price as a function of the number of running Virtual Machines.

Chapter 5

Experimental results

This chapter introduces the results obtained from the set of experiments. The resource model experiments evaluate the utility achieved by the SLA composition methodology by simulating how Cloud resource requests are served by Cloudcompaas using the methodology compared to using fixed templates. The Quality of Service experiments evaluates the utility achieved by an elastic Cloud services using Cloudcompaas QoS assessment rules compared to an static Cloud services. The SLA composition experiment evaluates the performance of the SLA composition algorithm to prove that the applied optimizations provide competitive performance.

5.1 Quality of Service assessment experiments

This section introduces a set of experiments designed to check the Quality of Service assessment capabilities of the methodology. The experiment consists in the simulation of real use case scenarios on Cloudcompaas, and the measure of the performance is done in terms of the value of certain QoS indicators. Discussion is made comparing a fixed Cloud service with an elastic Cloud service.

5.1.1 Setup

The experiment setup consists on the use case described in section 1.4.

The case study simulates a service deployment, which consists on an SLA template that defines three assets: a Virtual Service, a Virtual Container where this service will be executed, and a Virtual Machine that will host the Virtual Container.

The experiments use jLinpack as a Virtual Service. The service listens to user requests for the execution of the Linpack benchmark for a variable size, defined by the user. This service is utilized to reflect user requests served by an application demanding intensive CPU and memory.

The Virtual Container is the software container that enables the execution of virtual services. In the experimental setup, the virtual container is the Java runtime.

The Virtual Machine defines the virtual hardware that hosts the virtual containers and services. In the experimental setup, the virtual machine is a 'small' predefined instance, which has 512 MB of RAM and 1 CPU core.

In the experiments, these resources are deployed by the SLA Manager on an IaaS Cloud. The IaaS backend used for the experiments is an OpenNebula deployment on a cluster composed by 8 nodes running Ubuntu Server 10.04 (x86-64) on Intel ©Xeon ©Processor L5430 (12M Cache, 2.66 GHz, 1333 MHz FSB). The nodes have 16 GB DDR3 (1333 MHz) of RAM memory and a Hard Drive Disk SATA II (7200 RPM).

5.1.2 Execution scenarios

The case study is composed by three scenarios, and each scenario is measured in two different configurations. The scenarios model the load from different scientific computing domain. The load profiles correspond to the usage of the EGI grid by three different scientific

communities [109]. The scientific domain load profiles are extracted from the real usage of the EGI infrastructure and comprise the execution of different applications of different users. Those applications involve mostly multiple batch jobs.

Two configurations have been used in the experiments. The fixed configuration provides a predefined number of Virtual Machines. The elastic configuration provides a variable number of Virtual Machines, governed by the QoS rules of Cloudcompaas.

The metrics measured in the experiments are the price of the resources and the number of failed user requests. The price is directly related to the amount of resources used by the IaaS provider to serve the experiment. The number of failed user requests is a direct reflection of the application user satisfaction with the PaaS user / application provider. This value is complementary to the number of successful user requests served.

The price of the deployment measures the cost for running the assets for a certain period of time (the span of the experiment). SLAs define the price applied to each resource in their guarantee terms. These guarantees are evaluated at each monitoring cycle of 5 seconds. The cost a running a single Virtual Machine for 24 hours is 1.44 €. This price is the same than that of an Amazon EC2 small instance [39].

However these metrics does not enable measuring the tradeoff between price and failures made between different configurations. Usually the elastic configuration provides a lower value at the cost of increasing the number of failed user requests. Whether the tradeoff is positive or negative depends on the expected revenue produced by each user, or conversely the profit lost for not serving users compared with the money saved by reducing the price. In order to properly compare different configurations, derived metrics are calculated for each scenario.

The average expected revenue per user r is defined as the total revenue divided by the number of users served. Total revenue is the revenue obtained by the provider from the users of the service. The number of users served is the number of total users that accessed the service

minus the number of failed users. Failed users are the ones that accessed the service but ultimately were unable to finish they requests successfully.

$$r = \frac{total\ revenue}{total\ users - failed\ users} \tag{5.1}$$

This value represents the average money the service provider expects to obtain out of each individual user. Using this derived metric, it is possible to calculate the profit made by different configurations with total users t and failed users f.

$$profit = r * (t - f) - price \tag{5.2}$$

Profit can be used to properly compare both configurations and determine which one perform best. However, profit is determined by r, which is a value that depends on the service offered. For the experiments, three derived metrics based in extreme values of r have been calculate, quantifying the performance difference for each configuration.

The first derived metric is the break-even point Be. This is the value of r for which both configurations provide the same profit.

$$(t - f_f) * Be - p_f = (t - f_e) * Be - p_e \tag{5.3}$$

$$Be = \frac{p_f - p_e}{f_e - f_f} \tag{5.4}$$

For f_f and f_e the failed requests for the static and elastic scenarios respectively and p_f and p_e the price for the static and elastic scenarios respectively. This value indicates that for $r < Be$ the elastic configuration outperforms the static one. The higher the value, the wider the range of profits for which the elastic configuration dominates. How-

ever, Be is an absolute value, and hence it does not provide enough information to properly compare both configurations.

The second derived metric is the ratio of over performance Or. This value represents the ratio of profit (i.e. difference between revenue and infrastructure cost) for which the elastic configuration provides better performance than the static one. In order to derive Or from Be, the value of r for which the profit is 0, $r0_f$, is needed.

$$r0_f = \frac{p_f}{t - f_f} \tag{5.5}$$

$$Or = \frac{Be}{r0_f} \tag{5.6}$$

Therefore, whenever the ratio of profit of the static scenario is lower than Or, the elastic scenario will outperform the static one.

The third derived metric is the ratio of profitability Pr. It might happen that, since the price of the elastic configuration is lower, a value for r that is not profitable in the static configuration is indeed profitable in the elastic one. The profitability ratio represents the percentage of values of r that are profitable for the elastic scenario but they are not for the static one. The higher the value, the wider the range of revenues for which the elastic scenario outperforms the static one. In order to calculate the ratio, the values of $r0$, $r0_f$ and $r0_e$, for which the profit of the static and elastic configuration is 0 need to be calculated.

$$r0_e = \frac{p_e}{t - f_e} \tag{5.7}$$

$$Pr = 1 - \frac{r0_e}{r0_f} \tag{5.8}$$

Moreover, the sign of Be and Pr give information about the general behavior of the elastic configuration respect the fixed one. If both values are positive, the elastic configuration outperforms the fixed one

for $r < Be$. If Be is negative and Pr is positive, the elastic configuration outperforms the fixed one for all values of r. If both values are negative, the elastic configuration will never outperform the fixed one.

5.1.3 Experimental results and discussion

The three modelled scenarios correspond to the user load profile of Grid users in the fields of Chemistry, High-Energy Physics and Fusion. The Chemistry scenario begins with a very high user load that steadily drops until the half of the experiment, where it begins to rise again. At about three quarters of the experiment the load falls. The High-Energy Physics scenario begins with a very high user load, and keeps about the same load all along the experiment until the end. The Fusion scenario begins with a very low load, until about three quarters of the experiment where the user load peaks to the maximum value and drops again almost immediately to the previous values.

For each one of these scenarios the performance is measured in two different configurations. One configuration consists on allocating statically the minimum number of replicas needed to serve the maximum load on the experiment. The second configuration consists on allocating an initial number of replicas that are managed by Cloudcompaas. Cloudcompaas dynamically asses the QoS rules for the Virtual Service, which specify the adequate load regime for a proper performance. Using these rules the framework allocates and deallocates replicas on the fly to balance the current service load.

Each experiment is illustrated with a figure that includes the number of requests, number of replicas and failed requests per unit of time. A request is a user interaction with the service, while a failed request is a user interaction that times out unfinished. The total number of requests, price and failed requests for each experiment is included, and the derived metrics Be, Or and Pr calculated. Table 5.1 summarizes the metrics calculated for each experiment. Discussion in base of the value of these metrics is made in order to estimate the relative performance of both configurations for each scenario.

Scenario	Config.	Total requests	Failed requests	Price	*Be*	*Or*	*Pr*
Chemistry	Fixed	247,971	40	10.05€	0.0082€	20,229%	70.44%
	Elastic		901	2.96€			
High-Energy Physics	Fixed	565,259	122	10.22€	0.0071€	39,261%	2.54%
	Elastic		158	9.96€			
Fusion	Fixed	174,467	150	10.06€	0.1680€	291,106%	66.89%
	Elastic		190	3.33€			

Table 5.1: Summary of experimental results.

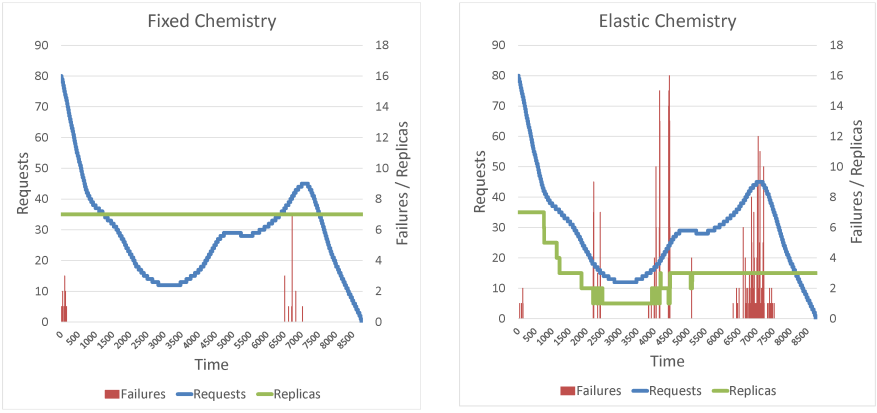

Figure 5.1: Experiment for the Chemistry scenario.

Figure 5.1 depicts the experimental results for the Chemistry scenario. Chemistry has an execution profile that exhibits a moderate variable load, with two alternating peaks and valleys of high and medium/low load. For the fixed configuration, 7 machines are able to accommodate the load peaks, producing 40 failed requests out of 247,971, which enters the random failure margin[1]. The total price is 10.05€, which corresponds to the number of credits needed to keep 7 machines running for roughly 24 hours. This price is similar for all three fixed

[1]Random failures correspond to failed requests due to network, computing or other errors independent from server capacity. Experiments suggests a random failure margin of 0∼200 for the presented scenarios.

scenarios, since the same number of machines is run for the same amount of time.

For the elastic configuration the number of failures is 901, more than 20 times higher than the fixed configuration. These failures come from the delay between the framework detecting an overload of the computational resources and the deployment of extra replicas to accommodate this load. Fine-tuned QoS rules or the usage of predictive models along with reactive techniques may reduce this delay and the number of extra failed requests.

On the other hand, the price for this configuration is 2.96€, less than a third part of the fixed configuration. This reduction in price comes for the downscale of the infrastructure when the load is not at its peak value. Since the load is at sub-peak values for most of the experiment, by turning off unused machines great savings are achieved.

Even though a reduction of a third of the price at the cost of 20 times more failures may intuitively seem a bad trade, the derived metrics show that the elastic configuration performs well. The Be is 0.0082, with a Or of 20,229% and a Pr of 70.44%. These values indicate that even though the failures increase considerably, the reduction in price makes up for the revenue lost by not serving the clients. Assuming that all the clients are equally valuable, a service would need a profit higher than 20,229% the price of the Cloud service in order for the fixed configuration to outperform the elastic one. Also the Pr of 70.42% shows that there is a wide range of services that are not profitable under the fixed configuration, but are profitable under the elastic one.

Figure 5.2 depicts the experimental results for the High-Energy Physics scenario. It has an execution profile that exhibits an almost constant high load with little variability. The fixed configuration produces 122 failed requests for a price of 10.22€, while the elastic scenario yields 158 failures for a price of 9.96€.

The behavior of both configurations is very similar on this scenario. Since the load variability is very low, the elastic scenario keeps running 7 machines during the complete duration of the experiment, up until

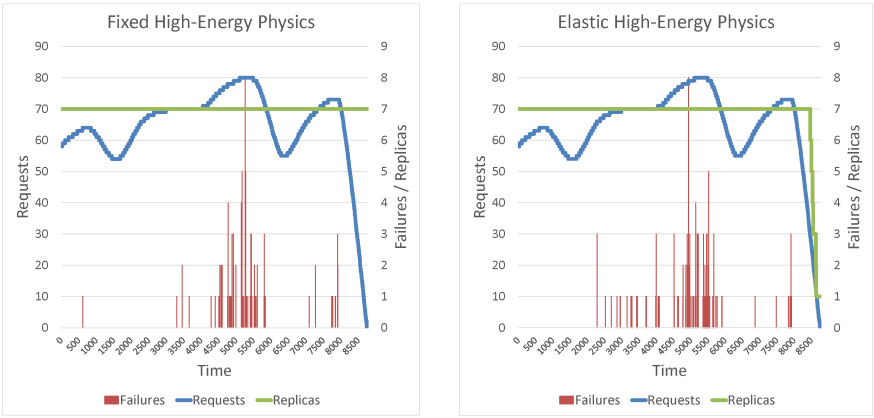

Figure 5.2: Experiment for the High-Energy Physics scenario.

the end when the load falls to 0. The elastic scenario therefore achieves a small saving in the cost while providing virtually the same number of failures, since both experiment are within the random failure margin.

This fact is reflected on the derived metrics. The elastic configuration has a negligible Pr of 2.54%, since it essentially behaves as the fixed configuration. The Be is 0.0071 and the Or 39,261%, which is sensibly lower than the Chemistry scenario. Nevertheless the value of Or indicate that the little savings obtained by the elastic scenario for the same number of failures produces a positive performance.

Figure 5.3 depicts the experimental results for the Fusion scenario. Fusion has an execution profile that exhibits a low load for most of the time, with a peak of very high load on the second half of the profile. This peak last for a small fraction of the total duration, and it has the steepest variation of all the experiments. The fixed configuration produces 150 failed requests with a total price of 10.06€, while the elastic scenario produces 190 failures for a price of 3.33€.

The most notable comparison is that the number of failures is very similar in both cases. This fact is due to the ability of the framework to detect an increase in the load and react quickly deploying

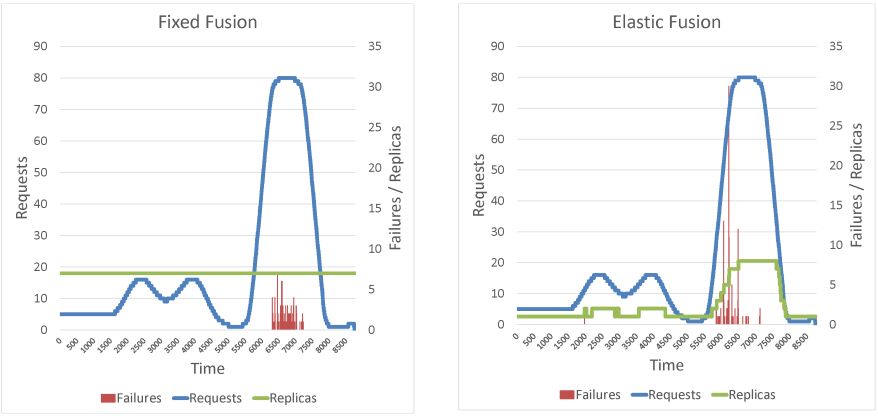

Figure 5.3: Experiment for the Fusion scenario.

new replicas when the load peek occurs, with a small number of failures happening as the load goes up. The fixed configuration of this scenario is a clear case of overprovisioning, since the resources are idle for most of the execution. Overprovisioning scenarios are favorable for an elastic configuration, since great savings could be achieved by turning off idle resources. The experimental metrics back off this intuitive appreciation, as the price of the elastic scenario is roughly one third of the fixed one.

Derived metrics indeed show that this is the scenario for which the elastic configuration achieves the best performance. With a Be of 0.1680 and a Or of 291,106%, that is, around 17 times higher than the Chemistry scenario. This is due to the fact that the elastic configuration yields almost the same number of failures than the fixed configuration, but provides a substantial saving in price. On the other hand the Pr is 66.89%, comparable to the value obtained for the Chemistry scenario.

Some general conclusions can be drawn from the experimental results obtained from the different scenarios. First, the experimental results show the general tendency of elastic configurations to yield a higher number of failures, due to the delay in the deployment of new resources

in response to load increases, and lower price, due to the deployment of less resources when the load is low. Second, for services that serve a large number of users and rely on economies of scale to provide profits, the tradeoff between failures and cost is positive, since the opportunity cost of not serving users is lower than the money saved in the infrastructure. Third, the performance of the elastic configuration depends on the load profile of the service. For a load with little variance, the elastic configuration provides little advantage. For an average variable load, the elastic configuration provides a large performance improvement. The best scenario occurs for very irregular loads with large peaks, where the elastic configuration vastly outperforms the fixed one by avoiding overprovisioning of resources. Fourth, the Pr values for the variable load profiles show that there is a wide range of services that not being profitable under an fixed configuration might indeed be profitable by adopting an elastic configuration.

5.2 Resource model experiments

This section presents a set of experiments designed to check the advantage provided by the methodology regarding Cloud services modeling. The experiment consists in the simulation of user request for VMs in a Cloud service and the measure of the performance, in terms of the value of certain indicators, such as cost and number of failed requests. Discussion is made concerning the utility achieved using static templates for the provision of resources compared to the utility achieved using composed SLA templates.

The experiment setup consists on the simulation of a medium size Cloud infrastructure. The experiment defines two scenarios, one using static templates for the VM configuration and other using composed SLA fragments. Each user requests from the framework a Virtual Machine with certain memory requirements, CPU requirements and Time To Live (TTL), and the framework answers providing with the smallest VM configuration that complies with the required resources. The first scenario (static templates) defines seven static VM configuration templates, as illustrated in Table 5.2. This scenario is analog to

VM Size	Memory (GB)	Cores
1	2	1
2	4	2
3	8	4
4	16	8
5	32	16
6	64	32
7	128	64

Table 5.2: VM resources for the experiments.

current available commercial Cloud providers. Cloud providers usually offer a limited set of resource alternatives. This limited offer is justified by the providers by the ease of managing a small number of alternatives as well as simplifying the decision for the user. The fact of using resource sizes which are multiples of one another is usually justified by the assumption that these sizes lower the fragmentation of the computing cluster memory and cores when Virtual Machines are allocated.

The second scenario (composed templates) defines 64 memory templates (from 2 to 128 in steps of 2) and 64 cores templates (from 1 to 64). These templates are composed covering all possible pairs a total of 64*64=4096 combinations. The simulation setup considers requirements up to 128 GB RAM and 64 cores. Each scenario can be measured on one side from the Cloud user point of view and on the other side from the Cloud provider point of view. This duality allows the experiment to provide a global vision of the potential advantages and disadvantages of both approaches.

From the Cloud user point of view, the indicators of the performance of the Cloud infrastructure are the cost of running the Virtual Machine and the number of SLA violations. From the Cloud provider point of view, the indicators of performance of the Cloud infrastructure are the number of active physical nodes and the number of rejected user requests.

5.2.1 Client-side setup

The cost of running a VM is the cost of the resources utilized multiplied by the running time. The number of SLA violations is related to the expected Quality of Service in the delivery of the resources. The QoS target on this experiment has been defined as the resources provided with respect to the resources required by the user. However, VM may produce peaks of load that go beyond its capacity. Since the experiments do not model a Cloud platform that provides QoS assessment, every time the VM load goes beyond its capacity, this behavior is considered a SLA violation. In order to model SLA violations, it is necessary to define its frequency and magnitude. The frequency of violations is simply defined by a probability γ, and the magnitude of the violation is defined using a normal distribution with mean μ and standard deviation σ. At each simulation cycle, a random roll is done on this probability to decide whether a violation occurs, and a second random roll on a normal distribution N(μ, σ2) decides the magnitude of the violation.

Client-side results and discussion

The simulation experimental results under the user point of view are summarized in Table 5.3. The values obtained for the cost and violations are in concordance with the model of the scenarios. The static template scenario incurs in a higher cost since the user has less flexibility to choose the configuration of the templates, as opposed to the composed scenario. On the other hand, the number of violations is significantly lower, since the difference between the required and the provided resources is higher in the static templates scenario than in the composed templates. This difference represents the spare resources that user applications may use before a QoS violation is raised.

The value of cost per violation is used to provide a fair comparison of the performance of between the different approaches. The cost per violation is the product of the cost and the average number of violations, and represents the balance between both parameters. The value of this parameter is 170% higher on the composed scenario, represent-

Scenario	Cost	Violations
Static	0.61	3.78
Composed	0.46	13.29

Table 5.3: Results from the user point of view.

Scenario	Cost	Violations
Static +20%	0.78	0.60
Composed +20%	0.51	0.68

Table 5.4: Results from the user point of view for the overprovisioning scenarios.

ing that the reduction of the cost is outweighed by the increase in the number of violations. Since SLA violations depend on the surplus of resources provided with respect to the resources required, this problem can be mitigated by enforcing SLAs to perform an overprovision of resources. Overprovisioning consists on assigning to the users a certain quantity of resources on top of the user request. This technique may end up assigning larger templates or the same templates to each request. The composed scenario is prone to provide larger templates, since the range of resources is smaller. The static scenario is prone to provide the same template, since the range of resources is larger, and this usually conveys an implicit overprovisioning of resources. Table 5.4 summarizes the experimental results obtained with an overprovisioning factor of 20% of the user request.

The results show that increasing the number of allocated resources effectively reduces the number of violations, incurring in a higher cost. However, the reduction on the number of violations in the composed scenario is notably higher than in the static scenario. On this case the cost per violation is 34% higher on the static scenario. This value represents that, using a 20% overprovisioning value, the composed scenario offers better performance, balancing out the increase in the number of violations with a lower cost.

5.2.2 Provider-side setup

The experiment simulates a cluster where clients allocate Virtual Machines on demand. Clients can request the allocation of a VM or shut down a running one at any time. The time that a particular VM is running is defined as its Time To Live (TTL). The rate at which clients request the deployment of a new VM is defined as the arrival rate of requests. The number of active physical nodes on a cluster defines how much energy the cluster consumes, and hence is a factor in the monetary cost of the infrastructure management. This value is averaged on the length of the experiment. The number of rejected user requests represents the number of VM requests from users that cannot be served since there are not enough free resources. If no machine in the cluster has enough free resources to allocate a user request, it is rejected. This typically occurs when the infrastructure is almost at full capacity.

The simulation runs on discrete units of time defined as simulation cycles. At each cycle, the arrival rate defines the chance that a user requests the deployment of a new VM (e.g. an arrival rate of 0.9 implies a 90% chance that a new request is issued). The VM Memory, Cores are drawn at random following a uniform distribution. The TTL is drawn from an exponential distribution such that 99% of them is lower than a certain value defined in the simulation, the 99 percentile TTL. When a request is received, the platform chooses the physical node that will host the VM using the best fit strategy. Best fit selects a candidate host such that it has enough free resources to allocate the VM and the spare resources left after the allocation are the minimum of all possible candidates, that it, it selects the machine that fit best the request. In case no active physical node has spare resources to allocate the VM, a new physical node is added to the cluster active node list. In case no more nodes are available in the cluster, the user request is rejected. At each cycle the simulation assesses the VM state, calculating possible violations, cost and diminishing its TTL. Once the TTL of a VM reaches 0, the VM is eliminated. If the hosting physical node becomes empty due to this operation, the node becomes inactive.

Parameter	Small	Default	Large
Cores	40	80	160
Memory	80 GB	160 GB	320 GB
Arrival rate	0.1	0.5	0.9
99% TTL	500	1000	2000

Table 5.5: Parameters of the cluster configuration.

The hardware configuration of the simulation mimics that of a real Cloud infrastructure from a Google datacenter. Section 6.7 of [110] shows how a rack from the Google datacenter in Las Vegas is composed by 20 servers, each one with two processors AMD Barcelona (dual core) and 8 GB of RAM, for a total of 20 machines, 80 cores and 160 GB of RAM. Preliminary tests proved that the behavior of the platform is highly dependent on the configuration of the parameters of the simulation. In order to generalize the conclusions of the experiments, several cluster configurations have been used to measure the value of the performance indicators.

Table 5.5 summarizes the configuration values used on the simulations. The values affecting the performance of the experiments are the capacity (number of cores and memory) per machine, the arrival rate and the 99 percentile TTL. Each experiment set all parameters to the default value except for one. Then, two experiments are performed setting the free parameter to the small and large value respectively. This method enables comparing the impact on the experiment outcome of the variation each parameter individually.

Provider-side results and discussion

The experimental results under the provider point of view are illustrated at Figure 5.4. The experiments with different simulation values enable comparing the performance of the scenarios in several situations and draw conclusions about the general behavior of both approximations. Notice that the overprovisioning ratio has not been included in the experiments, since from the provider point of view the overprovi-

sioning is transparent, as he only sees requests for resources. Experiments show distinctive trends in the performance of the scenarios.

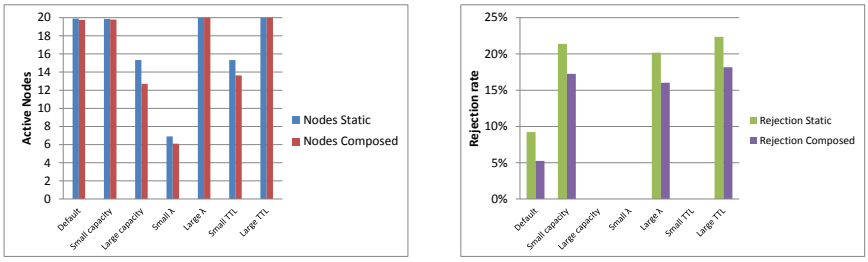

Figure 5.4: Active VM and rejection rate for each scenario and configuration.

The first trend shows that when the number of active VM is lower than 20, the experiments produce almost no rejections. This occurs because the cluster is not working at full capacity i.e. it has spare resources to accommodate user requests. Conversely when the number of active VM is close to 20 the rejection rate increases dramatically. This occurs because the cluster is at full capacity and it cannot accommodate more user requests. The second trend shows that the values obtained for the composed scenario are consistently lower than the static scenario for both rejection rate and number of active VM. This trend implies that the composed scenario is able to fit more user requests using the same number of machines. This ability to fit more users in the same number of machines implies that when the cluster is not at full capacity, less machines are needed to serve users, and when the cluster is at full capacity, more users can be served with the same number of machines.

5.3 Algorithm Performance

This set of experiments explores the performance of the execution of SLA composition algorithm. These experiments explore the impact of the number of restrictions imposed by a user in the execution

time of the optimized version of the algorithm. Notice that the non-optimized versions of the algorithm run on exponential time, making the comparison with the optimized version unpractical. The aim of the experiment is to measure the performance of the best version of the algorithm available. This implementation includes the following optimizations.

- branch and bound by size;
- dynamic programming;
- guided solution space exploration;
- trimming by semantic constraints;
- trimming by user constraints.

The cost of the algorithm has been measured in steps, where each step is a recursive invocation of the algorithm main body. The experiments have been performed by issuing user request with varying number of restrictions and number of SLA fragments of each resource category. A user restriction is a condition imposed in the composition algorithm. For instance, the following conditions are restrictions, VM = small, Runtime = Java, Service = jLinpack. Restrictions reduce the search space in the composition algorithm, and therefore reduce execution cost. The experiment aims to measure how introducing more restrictions impacts on the execution cost. The number of SLA fragments represents how many SLA the framework stores for each resource category. A value of 4 indicates 4 SLA fragments for VM, Runtime, Service, etc. The motivation is that each user request explores a different set of resources, and therefore increasing the number of only one category of resources will produce varying results for different search executions. The experiment aims to measure how the global grow of the number of SLA fragments and the restrictions imposed by the user impact the execution cost.

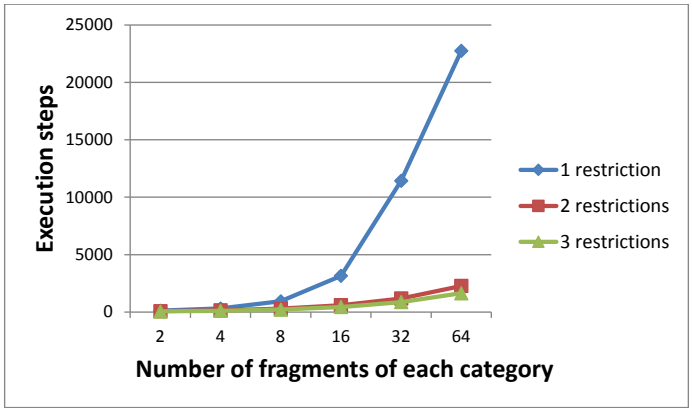

Figure 5.5: Execution time for the SLA composition algorithm for different number of SLA fragments and user query parameters.

5.3.1 Algorithm Performance results and discussion

Figure 5.5 shows the execution steps obtained for different user queries and number of template fragments for the fully optimized algorithm.

The number of execution steps grows with the size of the search space. However the progression of the growth differs significantly for different cases. For queries with one restriction, a polynomial regression fits the growth curve to a second degree polynomial with a correlation coefficient of 0.999999, and queries with two and three restrictions fit to a one degree polynomial with a correlation coefficient of 1.

The algorithm reduces the execution steps as more restrictions are added. This tendency become irregular between one and two restrictions, where the number of execution steps greatly decreases reducing the asymptotic cost from $O(n^2)$ to $O(n)$. This experiment shows the unpredictable nature of the branch and bound algorithm in the number of execution steps for different cases. Even though the theoretical cost of the algorithm is $O(2^n)$, the experimental cost of the algorithm for typical inputs approximates $O(n^2)$, and can go down to $O(n)$ for some input restrictions.

Chapter 6

Concluding remarks

This Thesis introduces a novel methodology for the representation of Cloud services using Service Level Agreements. This methodology provides several advantages such as an independent definition of each element, composition of SLAs and lower cost for domain representation. This methodology avoids ad-hoc formulations since it is based on available standards. It provides SLA-driven Cloud platforms with an integrated solution to the problem of the representation Cloud services. As a fundamental component of the methodology stands the SLA composition algorithm. This Thesis introduces the design and optimized implementation of this algorithm, which enables on-the-fly composition of SLA fragments in order to produce complete SLA templates, and captures the dynamic nature of a Cloud environment.

The proposed methodology also tackles the static and dynamic features of Cloud services. A model for the representation of resources and their dependences has been defined. Also general mechanisms for the management of the QoS and dynamic aspects of Cloud services has been developed and tested.

A use case on a simulated environment shows how composed SLA documents are able to represent information more concisely and provide better utility than fixed SLA documents. Users are able to reduce

the cost by adjusting the resource request to the expected resource usage at a finer scale. Providers are able to serve more users using fewer machines, obtaining greater profit by reducing cost and increasing revenue. The methodology also encompasses open problems that provide further research work into this topic. Future works include a restriction representation system, including more complex relationships between elements and restrictions checking at a semantic level.

This Thesis introduces Cloudcompaas, a SLA-driven Cloud framework. Cloudcompaas features an extension of the WS-Agreement SLA specification providing an SLA-driven management of the complete Cloud resource lifecycle. Finally, a complete working prototype has been introduced as a proof of concept, showcasing an elastic deployment. A set of experiments for different load profiles and configurations measure key metrics that quantify the performance improvement provided by using an elastic configuration.

The integral approach followed by the Cloudcompaas project features some open problems that originates a wide variety of research lines. These research lines includes but are not limited to a negotiation protocol for the establishment of SLA, the design of monitoring systems, a decision making system for the election of the corresponding corrective action for each type of violation, distributed SLA monitoring and a disaster recovery protocol.

Appendix A

Curriculum Vitae

Andrés García García started working in the High Performance and Grid Computing Group (GRyCAP) in November 2007 as an undergraduated. Upon obtaining the bachelor in computer science by the Universitat Politècnica de València (UPV) in September, 2008, he joined the group as a graduate under a collaboration fellowship. He obtained the Master Degree in September 2010 with the Master Thesis Cloud computing PaaS Platform - Cloudcompaas. Since then, he continued developing the Cloudcompaas framework specifically towards the field of Service Level Agreements as the subject of his Ph.D. Thesis.

Participation in research projects

Title: Organización y Puesta en Marcha de la red de e-ciencia en España (CAC-2007-52)
Entity: Universitat Politècnica de València.
Duration: 3 years.

Title: Supporting and structuring Healthgrid activities & research in Europe: Developing a roadmap.

Entity: European Commission.
Duration: 2 years and 4 months.

Title: Biomasa@UPV, set up of a Desktop Grid infrastructure for the optimization of energy resources.
Entity: GRyCAP, Instituto ITACA, UPV, in colaboration with the Instituto de Ingeniería Energética (IEE), UPV.
Duration: 6 months.

Fellowships

Name: Beca para la Formación de Personal Investigador de carácter predoctoral.
Funding entity: Generalitat Valencia.
Fellowship aim: Carrying out the Master Degree and Ph.D. Thesis.
Duration: 04/2009∼04/2013
Destination entity: Instituto ITACA, UPV; Instituto I3M, UPV.

Name: Collaboration fellowship.
Funding entity: Universitat Politècnica de València.
Fellowship aim: Carrying out the Ph.D. Thesis, participation in other research projects.
Duration: 10/2008∼03/2009
Destination entity: Instituto ITACA, UPV.

Name: Specialization fellowship.

Funding entity: Universitat Politècnica de València.
Fellowship aim: Carrying out the Computer Science degree Dissertation.
Duration: 12/2007∼09/2008
Destination entity: Instituto ITACA, UPV.

Participation in conferences, workshops and other scientific events

Name: 3ª Reunión Plenaria de la Red Española de e-Ciencia.
Place: Universitat Politècnica de València, Valencia, Spain, 29-30/10/2009.
Organizer: Grupo de Grid y Computación de Altas Prestaciones, GRyCAP, UPV.
Participation: Organization.

Name: Cracow Grid Workshop 2009.
Place: AGH University of Science and Technology, Kracow, Poland, 12-14/10/2009.
Organizer: ACC Cyfronet AGH, Institute of Computer Science AGH, IFJ PAN
Participation: Speaker.

Name: Ibergrid 2009
Place: Universitat Politècnica de València, Valencia, Spain, 20-22/05/2009.
Organizer: Grupo de Grid y Computación de Altas Prestaciones, GRyCAP, UPV.
Participation: Organization, Speaker.

Name: I Jornadas Ibéricas de Supercomputación.
Place: Universitat Politècnica de València, Valencia, Spain, 19/05/2009.
Organizer: Centro Informático Científico de Andalucía (CICA) Junta de Andalucía.
Participation: Organization.

Name: IX Jornadas de seguridad RedIRIS: Cloud computing.
Place: Universitat Politècnica de València, Valencia, Spain, 10/03/2011.
Organizer: RedIRIS.
Participation: Attendee.

Name: 6th International Conference on Software and Data Technologies, ICSOFT 2011.
Place: Universidad de Sevilla, Sevilla, Spain, 18-21/06/2011.
Organizer: Institute for Systems and Technologies of Information, Control and Communication (INSTICC).
Participation: Speaker.

Courses

Name: Cloud computing: Tecnologías y Herramientas para Trabajar en la Nube.
Organizer: Universitat Politècnica de València.
Duration: 32 hours.

Publications

Paper "Design of a Platform of Virtual Service Containers for Service Oriented Cloud Computing", CGW 2009 Proceedings. March 2010.

Paper "Biomass@UPV: Computacional Resources Optimization of GIS-based Applications using a BOINC Infrastructure", 3rd Iberian Grid Infrastructure Conference Proceedings, May 2009.

Paper "Overview of current commercial PaaS platforms" has been sent to the Workshop "IWCCTA 2011 - International Workshop on Cloud computing, Technology and Applications", inside the framework of the conference "ICSOFT 2011 - 6º International Conference on Software and Data Technologies", July 2011

Paper "Towards SLA-driven Management of Cloud Infrastructures to Elastically Execute Scientic Applications", Miguel Caballer, Andrés García, Germán Moltó, and Carlos de Alfonso, Ibergrid 2012.

Paper A. García García, et al., "SLA-driven dynamic cloud resource management". Future Generation Computer Systems (2013), http://dx.doi.org/10.1016/j.future.2013.10.005

Paper Andrés García García, and Ignacio Blanquer Espert, "Cloud domain representation using SLA composition". Journal of Grid Computing, under review.

Paper Toni Mastelić, Andrés García García, and Ivona Brandic, "Towards Automatic Management of XaaS Offerings". CCGRID 2014, under review.

Paper Miguel Caballer, et al., "A Platform to Enable Execution of Programming Models on the Clouds". Journal of Systems and Software, under review.

Research visits

Duration: 01/02/2013∼30/04/2013
Host institution: Distributed Systems Group, Technische Universität Wien
Topic of the visit: Integration of the M4Cloud tool with the Cloudcompaas framework.

Bibliography

[1] P. Mell and T. Grance, "The NIST definition of cloud computing (draft)," *NIST special publication*, vol. 800, no. 145, p. 7, 2011.

[2] The Open Cloud Manifesto, "Cloud Computing Use Cases White paper," tech. rep., The Open Cloud Manifesto, 2010.

[3] The VENUS-C consortium, "Virtual Multidisciplinary Environments Using Clouds," 2012.

[4] SIENA, "SIENA Roadmap on Distributed Computing Infrastructure for e-Science and Beyond Europe," tech. rep., SIENA, 2012.

[5] P. Deussen, K.-P. Eckert, L. Strick, and D. Witaszek, "Cloud Concepts for the Public Sector in Germany-Use Cases," tech. rep., Fraunhofer Institute FOKUS, 2011.

[6] R. Nyrén, A. Edmonds, A. Papaspyrou, and T. Metsch, "Open Cloud Computing Interface - Core," tech. rep., Open Grid Forum, 2010.

[7] D. M. T. Force, "Cloud Infrastructure Management Interface," tech. rep., Distributed Management Task Force Inc., 2013.

[8] SNIA, "SNIA Cloud Data Management Interface," 2010. http://www.snia.org/tech_activities/standards/curr_standards/cdmi [Online; accessed 12-12-2013].

[9] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Gener. Comput. Syst.*, vol. 25, pp. 599–616, June 2009.

[10] A. McCloskey, B. Simmons, and H. Lutfiyya, "Policy-based dynamic provisioning in data centers based on SLAs, business rules and business objectives," in *Network Operations and Management Symposium, 2008. NOMS 2008. IEEE*, pp. 903–906, 2008.

[11] Q. He, J. Yan, R. Kowalczyk, H. Jin, and Y. Yang, "Lifetime service level agreement management with autonomous agents for services provision," *Inf. Sci.*, vol. 179, pp. 2591–2605, July 2009.

[12] S. Hudert, H. Ludwig, and G. Wirtz, "Negotiating SLAs-An Approach for a Generic Negotiation Framework for WS-Agreement," *Journal of Grid Computing*, vol. 7, no. 2, pp. 225–246, 2009.

[13] M. Chhetri, J. Lin, S. Goh, J. Yan, J. Y. Zhang, and R. Kowalczyk, "A coordinated architecture for the agent-based service level agreement negotiation of Web service composition," in *Software Engineering Conference, 2006. Australian*, pp. 10 pp.–, 2006.

[14] O. Rana, M. Warnier, T. B. Quillinan, and F. Brazier, "Monitoring and Reputation Mechanisms for Service Level Agreements," in *Proceedings of the 5th international workshop on Grid Economics and Business Models*, GECON '08, (Berlin, Heidelberg), pp. 125–139, Springer-Verlag, 2008.

[15] L. Eyraud-Dubois, H. Larchevêque, *et al.*, "Optimizing Resource allocation while handling SLA violations in Cloud Computing platforms," in *IPDPS-27th IEEE International Parallel & Distributed Processing Symposium*, 2013.

[16] V. Emeakaroha, T. Ferreto, M. Netto, I. Brandic, and C. De Rose, "CASViD: Application Level Monitoring for SLA

Violation Detection in Clouds," in *Computer Software and Applications Conference (COMPSAC), 2012 IEEE 36th Annual*, pp. 499–508, 2012.

[17] I. Breskovic, J. Altmann, and I. Brandic, "Creating standardized products for electronic markets," *Future Generation Computer Systems*, vol. 29, no. 4, pp. 1000 – 1011, 2013.

[18] A. García, C. de Alfonso, and V. Hernández, "Design of a Platform of Virtual Service Containers for Service Oriented Cloud Computing," in *Cracow Grid Workshop '09 Proceedings*, (Cracow, Poland), pp. 20–27, ACC CYFRONET AGH, 2010.

[19] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu, "Web Services Agreement Specification (WS-Agreement)," tech. rep., Global Grid Forum, Grid Resource Allocation Agreement Protocol (GRAAP) WG, Sept. 2011.

[20] Fraunhofer Institute SCAI, "WSAG4J: The WS-Agreement for Java framework," 2011.

[21] J. Markoff, "Internet Critic Takes on Microsoft," *The New York Times*, April 9, 2001.

[22] E. Schmidt, "Conversation with Eric Schmidt hosted by Danny Sullivan , Search Engine Strategies Conference," August 9, 2006.
http://www.google.com/press/podium/ses2006.html [Online; accessed 12-12-2013].

[23] B. contributors, "Why Can't We Compute in the Cloud?," *The New York Times*, August 24, 2007.

[24] P. U. s Center for Information Technology Policy, "Computing in the Cloud Workshop," Januray 14-15, 2008.

[25] eBay Inc., " eBay ," 2010.
http://www.ebay.com/ [Online; accessed 12-12-2013].

[26] Google Inc., " Google Docs ," 2010. http://docs.google.com/ [Online; accessed 12-12-2013].

[27] Google Inc., "Google App Engine," 2012. http://code.google.com/appengine/ [Online; accessed 12-12-2013].

[28] L. Ellison, "Quote from Larry Ellison," September 26, 2008.

[29] Gartner Inc. , " Gartner's 2009 Hype Cycle Special Report Evaluates Maturity of 1,650 Technologies ," tech. rep., Gartner Inc., August 11, 2009.

[30] Microsoft Corporation, "Windows Azure," 2012. http://www.microsoft.com/windowsazure/ [Online; accessed 12-12-2013].

[31] Sun Microsystem , " Network.com, posteriormente renombrado Sun Cloud ," 2010. [Offline. Sun Cloud fue cancelado tras la adquisición de Sun Microsystem por parte de Oracle Corporation].

[32] International Business Machines Corp. , " Blue Cloud ," 2010. http://www-03.ibm.com/press/us/en/pressrelease/22613.wss [Online; accessed 12-12-2013].

[33] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "Above the Clouds: A Berkeley View of Cloud Computing," Tech. Rep. UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[34] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A break in the clouds: towards a cloud definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, pp. 50–55, Dec. 2008.

[35] University of Chicago , " Nimbus ," 2010. http://www.nimbusproject.org/ [Online; accessed 12-12-2013].

[36] Abiquo , " Abiquo ," 2010. http://www.abiquo.com/ [Online; accessed 12-12-2013].

[37] e. a. Keith Jeffery, Burkhard Neidecker-Lutz, "The Future Of Cloud Computing," tech. rep., European Commission, 2009.

[38] Distributed Management Task Force, "Open Virtualization Format," tech. rep., Distributed Management Task Force Inc., 2013.

[39] Amazon.com Inc., "Amazon EC2 (Elastic Compute Cloud)," 2012.
http://aws.amazon.com/ec2/ [Online; accessed 12-12-2013].

[40] IBM, "IBM Smart Cloud," 2012.
http://www.ibm.com/cloud-computing/us/en/ [Online; accessed 12-12-2013].

[41] Hewlett-Packard Development Company, L.P., "HP Cloud," 2012.
https://www.hpcloud.com/ [Online; accessed 12-12-2013].

[42] AppFog, Inc. , " AppFog ," 2013.
https://www.appfog.com/ [Online; accessed 12-12-2013].

[43] Apprenda Inc. , " Apprenda ," 2013.
http://apprenda.com/ [Online; accessed 12-12-2013].

[44] CloudBees Inc. , " CloudBees ," 2013.
http://www.cloudbees.com/ [Online; accessed 12-12-2013].

[45] Cloudera, Inc. , " cloudera ," 2013.
http://www.cloudera.com [Online; accessed 12-12-2013].

[46] Engine Yard Inc., " Engine Yard Cloud ," 2010.
http://www.engineyard.com/ [Online; accessed 12-12-2013].

[47] Heroku Inc. , " Heroku ," 2013.
https://www.heroku.com [Online; accessed 12-12-2013].

[48] RightScale Inc., " RightScale ," 2010.
http://www.rightscale.com/ [Online; accessed 12-12-2013].

[49] Cloudscaling Inc. , " Cloudscaling ," 2013.
http://cloudscaling.com/ [Online; accessed 12-12-2013].

[50] GoGrid Cloud Hosting , " GoGrid ," 2010. http://www.gogrid.com/ [Online; accessed 12-12-2013].

[51] Joyent, Inc. , " Joyent ," 2013. http://www.joyent.com/ [Online; accessed 12-12-2013].

[52] Rackspace, US Inc. , " Rackspace ," 2013. http://www.rackspace.com/ [Online; accessed 12-12-2013].

[53] Savvis Inc., " Savvis ," 2010. http://www.savvis.net/ [Online; accessed 12-12-2013].

[54] CenturyLink Inc. , " CenturyLink ," 2013. http://www.centurylinkcloud.com/ [Online; accessed 12-12-2013].

[55] Citrix Systems, Inc. , " Citrix ," 2013. http://www.citrix.com/ [Online; accessed 12-12-2013].

[56] B. Wilder, *Cloud Architecture Patterns.* Oreilly and Associate Series, Oreilly & Associates Incorporated, 2012.

[57] D. Barry, *Web Services, Service-Oriented Architectures, and Cloud Computing: The Savvy Manager's Guide.* The Savvy Manager's Guides, Elsevier Science, 2012.

[58] J. Rhoton and R. Haukioja, *Cloud Computing Architected.* Recursive, Limited, 2011.

[59] J. Rosenberg and A. Mateos, *The Cloud at Your Service: The When, How, and Why of Enterprise Cloud Computing.* Manning Pubs Co Series, Manning Publications Company, 2010.

[60] T. Erl, R. Puttini, and Z. Mahmood, *Cloud Computing: Concepts, Technology & Architecture.* The Prentice Hall Service Technology Series from Thomas Erl, Pearson Education, 2013.

[61] B. Sosinsky, *Cloud Computing Bible.* Wiley, 2010.

[62] I. Brandic, "Towards self-manageable cloud services," in *Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International*, vol. 2, pp. 128 –133, july 2009.

[63] E. Keller and H. Ludwig, "The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services," *Journal of Network and Systems Management*, vol. 11, p. 2003, 2003.

[64] O. Wäldrich, P. Wieder, and W. Ziegler, "A meta-scheduling service for co-allocating arbitrary types of resources," in *Proceedings of the 6th international conference on Parallel Processing and Applied Mathematics*, PPAM'05, (Berlin, Heidelberg), pp. 782–791, Springer-Verlag, 2006.

[65] D. D. Lamanna, J. Skene, and W. Emmerich, "SLAng: A Language for Defining Service Level Agreements," in *Proceedings of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems*, FTDCS '03, (Washington, DC, USA), pp. 100–, IEEE Computer Society, 2003.

[66] V. Tosic, K. Patel, and B. Pagurek, "WSOL - Web Service Offerings Language," in *Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web*, CAiSE '02/ WES '02, (London, UK, UK), pp. 57–67, Springer-Verlag, 2002.

[67] L. Zhao, Y. Ren, M. Li, and K. Sakurai, "Flexible service selection with user-specific qos support in service-oriented architecture," *Journal of Network and Computer Applications*, vol. 35, no. 3, pp. 962 – 973, 2012.

[68] L. Qi, W. Dou, X. Zhang, and J. Chen, "A qos-aware composition method supporting cross-platform service invocation in cloud environment," *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1316 – 1329, 2012.

[69] R. Tolosana-Calasanz, J. Ángel Bañares, C. Pham, and O. F. Rana, "Enforcing qos in scientific workflow systems enacted over

cloud infrastructures," *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1300 – 1315, 2012.

[70] L. Wu, S. K. Garg, and R. Buyya, "SLA-based admission control for a software-as-a-service provider in cloud computing environments," *Journal of Computer and System Sciences*, vol. 78, no. 5, pp. 1280 – 1299, 2012.

[71] S. Gogouvitis, K. Konstanteli, S. Waldschmidt, G. Kousiouris, G. Katsaros, A. Menychtas, D. Kyriazis, and T. Varvarigou, "Workflow management for soft real-time interactive applications in virtualized environments," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 193 – 209, 2012.

[72] H. Ludwig, A. Keller, A. Dan, and R. King, "A service level agreement language for dynamic electronic services," in *Advanced Issues of E-Commerce and Web-Based Information Systems, 2002. (WECWIS 2002). Proceedings. Fourth IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS 2002)*, pp. 25–32, 2002.

[73] A. Kertesz, G. Kecskemeti, and I. Brandic, "An SLA-based resource virtualization approach for on-demand service provision," in *Proceedings of the 3rd international workshop on Virtualization technologies in distributed computing*, VTDC '09, (New York, NY, USA), pp. 27–34, ACM, 2009.

[74] V. C. Emeakaroha, I. Brandic, M. Maurer, and S. Dustdar, "Low level Metrics to High level SLAs - LoM2HiS framework: Bridging the gap between monitored metrics and SLA parameters in cloud environments," in *High Performance Computing and Simulation (HPCS), 2010 International Conference on*, pp. 48 –54, 28 2010-july 2 2010.

[75] R. Buyya, S. K. Garg, and R. N. Calheiros, "SLA-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions," in *Proceedings of the 2011 International*

Conference on Cloud and Service Computing, CSC '11, (Washington, DC, USA), pp. 1–10, IEEE Computer Society, 2011.

[76] R. N. Calheiros, C. Vecchiola, D. Karunamoorthy, and R. Buyya, "The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid Clouds ," *Future Generation Computer Systems*, vol. 28, no. 6, p. 861(10), 2012-06-01.

[77] The Reservoir Consortium, "Resources and Services Virtualization without Barriers," 2012.
http://www.reservoir-fp7.eu/ [Online; accessed 12-12-2013].

[78] The BonFIRE Consortium, "Building service test beds on FIRE," 2012.

[79] The OPTIMIS Consortium, "OPTIMIS: Optimized Infrastructure Services," 2012.
http://www.optimis-project.eu/ [Online; accessed 12-12-2013].

[80] The 4CaaSt Consortium, "Building the PaaS Cloud of the Future," 2012.
http://4caast.morfeo-project.org/ [Online; accessed 12-12-2013].

[81] The SLA@SOI consortium, "SLA@SOI Empowering the service economy with SLA-aware infrastructures," 2011.

[82] The Cloud-TM Consortium, "Cloud-TM: A novel programming paradigm for cloud computing," 2012.
http://www.cloudtm.eu/ [Online; accessed 12-12-2013].

[83] The Cloudscale Consortium, "Scalability Management for Cloud Computing," 2012.
http://www.cloudscale-project.eu/ [Online; accessed 12-12-2013].

[84] The PaaSage Consortium, "PaaSage: Model Based Cloud Platform Upperware," 2012.
http://www.paasage.eu/ [Online; accessed 12-12-2013].

[85] The Contrail Consortium, "Open computing infrastructure for elastic services," 2012.
http://contrail-project.eu/ [Online; accessed 12-12-2013].

[86] W. Barth, *Nagios: System and Network Monitoring*. San Francisco, CA, USA: No Starch Press, 2nd ed., 2008.

[87] M. Massie, "The ganglia distributed monitoring system: design, implementation, and experience," *Parallel Computing*, vol. 30, pp. 817–840, July 2004.

[88] Zenoss, Inc., "Zenoss."
http://community.zenoss.org/index.jspa [Online; accessed 12-12-2013].

[89] Zabbix SIA., "Zabbix."
http://www.zabbix.com [Online; accessed 12-12-2013].

[90] FireScope, Inc., "FireScope."
http://www.firescope.com/ [Online; accessed 12-12-2013].

[91] Edgewall Software, "Munin."
http://munin-monitoring.org/ [Online; accessed 12-12-2013].

[92] Florian octo Forster, "Collectd."
http://collectd.org/ [Online; accessed 12-12-2013].

[93] F. Doelitzscher, M. Held, C. Reich, and A. Sulistio, "Viteraas: Virtual cluster as a service," in *Cloud Computing Technology and Science (CloudCom), 2011 IEEE Third International Conference on*, pp. 652 –657, 29 2011-dec. 1 2011.

[94] M. Maurer, I. Brandic, and R. Sakellariou, "Simulating Autonomic SLA Enactment in Clouds Using Case Based Reasoning," in *ServiceWave*, pp. 25–36, 2010.

[95] G. Katsaros, G. Kousiouris, S. V. Gogouvitis, D. Kyriazis, A. Menychtas, and T. Varvarigou, "A Self-adaptive hierarchical monitoring mechanism for Clouds," *J. Syst. Softw.*, vol. 85, pp. 1029–1041, May 2012.

[96] The SLA@SOI Consortium, "Deliverable D.A3a SLA-aware Service Management."

[97] J. Sauvé, F. Marques, A. Moura, M. Sampaio, J. Jornada, and E. Radziuk, "SLA Design from a Business Perspective," in *In Proceedings of DSOM 2005*, Springer, 2005.

[98] A. Pichot, O. Wäldrich, W. Ziegler, and P. Wieder, "Dynamic sla negotiation based on ws-agreement," in *WEBIST (1)*, pp. 38–45, 2008.

[99] C. Herssens, S. Faulkner, and I. J. Jureta, "Context-Driven Autonomic Adaptation of SLA," in *Proceedings of the 6th International Conference on Service-Oriented Computing*, ICSOC '08, (Berlin, Heidelberg), pp. 362–377, Springer-Verlag, 2008.

[100] A. Garcia Garcia, C. De Alfonso Laguna, and V. Hernandez Garcia, "Design of a Platform of Virtual Service Containers for Service Oriented Cloud Computing," in *Cracow Grid Workshop 2009*, pp. 20–27, Academic Computer Centre CYFRONET, 2009.

[101] M. Jeckle, D. M. Doolin, P. McMahan, P. McMahan, R. Wade, B. Toy, and J. Dongarra, " Linpack Java implementation ," 2004. http://www.jeckle.de/freeStuff/jLinpack/ [Online; accessed 12-12-2013].

[102] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the third annual ACM symposium on Theory of computing*, STOC '71, (New York, NY, USA), pp. 151–158, ACM, 1971.

[103] R. M. Karp, "Reducibility Among Combinatorial Problems," in *Complexity of Computer Computations* (R. E. Miller and J. W. Thatcher, eds.), pp. 85–103, Plenum Press, 1972.

[104] R. T. Fielding, *Architectural styles and the design of network-based software architectures.* PhD thesis, University of California, Irvine, 2000. Chair-Taylor, Richard N.

[105] R. Kübert, G. Katsaros, and T. Wang, "A RESTful implementation of the WS-agreement specification," in *Proceedings of the Second International Workshop on RESTful Design*, WS-REST '11, (New York, NY, USA), pp. 67–72, ACM, 2011.

[106] The HSQLDB Development Group , "HyperSQL DataBase," 2010.
http://hsqldb.org/ [Online; accessed 12-12-2013].

[107] M. Maurer, I. Brandic, and R. Sakellariou, "Adaptive resource configuration for cloud infrastructure management," *Future Generation Computer Systems*, vol. 29, no. 2, pp. 472 – 487, 2013.

[108] M. Maurer, I. Brandic, V. Emeakaroha, and S. Dustdar, "Towards Knowledge Management in Self-Adaptable Clouds," in *Services (SERVICES-1), 2010 6th World Congress on*, pp. 527 –534, july 2010.

[109] The Venus-C Consortium, "D3.10 - Future Sustainability Strategies," 2012.
http://www.venus-c.eu/deliverables_year2/VENUS-C_D3.10.pdf [Online; accessed 12-12-2013].

[110] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 5th ed., 2011.