IMT School for Advanced Studies, Lucca

Lucca, Italy

Improving Service Quality in Cloud Computing: from Definition to Deployment

PhD Program in Computer Science

XXX Cycle

By

Vincenzo Scoca

2017

Contents

List of Figures

List of Tables

Declaration

Most of the material in this thesis has already been published in scientific venues.

More in details, the work presented in Chap. 3 is coauthored with Rafael Brundo Uriarte, Rocco De Nicola and Francesco Tiezzi and is currently under review for the *Future Generation Computer System Journal* and is based on the already published works [70, 71] of Rafael Brundo Uriarte, Rocco De Nicola and Francesco Tiezzi from IMT School for Advanced Studies, Lucca.

Chap. 4, instead, is based on [57], coauthored with Rafael Brundo Uriarte and Rocco De Nicola.

Finally, the work in Chap. 5 is inspired by a collaboration with Atakan Aral and Ivona Brandic (Technical University of Vienna) and is based on [56], coauthored with Atakan Aral, Ivona Brandic, Rocco De Nicola and Rafael Brundo Uriarte.

Publications

1. Rafael Brundo Uriarte, Rocco De Nicola, Vincenzo Scoca, and Francesco Tiezzi, "Defining QoS in Dynamic Environments", in *Future generation computer systems*. Under review
2. Vincenzo Scoca, Atakan Aral, Ivona Brandic, Rocco De Nicola, and Rafael Brundo Uriarte, "Scheduling Latency-Sensitive Applications in Edge Computing", in *Proceedings of the 8th International Conference on Cloud Computing and Services*, pages 158-168, 2017
3. Vincenzo Scoca, Rafael Brundo Uriarte, and Rocco De Nicola,"Smart contract negotiation in cloud computing", in *Cloud Computing (CLOUD), 2017 IEEE 10th International Conference*, pages 592-599, 2017
4. Michelangelo Diligenti, Marco Gori, and Vincenzo Scoca, "Learning efficiently in semantic based regularization", in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, vol. 4, pages 33-46, 2016

Abstract

Service quality is crucial in all stages of the Cloud service life cycle, from service acquisition, where Cloud consumers and providers negotiate for a mutual agreement, to service execution, where service management is driven by the agreed requirements.

Much work has been devoted to specification and enforcement of service quality terms in the Cluster, Grid and Cloud domains. However, the dynamism present in Cloud services is ignored. We propose a theoretical and practical framework which addresses the first phases of the service life cycle: (i) the definition of service provision; (ii) the negotiation of offers/requests expressed and (iii) the service deployment, mainly focused on latency-sensitive applications.

We introduce SLAC, a specification language for the definition of service requirements, the so-called service level agreements (SLAs), which allows us to define conditions and actions that can automatically modify those terms at runtime. Experimental results show that the use of SLAC can drastically reduce the service violations and penalties to the advantages of providers and consumers.

Then, we define a novel matchmaking and negotiation framework, which evaluates the compatibility of SLAC requests/offers, and provides the modifications necessary to reach an agreement. Experiments demonstrate the effectiveness of our proposal.

We also introduce a new scheduling algorithm for latency-sensitive services, in a Cloud/Edge computing scenario, which takes into account not only the service requirements but also network latency, bandwidth and computing capabilities. Again, experimental results confirm the advantages of this new approach over existing solutions.

Chapter 1

Introduction

Cloud computing is a paradigm that implements the concept of utility computing and defines computing resources (hardware and software) as *services* that can be delivered as traditional utilities, such as water and electricity [10].

In this context the *service quality* is expressed by a set of non-functional properties, the so called Quality of Service (QoS) metrics, like availability, security and reliability, that specify the guarantees to be respected in terms of performance, security and other non-functional aspects.

The impact of QoS on the service life cycle is crucial in all of its stages. Indeed, at acquisition time, given a set of functionally equivalent services, a consumer will select the one offering non-functional requirements, such as performance, reliability and security, that mostly fits his/her needs. Besides, at execution time service, configuration and resource management are driven by the specified QoS terms. More precisely, service execution is continuously monitored to assess the fulfilment of its quality terms, in order to apply the appropriate corrective actions to avoid the violations of the agreed terms.

Therefore, given the relevance of the service quality in determining the success of a Cloud service, a large body of work has already been carried out for the specification and enforcement of service quality terms, as described in Sec. 6. However, they do not consider one of the main aspects of Cloud: the *dynamism* that characterizes cloud services, where users and providers' requirements may change at run time.

Cloud paradigm is inherently dynamic from both consumer and provider perspectives. From the provider's standpoint, new resources are added and removed on-the-fly, whilst service requests and prices vary over time as the pay-per-use model is employed. From the consumer's perspective, instead, the requirements vary considerable as Cloud are used to, for example, outsource internal services or to complement the computing capacity through a hybrid Cloud.

Consequently, such dynamism directly affects the service quality and then suitable solutions are needed for the proper specification of the requirements.

Currently, consumers and providers define their service quality needs and their offers by means of Service Level Agreements (SLAs) specifying (i) the scope and the performance expectations of the service; (ii) the obligations for both consumer and provider; (iii) the actions to be applied if the expectations are not met; (iv) the bounding liabilities [7]. However, existing SLA languages, allows the definition of only *static* agreements that remain valid for the whole service execution, without any possibility of automatically changing the terms of agreements when specific conditions are met.

In the light of these challenges, we defined SLAC (*Service Level Agreements for Clouds*) a SLA definition language based on [40, 70, 71] specifically devised for Cloud which encompasses a mechanism for the automatic modifications of SLA terms. Essentially, SLAC allows the specification of actions which can automatically change the set of both functional and non-functional requirements, when specific conditions are met. Therefore, both consumers and providers can specify dynamic requirements fixing the initial values of the terms as well as the conditions to modify them and the related new term values. Our language offers, then, the possibility to specify dynamic requirements to support the intrinsically dynamic nature of Cloud services.

An example of a SLA defined using SLAC is reported Fig. 5. In this example, the initial state of the contract is the "Base" quality, with 2 Virtual Machines (VMs) and no requirement about Response Time (RT). During service execution, the consumer may request the upgrade to "Diamond" quality (more VMs and the stipulation of the maximum RT). When in the "Diamond" state, the provider might need to downgrade the service to "Ruby" (for example, in case of overbooking of resources) and from "Ruby" back to "Diamond".

However, the adoption of this new specification language does introduce a

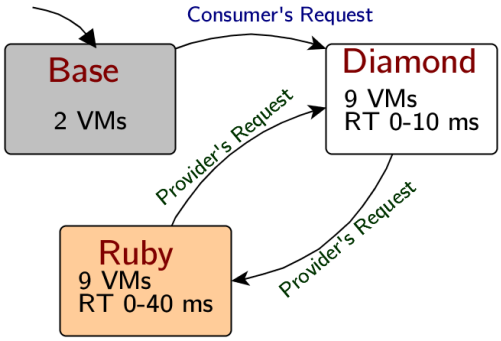

Figure 1: Example of a smart contract in the Cloud domain.

valid support to cope with Cloud dynamism but carries with it new challenges related to the negotiation of the SLAs specified with the new formalism and to the service management.

Indeed, the adoption of traditional bilateral negotiation methods for negotiating over dynamic contracts would require far too many interactions with proposals and counter-proposals. Also the widely used *matchmaking* alternative relying on brokers that selects the offers published by Cloud providers most compatible with consumer's requests, cannot be easily adapted to "dynamic" contract since they need to consider an enormous number of alternatives and nevertheless might not lead to exact matches. Of course, the absence of exact matching between offers and requests does not necessarily mean that the parties are not willing to give in something to reach an agreement. The example described in Fig. 2 is possibly one of such cases. In this case, the only change not covered by the provider's offer is the possibility to increase the response time (from "Gold" to "Silver"), which in most cases is beneficial for the provider. Adding this possibility to the service offer would improve their compatibility. Also other solutions could satisfy the involved parties, for example, finding a compromise between the values requested and those offered (e.g., a state with 4 VMS and 25 ms) or removing the "Silver" state from the consumer's request.

To deal with these issues, we have developed a novel negotiation framework for SLAC contracts, responsible of matching offers and requests and of facilitating an agreement between the parties in case no immediate matching is possible.

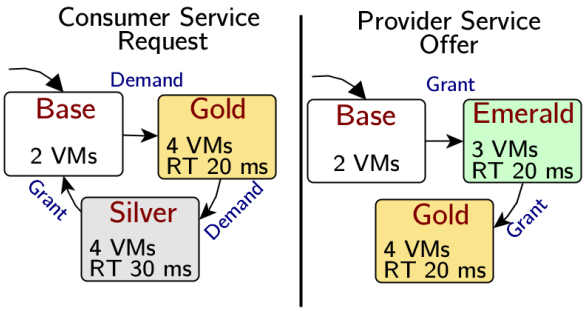

Figure 2: Example of SLA offer/request proposal using SLAC.

Our solution provides a simple formal language for modelling the behaviour of the contracts in terms of transitions and states allowing the representation of these models as automata with labels associated to nodes and arcs, as shown in Fig. 2. These automata are feed to our software negotiation framework which assesses the compatibility of offer/request and suggests the changes necessary for reaching an agreement.

Afterwards, while analysing scenarios characterized by high dynamism and looking for a good case study for the application of dynamic SLA, we noticed that there are still open gaps in the scheduling of latency sensitive services. We focused then, on the development of a suitable approach for the initial deployment of service instances. We decided to address latency sensitive scenarios because they are highly dynamic and because we could not find any work considering the scheduling problem in Edge platforms, that represent the most suitable deployment solution for this type of services.

The shift from Cloud to Edge computing has been determined by features of the Edge platform from which latency-sensitive services can benefit. Indeed, the currently adopted Cloud-based deployment solutions are not fully suitable to deal with strict latency requirements while the Edge platform represents a promising alternative, since it offers a set of enabling technologies that move both computation and storage capabilities (i.e., micro cloud data centres) to the edge of the network, and considerably reduces users' end-to-end latency. Moreover, the Edge infrastructure can be seen as a natural a generalization of the Cloud and, thus, it does not requires any change in the specification and ne-

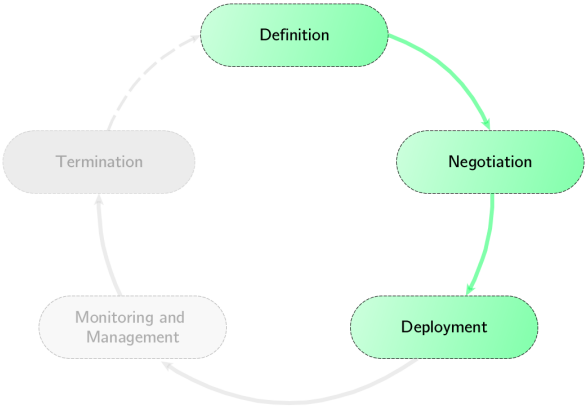

Figure 3: Stages of the service life cycle to which the thesis contributes.

gotiation phases, only resource and service management techniques need to be changed.

However, to effectively exploit the advantages of the Edge infrastructure, service instances need to be scheduled on the node which better fits service requirements, taking into account contextual knowledge, i.e., user, application and network information [76]. Indeed, defining an optimal placement allows not only to maximize user experience by reducing end-to-end latency, but it allows also to optimize network traffic by reducing bandwidth consumption and related costs, and improves energy efficiency by reducing the need to offload tasks to the Cloud which requires more energy than micro data centers.

We propose then, a novel score-based scheduling framework specifically designed for latency-sensitive services in Edge. The proposed technique is latency, bandwidth, and resource aware. It schedules each service instance on the VM type whose computing and network capabilities can optimize service response time experienced by end users. First, eligibility of available VM types on edge nodes for hosting given services is evaluated based on VM network and resource capabilities, and then such services are scheduled in the most suitable VMs according to eligibility scores, guaranteeing optimal service quality.

The different stages of a service life cycle in the context of service quality management in Cloud computing, to which this thesis is contributing, are

reported in Fig. 3. We devised a framework that considers the first phases of services' life cycle by (i) defining SLAC, a new specification language for the definition of dynamic agreements (Chap. 3); (ii) developing a negotiation framework suitable for the negotiation of the SLA defined with SLAC (Chap. 4) and (iii) devising a novel scheduling framework for the initial deployment of service instances in Edge that is specifically tailored for latency-sensitive services (Chap. 5).

In the rest of this chapter, we present the main research questions that have been considered in this thesis, and have been a guideline for our work.

Research Question 1

How to properly describe service quality terms in Cloud Computing?

The definition of service level agreements specifies the requirements to be respected during the service provisioning. However, Cloud services are inherently characterized by a dynamism which makes difficult to properly define these requirements with current specification languages. This dynamism is related to consumer and provider needs that continuously change during service execution. Defining static quality terms valid for the whole service life cycle is not the most appropriate approach; a specification language is needed that allows the autonomous modification of the defined requirements, when specific conditions are met. We address this research question in Chap. 3.

Research Question 2

How to negotiate SLAs for Cloud Computing with the new dynamic language?

The adoption of a specification language that allows the definition of service requirements that can change at run time without the need of renegotiation, introduces new challenges in the negotiation of initial offer/request, since multiple SLA configurations need to be considered. The challenges characterizing this new specification language and an innovative framework for the negotiation of contracts specified with this new formalism are addressed in Chap. 4.

Research Question 3

How to enhance specific service quality
in distributed environments?

Edge-based solutions have been proposed for dealing with highly dynamic computational requirements of latency sensitive application, but they pose new challenges for services and resources management. In this context, solutions are missing for services scheduling; in Chap. 5 we introduce a new framework for the initial deployment of latency-sensitive service instances in Edge-based architectures.

Chapter 2

Quality of Service in Cloud Computing: Background

In this chapter we provide an overview of the basic concepts related to the main topics of the thesis. The overall contributions of this chapter are:

- A comprehensive description of the main features of Cloud and Edge Computing, which represent the background domains of the topics addressed in this thesis.

- An overview of service quality in and its impact on the service life cycle phases, which provides a general description of the scenario used to illustrate the works presented in the other chapters.

- A preliminary analysis of smart contracts and of their applicability for Service Level Agreements (SLAs) specification in Cloud. These topics will be further developed in Chap. 3.

- A discussion of the approaches currently adopted for the autonomic negotiation of SLAs and for service scheduling in Cloud. This discussion will be instrumental to introduce the basic notion for the results of Chap. 4 and Chap. 5.

2.1 Cloud Computing

In the last half century research in the Information and Communication Technology (ICT) area has guranteed significant advances, and nowadays there is an increasingly perceived vision that *computing* will one day be the 5th utility, after water, electricity, gas and telephony [10]. This vision of computing as utility can be found in the advances achieved in the development of the Internet infrastructure that allowed a worldwide aggregation of computer networks giving the user the perception of utilizing an endless amount of distributed computing resources, without knowing where they are hosted and how they are delivered.

Several computing paradigms have been developed and adopted over the years, with the aim to offer computing systems that accurately put this new vision at work: the most important ones being *Grid* and *Cloud* computing.

Grid Computing was developed in the mid 1990s to allow consumers to obtain computing power on demand [30]. A more formal definition of Grids is given in [10]:

> A Grid is a type of parallel and distributed system that enables the sharing, selection, and aggregation of geographically distributed 'autonomous' resources dynamically at run time depending on their availability, capability, performance, cost, and users' quality-of-service requirements.

On the other hand, in [45], Cloud has been defined as:

> 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.

Cloud Computing refers mostly to a new operation model that leverages on existing technologies, such as Grid Computing, Virtualization and Autonomic Computing, to meet the technological and economic requirements of today's demand from IT companies [83]. The main reasons making Cloud paradigm more and more attractive for business in the IT area are reported by Zahong et al. in [83] and listed below:

No up-front investment. Cloud offers both hardware and software resources using the pay-as-you-go model, allowing consumers to rent the needed resources according to their needs, without a high initial investment.

Low operating costs. Providers allocate resources to consumers according to their demand, without over-provisioning, which allows the providers to grant the resources only when actually needed and then to reduce the operating costs by taking advantage of the economy of scale (energy, hardware, maintenance).

High scalability. Cloud provides two modalities of resource scaling: horizontal and vertical. Horizontal scalability allows the addition or removal of resource instances (e.g., servers), whereas vertical scalability does not modifies the number of resources but changes the specification of existing ones (e.g., increasing or decreasing the power offered by each server).

Reducing business risks and maintenance expenses. In the scenario where Cloud consumers demand for hardware resources to a Cloud provider, they move some of the business risks and maintenance process to the provider.

All the benefits listed above arise from the intrinsic features of the Cloud computing paradigm. Indeed, in a Cloud environment, several services from different consumers can be hosted in a single data-center (*multi-tenancy*) and the performance and management issues are ruled by Service Level Agreements (SLAs) subscribed by both the Cloud provider and the service consumers. To support multi-tenancy, Cloud providers offer a pool of resources that can be dynamically assigned to multiple resource consumers (*shared resource pooling*) allowing service providers to acquire resources according to the current demand (*dynamic resource providing*) based on a pay-per-use pricing model (*uility-based pricing*).

Business models

In this context the term resource refers to both hardware and software components that are provided to Cloud consumer *as-services*. Therefore, the adoption

of this service-driven operating model brought to the development of three different business model, as also reported in the NIST definition of Cloud Computing [45]:

Software as a Service (SaaS). Provider's applications represent the service delivered to final users. In this context, users can directly use provider's application by accessing it with different type of devices, without any management responsibility. Indeed, only the provider is in charge of managing and controlling the underlying infrastructure.

Platform as a Service (PaaS). Cloud providers offer complete software development platforms (i.e., operating systems and application development tools and frameworks), where consumers can deploy their own applications without any responsibility related to the management of the underlying software layers.

Infrastructure as a Service (IaaS). Processing, storage, network, and other fundamental computing resources are provided to consumers who can then deploy and run any arbitrary software on the top of them. In this model, consumers are responsible of the software infrastructure management, i.e., they are in charge of installing and configuring the requested resources (e.g., VMs).

Deployment Models

Different deployment models can be adopted in Cloud. The main are:

Private Cloud. A single organization uses the whole Cloud infrastructure, that is managed by the organization itself or by a third party. Although this model offers the highest degree of control of the infrastructure, it is the most similar to traditional server farms, limiting the benefits introduced by the Cloud paradigm.

Public Cloud. A provider offers its resources as services to the general public. The main advantages of this model are represented by no up-front costs on infrastructure for the consumer and by the shift of risks to infrastructure providers at the cost of reduced control.

Hybrid Cloud. This model combines the previous models and aims at addressing the limitations characterizing the previous two models.

2.2 Edge Computing

Recently, due to the advances of connectivity infrastructure and the proliferation of Internet of Things (IoTs), new type of services have been introduced that are characterized by very strict latency requirements and by high bandwidth demand along with the need of high computing power for data processing tasks.

Therefore, even though Cloud-based deployment solution still represent the optimal solution for the processing of large amount of data, the large geographical distribution of Cloud data-centers cannot guarantee the very strict end-to-end latency required by these services.

Indeed, IoT services are usually related to the processing of data obtained through sensors (e.g., camera,) for pattern matching and data analysis to take timely decisions (i.e., on the order of very few milliseconds) [19]. Thus, the network latency characterizing Cloud networks cannot cope with these strict requirements and the high bandwidth consumption determined by the large amount of data generated by the sensors may turn out to be a bottleneck.

Consequently, innovative distributed computing solutions have been developed allowing the computation tasks to be executed at the Edge of the network, in close proximity of the end users.

This novel computing technology has been named *Edge Computing*. This term is also often interchanged with *Fog Computing*, which is mainly related to an actual architectural design of an Edge network [24, 76].

The main idea underneath this new paradigm is to move the Cloud technologies in close proximity of end users [14] for enabling ubiquitous access to a shared network of computing resources and facilitate the deployment of distributed, latency-aware applications and services [34].

An *edge node* is essentially any type of computing resources and routing devices in the network along the path from the data source and the Cloud data-centers [61]. In the literature, this set of resources have been grouped in five different types, namely servers, networking devices, cloudlets, base stations, vehicles as reported in [43] and listed below:

Servers. Edge servers are usually micro data-centers or nano servers geographically distributed and deployed at highly frequented places (i.e., bus terminals, shopping centres, etc.). Similarly to Cloud data-centers, edge servers exploit virtualization technologies to provide storage, computing and networking facilities.

Networking devices. Networking devices, like gateway routers and switches, provide not only networking functionalities but are also equipped with some processing power and memory to be used for small data processing tasks.

Cloudlets. Cloudlets are considered as micro-cloud, offering higher amount of resources compared with other types of devices but acting as centralized components. Therefore, even if they are deployed at the edge of the network they might incur in the same limitations of Cloud Computing.

Base Stations. Traditional base stations are equipped with processing and storing capabilities. However, the main advantages offered by these devices is the single-hop connection they can provide with the main network backbone.

Vehicles. The high number of vehicles currently present on the streets can be used to offer a widely distributed and highly scalable Edge environment. Indeed, vehicle are now equipped with exploitable processing and storing capabilities.

Currently three different Edge architectures have been devised, namely *Fog computing*, *Cloudlet* and *Mobile Edge Computing (MEC)*

In the fog architecture, the Edge part is composed of network devices (i.e., fog nodes) which complement a Cloud infrastructure making computation and storage functionalities possible anywhere along the path from end users to the Cloud data-centers [76]. This platform is most suitable for Internet of Things use cases, because, thanks to the proximity of all fog devices involved, it can guarantees the very strict latency requirements required by those services.

The cloudlet platform, instead, is characterized by a trusted cluster of computers well connected to the internet with resources available for use by nearby

devices. Cloudlet resources are usually deployed at WiFi access points or network base stations offered over a single-hop access with high bandwidth, guaranteeing then low latency for the deployed applications[24]. This architecture is mostly used for the deployment of applications which requires not only strict latency requirements, but also high processing power that cannot be provided by the network devices present in the Fog platform.

The MEC architecture brings computational and storage capacities to the edge of the network within the Radio Access Network to reduce the service end-to-end latency and to improve context awareness [24]. MEC nodes are servers that are usually co-located with Radio Network Controllers or with macro base-stations. This solution has been mainly devised to allow the offloading of heavy processing tasks, that had to be executed on mobile devices, on more powerful resources, improving the life-time of the device battery. Indeed, user devices can send task computation on a MEC node that either directly process it or forward it to the remote data centers.

2.3 Quality of Service in Cloud

Both Cloud consumers and providers expect that applications and services deployed on Cloud platforms guarantee high quality services that offer good level of user experience. In fact, a good service appreciated by users increases also providers' trustworthiness and thus their revenues.

In this scenario we define *service quality* as the degree to which a set of service characteristics meet the agreed requirements. Specifically, it is as a combination of non-functional attributes describing service characteristics that are considered relevant for the service-client interaction.

Moreover, these service quality characteristics can be aggregated in two groups, namely Quality of Experience (QoE) of Quality of Service (QoS) [41]. The ones in the first group express a subjective definition of service quality through subjective attributes such as usability, that allow users to express their personal perception about the quality experienced. The ones in the second group, instead, express service quality by means of objective measurable metrics like, e.g., response time and availability. This set of QoS terms, also known as *Service Level Objects* (SLOs), is then used for the formal description of the quality

requirements to be guaranteed during service provision, defining the so called *Service Level Agreements* (SLAs)

Here we provide a description of the most common service quality attributes [7].

Availability. Represents the ability of an IT service to perform its agreed function when required. It can be expressed as:

$$\text{Availability} = \frac{\text{Agreed Service Time} - \text{Outage Downtime}}{\text{Agreed Service Time}} \tag{2.1}$$

where the Agreed Service Time is the period during the measurement window that the system be running, whereas the Outage Downtime represent the total time, still in the measurement window, where the system was not available.

Latency. The latency indicator refers to the elapsed time between the sending of a request and receipt of the reply. Usually it is measured as best thought of as a statistical distribution rather than a single value that can crisply be measured. It can be specified as a distribution requirements via two points like the maximum acceptable latency at both the 90th percentile (slowest 1 in 10 transactions) and the 99.999th percentile (slowest 1 in 100,000 transactions)

Reliability. Given a logically, syntactically and semantically correct service request, service reliability requirements specify the probability to obtain the correct response within the maximum acceptable service latency. Usually it is measured as the percentage of defective (or failed) operations per million attempts, or DPM, that is:

$$99.9\% \text{ service reliability} = 1000 \text{ defective operations per million}$$

Accessibility. This indicator is related to individual users and defines the probability than he/she can successfully acquire service on demand. The metric used to measure service accessibility is maximum DPM,

Retainability. This metric is related to session-oriented applications and denotes the probability that an ongoing service session will continue to provide the service under certain conditions for a given time. For example,

it represents the probability that a streaming movie plays to the end with no perceptible visual or audible impairments. Metric values may be quantified as application sessions per million (DPM) that were prematurely terminated or as experienced (unacceptable) service impairments.

Throughput. This requirement specifies the minimum number of transitions successfully processed in a given unit of time.

2.4 Service Life Cycle and Quality

As described in Sec. 2.3 service consumers and providers specify the service quality to be guaranteed throughout the whole service provision by means of a set of non-functional attributes that are relevant for the service-client interaction.

These requirements, the so called SLOs, are then formalized in a contract, the *Service Level Agreement* (SLA), that specifies the quality guarantees to be respected and enforced during the service provision as well as the actions to be taken when the guarantees are violated.

More in detail, a SLA is the contract legally binding service consumers and providers and is concerned with:

- The *parties* involved in the agreements, specifying their roles and responsibilities.
- The *functional* aspects of the provided service.
- The set of *obligations and guarantees* imposed on each party related to the service provision.
- The *service quality attributes* (i.e., QoS terms) specifying the non-functional requirements, focused on quality aspects (e.g., service response time) to be guaranteed during the service provision.
- The *timing and condition* for the validity of the contract.

This contract is also characterized by a specific life cycle which is tightly coupled with the service life cycle, as depicted in Fig. 4. Here we describe each phase of the SLA life cycle and how it is related to the service one.

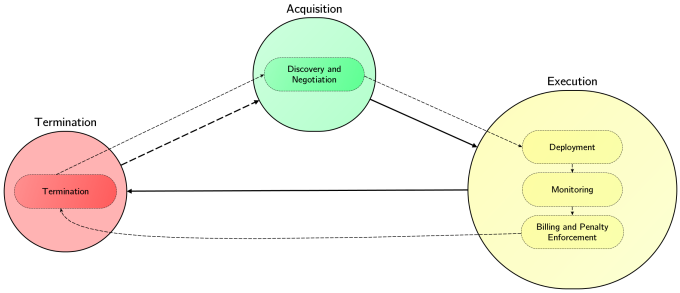

Figure 4: SLA and service life cycle.

The first phase of the SLA life cycle is the SLA discovery and negotiation, which characterizes the service acquisition stage. Consumers and providers define a SLA template where they specify the service requirements they request/offer. After the formulation is completed, Cloud providers publish their offers in specific repositories, and, then, Cloud consumers can retrieve the offer that best fits their requirements by means of brokers or discovery services. However, it is very unlikely that offer/request are already 100% compatible, therefore a negotiation step between the parties is needed to reach a final agreement. In case the outcome of this step is positive, a final SLA is produced and signed by the parties which is then validated and deployed. It has to be said that agreement negotiation is not always available. In many cases the provider is not willing to negotiate and the consumer need to choose the proposal that mostly fit his needs.

The next phase of the SLA life cycle is the SLA deployment, which defines the first step of the service execution. In this phase each SLA signatory party extract from the final agreement the appropriate configuration information needed to define the proper role and duties. Therefore, they can configure the service implementation accordingly.

As next step, the service is actually executed and the SLA is monitored. By SLA monitoring we mean the assessment of the SLOs defined within the SLA, by means of measurement techniques which allow to gather real-time information about the service status to be compared with the SLOs defined.

This is a constant process and plays an essential role in the SLA life cycle since all the violations that may occur during the service execution, need to be

forwarded to the party in charge for the billing and for penalty enforcement.

In this phase of the SLA life cycle the penalties defined in the SLA are enforced and corrective actions are executed by the service provider. If the violation is not critical, service management actions are carried out, otherwise the provider can ask the consumer to renegotiate the SLA terms to reduce the risk of violating the requirements again. Together with penalty enforcement also billing activities are periodically executed. These activities determines whether the SLA terms are met, the amount to be paid by the consumer and the overall penalties to be paid by the provider.

The last phase of service and SLA life cycles is the termination phase where service consumer and provider terminate service execution and the SLA is archived.

2.5 Smart Contracts

A first definition of smart contracts is provided by Nick Szabo in [65]:

> A smart contract is a set of promises, specified in digital form, including protocols within which the parties perform on these promises.

A more detailed one is given, instead, in the white paper issued by the Chamber of Digital Commerce [2] which defines a smart contract as a set of contractual terms and/or rule-based operations, embedded as code within a software, designed to carry out business logic. A protocol, in the form of an algorithm, constitutes a set of rules for how each party should process data in relation to a smart contract.

Essentially, this type of agreement represents the formalisation of an agreement, whose terms, such as payment, confidentiality and quality, are automatically enforced by relying on a previously agreed protocol without the need of third-party trusted intermediaries.

Such contracts may specify not only the required service and its quality but also the possible changes at run time of the terms of agreement through the definition of conditions and actions.

Several application fields have been reported in [2], ranging from digital identity to clinical trials but they are currently mostly used in the area of cryptocurrencies (e.g., Ethereum and Bitcoin).

In the area of Cloud computing, with their automatised enforcement and with solutions to deal with the dynamic nature of Cloud, the SLAs can be considered equivalent to smart contracts. However, current SLA definition languages provide only the possibility to specify static agreements that cannot be automatically changed at run time, In case adjustments are needed, a renegotiation process has to take place.

Further motivations and details about the application of smart contracts in Cloud and a possible use case scenario are provided in Chap. 3.

2.6 SLA Negotiation in Cloud

In the previous section we briefly discussed the advantages of smart contracts for the SLA specification. However, the adoption of this new formalism introduces also new challenges for the SLA negotiation process. Essentially, in this process, consumer and provider negotiate over service requirements, guarantee and obligations aiming at finding a definite mutual agreement.

Different types of negotiation can take place according to the number of parties involved [6]

One-to-one. This is the most common scenario; a consumer and a provider directly bargain for the acquisition of an item or a service.

One-to-many. In this context a consumer negotiates with several providers or businesses.

Many-to-one. This is a less common scenario; different consumers negotiate with single provider or company.

Many-to-many. Several consumers and providers are involved in the negotiation process. A typical example is provided by the negotiation between unions and employees where companies cooperate to reach an integrated agreement that saves their interests and unions bargain to get the best benefits from the employers they represent.

Moreover, each type of negotiation relies on three main components:

Negotiation objects. The negotiation objects are essentially the set of terms subject of negotiation.

Protocol. The rules specifying the negotiation process (i,e.,number of participants and their roles), the interaction among the negotiators and the legal actions that the participants can take depending on the state of the negotiation, are the main components of a negotiation protocol.

Decision models. Each party establishes a decision model to elaborate the negotiation moves to achieve their objectives.

However, when considering smart contracts, some of the negotiation components described before, namely objects and models, considerably change. Indeed, negotiation objects are not fixed, and the conditions and the modification actions specified by each negotiation party may change. Also, the dynamicity of the objects considered in a smart contract requires changes of the negotiation models, currently devised for bargaining over static terms.

A more detailed analysis of the limitations and of current negotiation solutions for the negotiation of smart contracts is presented in Sec. 6, and a new framework better suitable for this process is presented in Chap. 4.

2.7 Service Scheduling

Scheduling in Cloud Computing refers to the set of techniques and policies adopted for the autonomous management of services and resources to achieve a high performance computing and high system throughput [79].

More specifically, the scheduling process defines the order of servicing incoming services and the resources to be used for their execution, according to QoS terms, costs, revenues for Cloud providers, and users experience.

The three main steps that characterizing scheduling, reported in [59], are hereby described:

Resource discovering and filtering. A resource broker is in charge to discover resource availability and status information.

Resource selection. According to the resource information retrieved by the broker, the selection step assigns the available resources to all the services to be executed.

Task submission. Services are finally executed on the assigned resources.

Many approaches to resource scheduling have been followed. They range from very simple heuristic-based approaches, such as the Minimum Execution Time and Minimum Completion Time [46] to more advanced ones based on genetic algorithms and simulated annealing [35, 60]. However, the application of these solutions on higly distributed environments as those considered by the Edge Computing paradigm is not straightforward.

Indeed, both resource discover and selection processes developed for the scheduling algorithms currently used for Cloud, do not take into account infrastructural features and user-related information in their decision process.

In Sec. 6 we provide a more comprehensive analysis of the limitations of current scheduling algorithms for Cloud in the Edge scenario while an innovative framework for service scheduling in Edge is presented in Chap. 5.

Chapter 3

Dynamic Service Level Agreements

In this chapter we introduce SLAC, a novel SLA specification language specifically devised for Cloud which encompasses a mechanism for the definition of dynamic modifications of the agreements. This new specification language addresses the dynamism of the service requirements allowing the specification of modification actions that can automatically change the service terms, at runtime, when specific conditions are met. Essentially, this formalism define SLAs as *smart contracts* offering an enhanced specification of service requirements, providing a solution for the Research Question 1.

The main features introduced by the devised language are:

- A formally defined syntax and semantics to have non-ambiguous SLAs;
- A number of software tools to support services deployment;
- A novel mechanism to guarantee flexibility and elasticity to the involved parties;
- A brokering mechanism that supports multi-party contracts.

The rest of the chapter is organised as follows. Sec. 3.1 gives a comprehensive overview on the SLAC language. Sec. 3.2 illustrates the SLAC software

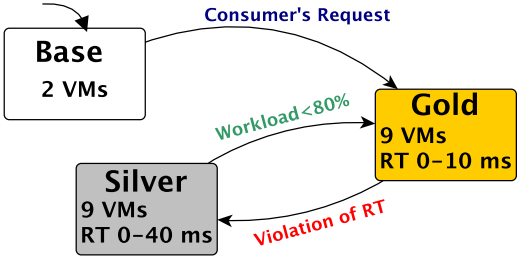

Figure 5: Example of a dynamic SLA in the Cloud domain.

framework. Sec. 3.3 describes the experiments showing the advantages of SLAC and dynamic SLAs.

3.1 The SLAC Language

The novelties introduced by the SLAC language enable the specification of *dynamic SLAs* that allow the parties signing the contract to specify not only the service requirements and quality, but also the modifications that can be applied at runtime to change the valid terms of the agreements.

Indeed, the parties can specify, in addition to the initial agreement, the conditions and the actions that, at runtime, can change the enforced terms of the contract; for example, a consumer can add virtual machines at any time paying the market price plus 0.01 EUR. This allows the formalisation of the horizontal and vertical scalability as well as changes in the service levels.

Therefore, this language, at specification level, is compatible with the concept of *smart contracts* described in Chap. 2.5, since it enables the parties to define a set of modification actions which can be executed to automatically change the SLA terms at runtime.

For this reason, throughout the whole thesis we will use the terms dynamic SLAs and smart contracts interchangeably.

A further example of dynamic agreements is depicted in Fig. 5. The initial valid terms of the SLA characterizes the “Base” service quality, with 2 Virtual Machines (VMs) provided and no requirement about Response Time (RT). During service execution, the consumer may request the upgrade of the quality to

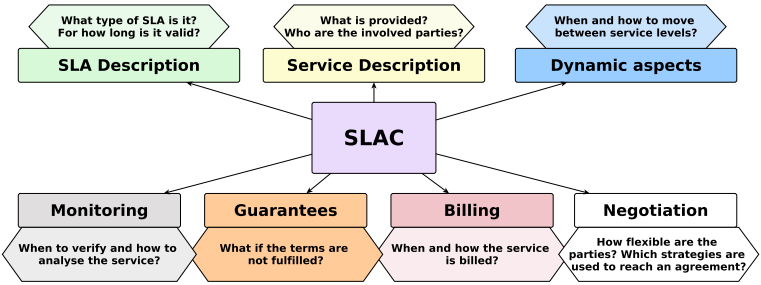

Figure 6: Overview of the main components of a SLA written in SLAC.

"Gold", to get more VMs and the guarantee that RT does not exceed a given threshold. Now, if the RT requirement is violated (due to, e.g., overbooking of resources), the service quality is downgraded to the "Silver" level, where the RT requirement is relaxed. The SLA terms are automatically upgraded again to the "Gold" quality when the service workload goes back to a normal level.

For the sake of readability, we focus on an informal description of the language components and their role in the SLA, by resorting to an illustrative example. The formal definitions of the SLAC syntax and semantics, specified in EBNF and denotational style, respectively, are available in an online technical report [69].

3.1.1 Overview of the Syntax

The distinctive features that differentiate SLAC from other SLA languages can be summarized by saying that SLAC is a domain specific formal language specifically designed for Cloud services that supports the main Cloud deployment models, enables the specification of multi-party agreements and permits dynamic changes of SLA according to pre-defined conditions.

Fig. 6 depicts the main components of a SLA defined using SLAC. We informally present them via the SLA example, reported in Tab. 1, which refers to the brokered provision of a IaaS service, assessed by an auditor.

SLA Description. This component defines the validity period (start and expiration time) and the parties involved in the provision of the services. A party is defined by its name and its roles. In the running example, the validity of

Table 1: Example of a SLAC SLA.

```
Effective From:  01/09/2017
Expiration Date:  01/09/2018
Parties:
 IMT    Roles:  Broker, Provider

 UNICAM Roles:  Provider
 Alice  Roles:  Consumer

 Bob    Roles:  Auditor
Term Groups:
 S_VM_IMT:
   IMT → Alice:cCpu 2 #
   IMT → Alice:RAM 4  GB
   IMT → Alice:Availability 99%
   IMT → Alice:Price 0.20 EUR
 S_VM_UNICAM:
   UNICAM → Alice:cCpu 2 #
   UNICAM → Alice:RAM 4  GB
   UNICAM → Alice:Availability 98%
   UNICAM → IMT:Price 0.16 EUR
   IMT → Alice:Price 0.20 EUR
 L_VM:
   IMT → Alice:cCpu 4 #
   IMT → Alice:RAM 8  GB
   IMT → Alice:Availability 99%
   IMT → Alice:Price 0.35 EUR
Terms:
  1 of S_VM_IMT
Dynamicity:
 on Alice request:
  replace S_VM_IMT with L_VM or
  replace S_VM_UNICAM with L_VM
 on Alice request:
  replace value of L_VM with _old+1 or
  replace value of L_VM with _old-1
 on IMT request:
replace S_VM_IMT with S_VM_UNICAM or
  replace S_VM_UNICAM with S_VM_IMT
 on IMT request:
  if L_VM > 1:
    replace value of
    L_VM.Availability with 97%
 on IMT request:
    replace value of
     L_VM.Availability with 99%
Invariants:
 L_VM in [0,2]
Monitoring:
 S_VM_IMT.Availability:
  Frequency:  10 s
  Window:  1 months
  By:  Bob
 S_VM_UNICAM.Availability:
  Frequency:  10 s
  Window:  20 days
  By:  Bob
 L_VM.Availability:
  Frequency:  6 s
  Window:  15 days
  By:  Bob
Guarantees:
 on violation of any.Availability:
   if Availability > 95%:
    IMT → Alice:  Bonus 1 EUR
   else:
    IMT → Alice:  Bonus 2 EUR
 on violation of any.Frequency:
   Bob → IMT,Alice:  notify
Billing:
 accounting:  hourly
 billing:  monthly
```

the SLA (lines 1-2) is one year. The involved parties (lines 3-7) are instead four: two providers (IMT and UNICAM), a consumer (Alice) and an auditor (Bob). The IMT party has two roles; it acts also as a broker and, when necessary, it may resort to UNICAM for offering the small VM service to Alice.

Service Description. This component specifies the details of the service and its quality. It consists of two parts: groups of terms and instantiation of the valid terms (lines 8-26 of the example). The terms of the agreement express the characteristics of the service along with their expected values. Moreover, for each term SLAC requires to specify the involved parties, i.e., the party responsible for fulfilling the term (a single party) and the contractors of the service (one or more). In our example, for instance, the term IMT $\rightarrow$ Alice:Availability 99% indicates that the party IMT provides to the party Alice the service (i.e., one or more virtual machines) with an availability of (at least) 99%. The explicit definition of the involved parties contributes to supporting multi-party agreements, reducing ambiguity and leveraging the role of the broker. Moreover, it improves privacy and security of the agreement; only parties involved in the term have access to it. For example, when the small VM service is provided by UNICAM, the SLA specification (lines 18-19) indicates both the cost that IMT, as a broker, has to pay to UNICAM, and the cost that Alice has to pay to IMT, which in fact is the same she pays when the service is provided directly by IMT. However, the former information is not accessible to the consumer, who is kept unaware of the agreements between the broker and the providers.Notably, the language offers a set of pre-defined metrics devised for IaaS. Yet, new metrics and their measurement definition can be easily defined without affecting the general language. For a detailed account on this extension procedure we refer to [40].

Another feature of the language is the use of groups to specify terms of agreements with different granularities. A group of terms is identified by a name (unique in the contract) and consists of one or more terms that are valid only inside the group. In the example, we have three groups that represent different types of virtual machines: small VMs provided by IMT (S_VM_IMT) and by UNICAM (S_VM_UNICAM), and large VMs (L_VM). Notably, to use a group in a SLA definition, it is not sufficient to define it, but it is also necessary to instantiate it by specifying the number of instances. For example, the code in lines 25-26 permits to actually deploy one instance of the S_VM_IMT group in the SLA.

Dynamic aspects. This component of the agreement is optional, but it is one of the main novelties of SLAC. Thanks to this, the user can possible run time changes of the enforceable terms of agreement, through the definition of Event-Condition-Actions (ECA) rules. When an event occurs (e.g., a party makes an explicit request), a condition (defined by an expression) is checked and the execution of one or more actions is requested (e.g., the change of the value of a given metric). These rules, for instance, permit agreeing on unilateral changes from one of the parties; e.g., it can be agreed that a party can request changes in the service that can be implemented without the authorisation of other parties. In our example, the consumer can change the type of VM (lines 28-30) and add or remove as many `L_VM`s as necessary (lines 31-33). Instead, the broker can freely change the provider of a small VM service (lines 34-36) and change the availability of large VMs (lines 37-41), with the constraint that the reduction of availability is allowed only in case more than one `L_VM` is instantiated. Replacement actions can use a reserved variable name `_old` to refer to the current value of the term metric or to the number of instances of a given group.

The `Invariants` section constrains the changes agreed in the `Dynamicity` section, by fixing bounds for terms of the contract. When an event triggering changes of the contract is detected, the corresponding changes are applied only if they are compliant with the invariants terms. In our example, an invariant is used to specify that the consumer cannot request more than 2 large VMs.

Monitoring. This SLA component specifies monitoring requirements to obtain information for configuring the monitoring system and defining the auditing agreement. In the example, the `Availability` metric is monitored by the `Auditor` with different accuracy depending on the virtual machine type. For example, the availability of the (more expensive) large VMs is checked by `Bob` more frequently within a smaller time window.

Guarantees. This (optional) component is concerned with the enforcement of the terms of agreement and specifies the actions to be taken in case of violations. Specifically, a guarantee is an ECA rule, where the event typically refers to a term defined in the `Terms` section of the agreement. In our running example, if the `Availability` of any (large or small) VM is violated, but it is still greater than 95%, the broker needs to give a bonus of 1 EUR to the consumer, whilst if it is less than or equal to 95% the penalty is doubled (lines 58-62). Moreover,

if a requirement about the monitoring frequency is violated (due to, e.g., maintenance activity of the provider), the auditor is required to notify the issue to provider and consumer (lines 63-64).

Billing. Since SLAC enables modification of the valid terms at runtime, it is necessary to define periods of accounting and the frequency of billing. In our scenario we defined the accounting period in hours and the billing monthly. Thus, if the consumer used a small VM for 48 hours and then requested a large VM, after one month, the service will be accounted for 48 hours at the price 0.20 EUR/h, and at 0.35 EUR/h for the other days of the month.

3.1.2 Informal Semantics

Intuitively, the SLAC's formal semantics associates a constraint to each Static SLA. It is formulated as a Constraint Satisfaction Problem (CSP), which is evaluated, at design-time, to identify inconsistencies in the specification and, at run time, to check compliance with the monitoring data collected from the Cloud system. To support dynamic SLAs, extensions are needed to allow the use of dynamic CSP [75]. Now, a dynamic SLA is represented as an *automaton*, whose states, representing the SLA states, are labelled by constraints, and whose transitions are labelled by events that trigger the state changes.

At run time, data representing the status of the Cloud are collected by the monitoring system and rendered as a CSP. Then, the CSPs of (the current state of) the SLA and of the monitoring data are combined for evaluation. After such an evaluation, the guarantees specified in the SLA are evaluated and, possibly, the execution of some action is requested to the Cloud manager. Similarly, the ECA rules of the `Dynamicity` section are evaluated; for each applicable rule, the terms of the `Invariants` section are checked in order to apply only those changes that are compliant with the invariants.

More formally, the semantics of a Static SLA, or of a single state of a dynamic SLA, is given by a function $[\![\cdot]\!]$ that given the SLA terms returns a pair composed of a set of group definitions and a constraint. This pair constitutes the CSP associated to (a state of) the agreement, that will be solved by means of a standard constraint solver as described in the next subsection. The automaton representing the semantics of a dynamic SLA is generated as follows. The ini-

Table 2: Semantics at work on our running example.

SLA
`Term Groups:`
S_VM_IMT:
IMT → Alice:`cCpu 2 #`
IMT → Alice:`RAM 4 GB`
IMT → Alice:`Availability 99%`
IMT → Alice:`Price 0.20 EUR`
`Terms:`
`1 of` S_VM_IMT
Constraints:
#SLA Terms
$1 \leq$ `S_VM_IMT` ≤ 1
#Constraints in the S_VM_IMT Group
$2 \leq$ `S_VM_IMT:IMT,Alice:cCpu:0` $\leq 2 \wedge$
$4 \leq$ `S_VM_IMT:IMT,Alice:RAM:0` $\leq 4 \wedge$
$99 \leq$ `S_VM_IMT:IMT,Alice:Availability:0` ≤ 100
$20 \leq$ `S_VM_IMT:IMT,Alice:price:0` ≤ 20

tial state of the automaton is a CSP obtained by applying $[\![\cdot]\!]$ to the terms of the `Term Groups` and `Terms` sections of the SLA specification. Then, all possible new states are created considering events, conditions and the triggered changes to the SLA constraints specified in the `Dynamicity` section and their invariants.

Tab. 2 presents an excerpt of the contract defined in our example scenario and the resulting constraints. Please notice that only terms and group terms of the SLA are considered for constraints generation, while additional information, such as monitoring frequency and monitoring window, are used only as parameters (not constraints) for the monitoring system.

Fig. 7 illustrates the automaton resulting from the interpretation of the dynamic aspects of our example. All transitions can be executed independently of the authorisation of the other parties; for instance, the consumer can request to replace a small VM with a large one and then she can add and remove large VMs according to her needs. This behaviour is an example of how services scalability can be handled and regulated. The provider, instead, can request to lower the availability of large VMs, but only when there are two instances of them.

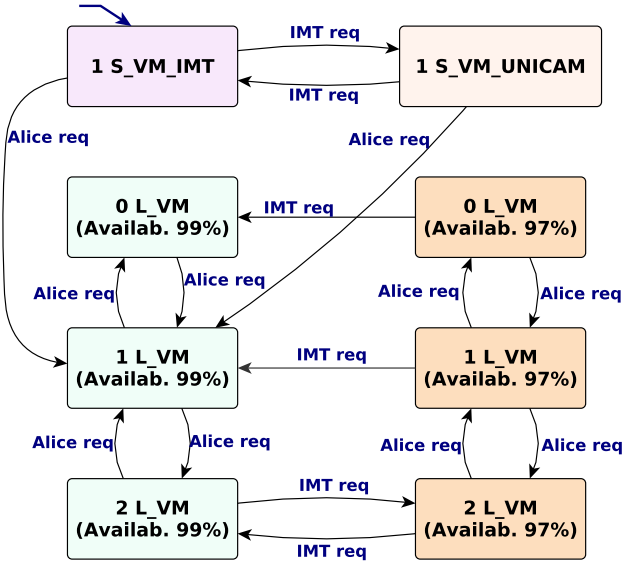

Figure 7: The automaton for the running example.

Notably, if the consumer reduces the number of large VMs to less than two, the availability remains at 97%.

3.2 The SLAC Software Framework

To support the use of SLAC, we developed an open source software framework that covers the whole SLA life cycle. The current prototypical version of the framework[1] works in a centralised fashion and its modules were developed using the Python programming language. Fig. 8 illustrates the main modules of the framework.

The *SLAC Web Editor* can be used by consumers and providers to define SLAs, and to offer or request services. The editor performs auto-completion and syntax verification and was developed using Xtext[2], a framework supporting the implementation of domain specific languages. If a final SLA is defined (e.g.,

[1] Available on `http://sysma.imtlucca.it/tools/slac/`
[2] `https://eclipse.org/Xtext/`

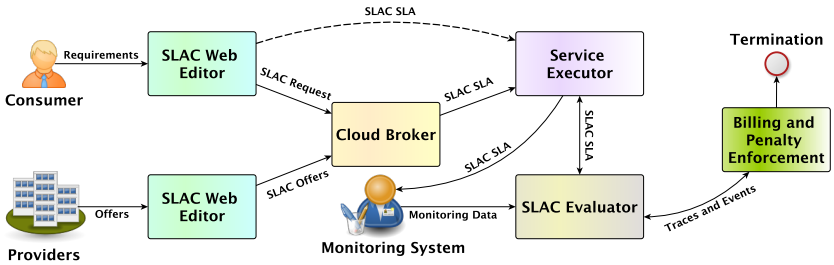

Figure 8: SLAC Management Framework: evaluation process.

due to a previous negotiation or in case of a fixed service), the negotiation phase is skipped. Otherwise, providers and consumers send their offers/request to a broker to match and negotiate services.

The *Broker* matches offers and requests of (dynamic) SLAs. Considering the difficulties of analysing and matching such SLAs, in [57] we proposed and implemented a brokering solution that autonomously matches the compatible offers and defines the final SLA based on the policies and on the preferences of the parties.

The *Service Executor* receives and processes the SLA created in the previous phase and automatically deploys and executes the service; but before doing this it asks the SLAC Evaluator to check SLA's consistencies and also filters the service specifications related to the deployment of the services. The Executor uses a scheduler, based on machine learning solutions defined in [73], that we have integrated within the Cloud management tool OpenNebula[3].

The *SLAC Evaluator* accepts as input a SLA, parses it and generates a set of constraints, corresponding to the specification along with the service definition, that are sent back to the Service Executor (also in case of changes in the state of the SLA). After that, the Evaluator sets up the *Monitoring System* (in our case, Panoptes [74]) to retrieve the data concerning the metrics related to the SLA. Then, the SLA Evaluator receives the monitoring data and transforms them into a set of constraints, whose satisfiability is verified together with SLA constraints. In case of non-satisfiability, the SLA guarantees are evaluated and

[3] `http://opennebula.org`

the due actions are activated. The Evaluator parses the SLA with the ANTLR4[4], by relying on the language's EBNF grammar. The constraints are handled by the Evaluator using the Z3 solver [20]. It is worth noticing that not all monitoring data are required when evaluating constraints; thus evaluation with partial observations of systems is possible.

The *Billing and Penalty Enforcement* module calculates costs and penalties, and bills the parties when scheduled or when traces and events (e.g., violations or changes in the service) are received.

Our software framework supports and automates all phases of the new life cycle proposed in Sec. 2.4:

Service Discovery. SLAC supports the definition of the needs of consumers and available resources of the providers and integrates the solution provided in [57] to verify the compatibility of offers and requests.

Negotiation. A brokering solution to support negotiation of dynamic SLAs is proposed in Chap. 4 and integrated in our framework.

Deployment. The Service Executor is integrated with the OpenNebula toolkit to enable the automatic deployment of services.

Monitoring. The monitoring system Panoptes is automatically configured to monitor the terms of the SLA, while the Evaluator listens to events or requests, processes the modifications, logs them, and requests the changes of terms to the Service Executor.

Billing and Penalty Enforcement. A specific framework module is responsible for this phase of the life cycle.

Termination. The SLA termination is handled by the Billing and Penalty Enforcement module.

3.3 Experiments

This section presents our experimentations to assess the benefits of the framework for both providers and consumers.

[4] `http://www.antlr.org/`

3.3.1 Provider's Benefits

To illustrate the benefits of dynamic SLAs, we use a Cloud testbed with Cloud services provided to multiple consumers. We compare three different approaches: *static SLAs* (SLAs do not change during their lifetime), *renegotiation* (parties can renegotiate the existing SLAs) and *dynamic* SLAs (terms of agreements can be dynamically changed). For the comparison, we analyse the number of violations, the penalties and the total revenue of the provider. The obtained results, discussed below, demonstrate the flexibility of SLAC and its capacity to reduce the number of SLA violations and to improve the revenue of the involved parties.

Use Case Implementation

For implementing the use case, we have embedded the SLAC software framework in a Cloud testbed. The resulting Cloud system has several components (see Fig. 9). The SLAC Evaluator parses and evaluates SLAs for the service specification and requirements, and sends the outcome to the Service Executor. This component is specifically designed to guarantee the correct deployment and execution of services, and to manage the Cloud infrastructure; it schedules services using the approach presented in [72, 73]. The Panoptes Monitoring System provides the status of the system and services to the *Violation Risk Analyser* and to the SLAC Evaluator, and is directly configured by the Service Executor. Specifically, the Violation Risk Analyser measures the risk of the running services of not meeting the deadline specified in the SLA, and notifies the *Renegotiation Decision System*. The violation risk analysis is performed using the Supervised Random Forest technique [9], and is based on the monitoring information and on the SLA itself. Finally, the Renegotiation Decision System creates proposals for modifying the SLA and decides whether to accept changes in the services.

In the experiments, each service is processed according to the workflow description depicted in Fig. 10. It is evaluated only once during its execution lifetime, at time t_r, a random time between the starting and the deadline of the service. In case of *static SLAs*, the services are computed employing the SLA defined at design time. When the service ends or the deadline is met, the system verifies whether the SLA was violated and determines the price and the possible

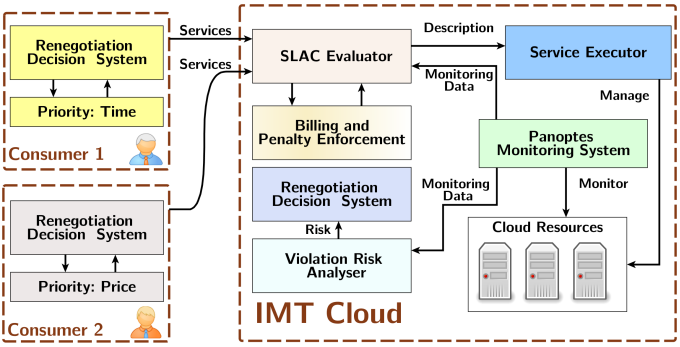

Figure 9: Components of the use case implementation.

penalties. In the *renegotiation* approach, the violation risk is measured first. If it is not higher or lower than given thresholds, the service is provided normally according to the SLA defined at design time. When renegotiating, a party sends a SLA proposal to the other party, who analyses it according to its priorities using a Fuzzy Decision System (see below). If the new agreement is accepted, the service continues and it is evaluated considering the new SLA, otherwise the initially defined agreement remains valid till the end of the service. The *dynamic* approach is similar to the renegotiation one, the only difference is that the involved parties do not have any active role. Indeed, in case of low or high violation risk, the agreement is modified automatically since the changes are pre-defined in the SLA. In both cases, to motivate or compensate a party for the changes, a bonus is given to the other party when a change is performed during the service execution. Although the bonus a priori is usually much smaller than the bonus required for renegotiating the SLA during the execution, for the sake of simplicity, we opt to use the same range of values of the renegotiation approach.

Fuzzy Decision System

To decide whether to accept or refuse a new agreement, the renegotiation approach needs to know the difference between the original agreement and the SLA proposed at renegotiation time. In our use case, to simulate this process, which is typically carried out by a human or by an autonomous decision system,

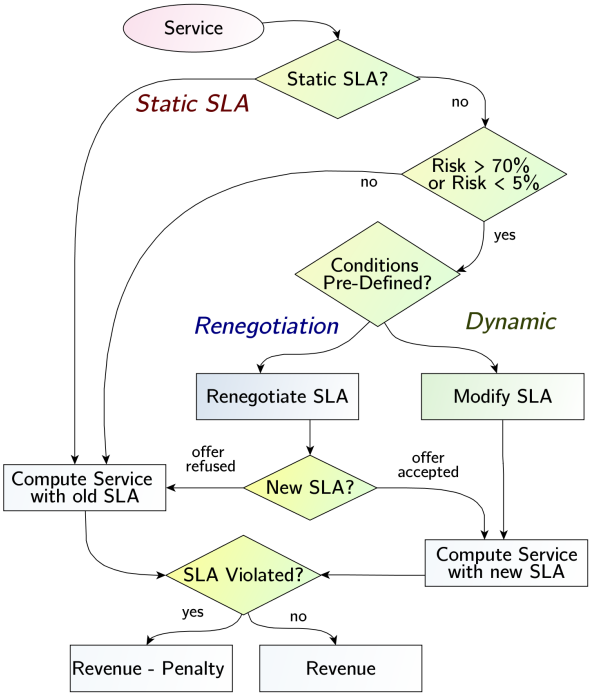

Figure 10: Flow diagram of the service processing for the Static, Renegotiation and dynamic approaches.

we designed a fuzzy logic decision support system inspired by the approach presented in [22].

The decision system takes as input the rates of change for the considered parameters, e.g., the increment in price and decides whether the new SLA is beneficial, neutral or not beneficial to the party. In the case of consumers, the system takes into account also their priorities. In our use case, the inputs are: the deadline for the service (D), the price to be paid for the service (P_r) and the penalty in case of violation (P_e). Tab. 3 exemplifies some rules that are used by the provider's fuzzy decision system. For a complete account of the fuzzy rules and the framework used in the experiments, we refer to the SLAC project's website [68].

Table 3: Fuzzy rules of the provider decision system.

Rule	Evaluation
P_e increases	not beneficial
P_r or D increases	beneficial
P_r and D increase	very beneficial
P_e increase $< 10\%$	neutral

Evaluation

The experiments were conducted with a Cloud with 2 physical machines, providing 12 heterogeneous VMs, in which agents are used to execute services.

Services are generated taking into account the distribution of a trace of a real-world Cloud environment, the Google's Cloud dataset [55], and the same services are executed using all three described approaches. Each service has an associated SLA, which is created along with the service, according to an estimation of the resources necessary to finish the service within the completion time. The considered features are: CPU, RAM, requirements, disk space, completion time and network bandwidth. Different types of services are used in the experiments, such as web crawling, word count, number generation and format conversion, which are close to real-world applications [47]. Service's penalties and prices are generated along with the SLA and are based on the service execution time and on a randomly defined number. Penalties are always higher than the price, since the price is paid even if a service is violated.

The training set for the SLA Risk assessment module is built for every experiment round by executing 1000 services. Then, it is used to train the machine learning algorithm to provide the probability of classifying a new service into the *violated* and *not violated* classes. In each round, new services are generated (for creating the training set and for testing the approaches) and the same services are executed for all approaches. The number of services ranges from 100 to 500 (with 50 services interval). We assume that the services' arrival is a Poisson process, i.e. the time between consecutive arrivals has an exponential distribution and that a service arrives, on average, every 0.7 seconds.

The Fuzzy decision system accepts proposals which are beneficial to the requested party. Therefore, the requester and the Renegotiation decision system

usually offer compensations for the requested party that fulfills the need of the requester; for example, if the violation risk is high, the provider requests more time to finish the service but offers a discount on the price and a higher penalty. The definition of the exact parameters of the considered metrics of the SLA modification proposal, which are used by the renegotiation and the dynamic approaches are randomly generated within a predefined range. The results of

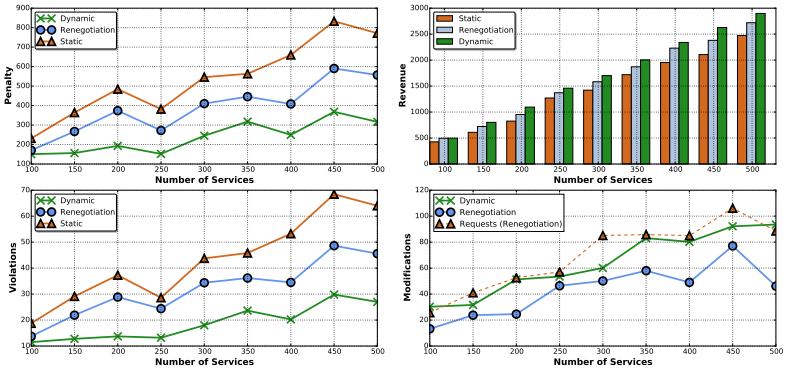

Figure 11: Performance analysis.

these experiments are illustrated in Fig. 11 where different numbers of services are considered. Tab. 4 presents the overall results, relative to the renegotiation and the dynamic approach, expressed as percentages: in the case of penalties and revenue, the results correspond to a comparison with the static approach, whilst for the other features they result from a comparison with the total number of services. Considering the parameters defined for the renegotiation approach and the benefit threshold used in the experiments, around 60% of the modification requests were accepted and carried out. Using the dynamic approach, 21% of the services were modified mainly due to high risk of violation (more than 19%). The total number of modifications is relatively high due to the accuracy of the machine learning algorithm, in which we prioritised the identification of high-risk SLAs. Consequently, the number of false positives increased, i.e., some SLAs that normally would not be classified as high-risk were considered so. In the renegotiation and dynamic approaches, 14% and 19% of the SLAs are

classified as high-risk.

Table 4: Experimental results.

	Renegotiation	**Dynamic**
Modification Requests	24%	0%
Modifications	13%	21%
High Risk	11%	19%
Low Risk	0.1 %	0.2%
Violated High Risk	19%	14%
Violated Low Risk	0%	3%
Penalties	-31%	-64%
Revenue	13%	22%

Overall, the flexibility provided by the dynamic approach increased the revenue by 22% and reduced the penalties by 64%, whilst these measures were only 13% and 31% for the renegotiation approach.

Experiments' Discussion

In the experiments, the benefits of renegotiation and dynamicity heavily depend on the accuracy of the violation risk analyses. The results show that, although the penalties were reduced by 64%, the impact on the total revenue was an increase of around 22%. The main reasons for this difference are: (i) the limited impact of the penalties on the total revenue due to the average number of violations; (ii) the compensation provided to the consumers when a modification is requested, which lowers the price paid for that service and sets higher penalties in case of violation; (iii) the number of modified SLAs which were violated since most of the modification requests increase the penalty as a compensation for the higher service completion time. This suggests that performing an analysis to define the additional time required to avoid violations instead of generating a random number could improve the total revenue.

The experiments were focused on avoiding SLA violations, and in few cases dynamicity and renegotiation were used to improve the revenue of the parties (only around 1.4% of the services were considered low-risk). In most scenarios, these mechanisms can be more aggressively employed to improve the revenue, exploiting improved accuracy of the risk analyser.

Also, the parameters defined in the SLA modification proposal may have a considerable impact on the results. We adjusted these parameters to simulate a real-world situation, where every party defends his interest.

We can conclude by saying that the results demonstrate that SLAC provides flexibility for the parties and significantly optimises SLA management. And, it can always be used together with the renegotiation approach in case not all relevant modifications are included in the SLA.

3.3.2 User's Benefits

The benefits of dynamic SLAs for consumers are manifold; here we focus on the economic impact by discussing a scenario where we compare static and dynamic SLAs. We do not cover the benefits of having guarantees for the dynamic part of the service, since these benefits are difficult to measure.

In our scenario, we compare standard Cloud offers, using static SLAs, with offers based on dynamic SLAs. The former provide a fixed amount of resources for a fixed price during the duration of the SLA, while the latter provide more flexible solutions. The dynamic approach is common in various domains; e.g., telecommunication companies normally offer a mobile plan that provides a bonus (more calls and messages) for the first six months and then the service is changed. In the Cloud domain, there are similar offers but they are typically defined in natural language and cannot be automatised.

In our scenario, an infrastructure provider offers one static service, where a VM costs 0.36 EUR, has 2 GB of RAM and 1 CPU; and two dynamic services, with the same initial characteristics. In the first dynamic offer, called *Small*, the consumer must deploy a minimum number of VMs for the duration of the SLA but has two advantages: the price, which is 0.32 EUR, and the capacity of the VMs that, between 22:00 and 7:00, have 1 additional GB of RAM. The second dynamic offer, called *Big*, requires at least twice the minimum number of VMs of *Small*, but the price is 0.29 EUR and at night the active VMs have additional 2 GB of RAM.

The consumers in this scenario need an infrastructure to run their websites. Thus, the performance of VMs can be also simplified to the maximum requests per hour supported by each VM. All the offers during the day support 1200

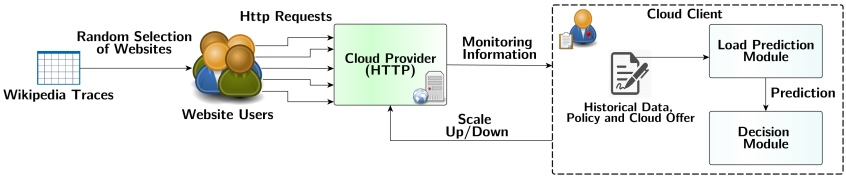

Figure 12: Experiment scenario used to measure the economic impact for consumers.

requests/hour, considering an average size of page in our dataset. At night, the static offer supports the same number of requests per hour, i.e. 1200, while the Small supports 1400 and the Big 1600. The change of the capacity at 22:00 and at 7:00 is automated through the definition of dynamic SLAs, which describe also the minimum VMs per hour.

To measure the impact on consumers, we use a dataset with the real traces of all Wikipedia HTTP requests for 20 days of October 2007 [67]. Then, each time the experiment is run, we select randomly 50 pages of the dataset, filter only the request of this dataset and create a time-series of the total requests/hour from the website. These filtered data are then treated as monitoring information and are passed to the Load Prediction module together with the policy of the consumer and the list of the aforementioned offers. For this module, we adopted the time-series forecasting tool Prophet[5], which is based on an additive regression model that considers growth, seasonality and holidays. With this module, we predict the number of requests for the next 100 hours and send this information to the Decision module. The latter decides which of the offers suits better the needs of the consumer, and, after signing the SLA of the selected offer, on every interaction decides whether to scale up/down the number of VMs (respecting the minimum VMs in the chosen offer). This scenario is executed 50 times for each minimum number of VMs of the dynamic offers and we measure the reduction of the total cost of the Small or Big, selected for the whole duration of the agreement for each website, when compared to static offer. Fig. 12 illustrates this scenario.

The results are the average of 50 runs with 50 random websites and are presented in Tab. 5. The reduction of the cost for the consumer depends heavily on

[5]https://facebookincubator.github.io/prophet/

Table 5: Average cost reduction of dynamic SLAs in comparison with static ones.

Small: Min VMs/h	Big: Min VMs/h	Cost Reduction
1	2	25.68%
2	4	21.94%
3	6	16.16%
4	8	8.15%

the type of website. Intuitively, sites with low traffic are penalised by the fact of a minimum number of VMs and the discount in the price might not compensate this disadvantage. In any case, the total reduction in cost for the consumers when using dynamic SLAs is up to 25%, with 1 and 2 VMs as minimum, for Small and Big offers respectively, and has 8% as lower bound, with 4 and 8 VMs.

Chapter 4

Smart Contract Negotiation

One of the main challenges introduced by the formalism presented in the previous chapter is the negotiation of offer/request formulated with this new language, as stated in the Research Question 2. Indeed, matchmaking and bilateral negotiation techniques mostly adopted for the negotiation of traditional contracts, cannot deal with the dynamism introduced by this new type of smart contracts, as previously described in Sec. 2.6.

In light of this challenge, we devised a novel framework for the automated negotiation of smart contracts in Cloud, that is presented in this chapter. More specifically, the main contributions of the framework are:

1. The definition of a simple formal language to specify interactions between offers and requests modelled as SLAC expressions;

2. An open source framework for adaptation, consistency check, verification of contracts properties and suggesting the changes necessary for reaching an agreement;

3. A methodology for autonomous negotiation that, by relying on the utility functions and on the level of flexibility of providers and consumers, determines the costs of agreeing.

The rest of the chapter is organised as follows: in Sec. 4.1 we present a formalism for the definition of smart contracts and for offers and requests of

Cloud's providers and consumers. In Sec. 4.2 we discuss negotiation framework, utility function and compatibility check, whilst in Sec. 4.3 we contrast the performance of our approach with more classical ones.

4.1 Modelling Smart Contracts

For the negotiation of SLAs expressed with SLAC, or smart contracts in general, it is needed to properly model the contract offer/request in a way suitable for the tasks usually involved in the negotiation process, as compatibility analysis and terms bargaining.

We have devised a formalism for modelling smart contracts suitable for these tasks. Essentially, we model the behaviour of SLAC contracts in terms of transitions and states (i.e., transition system) defining a model which is also suitable for contract negotiation and management tasks.

Before introducing our formalism, we will briefly describe the dynamic parts of SLAC that are relevant for the negotiation process.

To define offers and requests in SLAC in addition to static part of the SLA, i.e., the initial valid terms of the SLA, the parties involved in the negotiation may define how the valid terms may change. Here, we focus on the dynamic definition of the SLA (for details on the static part we refer to [71]). This definition requires specifying: the type of action; the condition of the action; and the modifications to be carried out in the SLA. The *type of action* defines either the action that the party wants to include in the agreement (*demands*); or the concession of an action in case the party demands it (*grants*). The *condition* is an abstraction encompassing several types of events such as, term violation, consumer or provider request (without need for authorisation) and request authorisation. Three types of *modifications* can be applied once the conditions are satisfied: a new term is introduced in the SLA (*add term*), or an existing term is removed (*delete term*) or modified (*replace value of*).

To model this specification, we devised a formal model which allows us to explicitly describe the behaviour of smart contracts in terms of state transformations, in a stepwise fashion. Contracts are described as pairs (P, σ) where P is a term of a process description language that permits describing the actual actions peers are willing to perform and σ represents the process data, i.e., the values of

specific attributes of processes in a given state. The transitions between states are defined by means of a set of rules (Sec. 4.1.2).

4.1.1 Syntax

To specify the actual states of a process, we need to introduce the syntactic elements for denoting processes and their data. A *process expression* describes the process components as a set of syntactic operators, whereas the *process data* is expressed by a function which associates variables used by process with their values. We have $\sigma : \mathcal{X} \to \mathcal{D}$, where $\mathcal{X}$ denotes a set of variables and $\mathcal{D}$ a set of values. To modify σ we use an update operation $\sigma[v/x]$ which substitutes the value associated to variable x with the value $\mathcal{E}(e)\sigma$ obtained by evaluating expression e after replacing its variables with their values in σ. Moreover, we will use *updates* of the form $z := e$ to assign the value resulting from the evaluation of e grounded in σ to variable z. We shall call them upd and use $\overline{upd}$ to denote sequences of updates.

A process expression is a term generated from the BNF-grammar reported in Tab. 6. The term P denotes a process which can be identified by a process constant $K \in \mathcal{K}$; the *action prefixing* operator $\mu(\overline{upd}).P$ denotes a process that executes the action $\mu(\overline{upd})$ and then behaves as the process P. The term $\mu \in \mathcal{L}$ represents an action label and $\overline{upd}$ denotes the sequence of updates performed together with the action. The term $P + P$ denotes the non-deterministic choice expressing an alternative among possible behaviours; the *parallel composition* operator, $P \mid P$, models the parallel execution of processes. The *conditional* operator permits building a process whose behaviour depends on the value of a boolean expression $\mathbf{b} \in \mathcal{B}$ and on the values of the variables in σ.

4.1.2 Semantics

The semantics of this formalism are given by a set of inference rules of the form:

$$\frac{\text{premises}}{\text{conclusion}}$$

Both premises and conclusion are represented by triples of the form $\left(P,\sigma\right) \xrightarrow{\mu} \left(P',\sigma'\right)$ [21]. The term $\left(P,\sigma\right)$ represents the state of the automata, where P is

Table 6: Syntax.

process names	$\mathcal{K}$		
variables	$\mathcal{X}$		
action labels	$\mathcal{L}$		
boolean expression	$\mathcal{B}$		
process	$P ::=$		
null process		0	|
process constant $K \in \mathcal{K}$		K	|
action prefixing		$\mu(\overline{upd}).P$	|
alternative choice		$P_1 + P_2$	|
parallel composition		$P_1 \mid P_2$	|
conditional		if **b** then P else P	

the process expression and σ is the function providing the current values of the variables associated to that state. The transition $\xrightarrow{\mu}$ between two states denotes that (P,σ) evolves in (P',σ') after the execution of the action μ modifying both the process and the data. Therefore, we defined a set of inference rules for each operator described in Sec. 4.1.1 required to specify the behaviour of smart contracts in Cloud and reported in Tab. 7.

This set of semantics rules can be conceptually divided in two subsets. The set of rules which model the dynamic behaviour of a single contract and the set of rules needed to model the synchronisation between two contracts. In the remainder of this section we will provide a description of all the transition rules defined.

The *Definition* rule states that if the process (P,σ) can execute the action μ and evolve in (P',σ') then the process named as K, where K is a process constant identifying the process (P,σ), can perform the same transition $(P,\sigma) \xrightarrow{\mu} (P',\sigma')$.

The *Update* rule describes the data update action. Given a data expression $\overline{upd}$, as defined in Sec. 4.1.1, a process executes the action μ, parametrised by $\overline{upd}$, that updates the variable values contained in $\overline{upd}$ and then behaves like P. Essentially, this rule can be used in order to describe the effect of a transition from one state to another one in a smart contract, modelling the modification applied to the term values.

Table 7: Semantics.

$$\frac{\overline{upd} = \{\ldots, x_i := e_i, \ldots\} \quad \sigma' = \sigma[\ldots, \mathcal{E}(e_i)\sigma)/x_i, \ldots]}{(\mu(\overline{upd}).P, \sigma) \xrightarrow{\mu} (P, \sigma')} \textit{(Update)}$$

$$\frac{(Q,\sigma) \xrightarrow{\mu} (Q',\sigma') \quad (P,\sigma) \xrightarrow{\alpha} (P',\sigma')}{(P,\sigma)|(Q,\sigma) \xrightarrow{\gamma(\alpha,\mu)} (P',\sigma')|(Q',\sigma')} \textit{(Sync)}$$

$$\frac{(P,\sigma) \xrightarrow{\mu} (P',\sigma')}{(P,\sigma)|(Q,\sigma) \xrightarrow{-\mu} (P',\sigma')|(Q,\sigma')} \textit{(Inter)}$$

$$\frac{(P,\sigma) \xrightarrow{\mu} (P',\sigma')}{(P,\sigma) + (Q,\sigma) \xrightarrow{\mu} (P',\sigma')} \textit{(Choice)}$$

$$\frac{(P,\sigma) \xrightarrow{\mu} (P',\sigma') \quad \mathcal{E}(\mathbf{b})\sigma = \text{true}}{\text{if } \mathbf{b} \text{ then } (P,\sigma) \text{ else } (Q,\sigma) \xrightarrow{\mu} (P',\sigma')} \textit{(If (true))}$$

$$\frac{(Q,\sigma) \xrightarrow{\mu} (Q',\sigma') \quad \mathcal{E}(\mathbf{b})\sigma = \text{false}}{\text{if } \mathbf{b} \text{ then } (P,\sigma) \text{ else } (Q,\sigma) \xrightarrow{\mu} (Q',\sigma')} \textit{(If (false))}$$

$$\frac{(P,\sigma) \xrightarrow{\mu} (P',\sigma') \quad K \triangleq (P,\sigma)}{K \xrightarrow{\mu} (P',\sigma')} \textit{(Definition)}$$

We defined also the operators *Choice* modelling the non-deterministic composition of two processes, expressing the possible execution progresses determined by an external action μ.

The conditional rules *If*, instead, evaluates the boolean expression $\mathcal{E}(\mathbf{b})\sigma$ over the values of the current process data σ and then evolves either in $\left(P', \sigma'\right)$ or in $\left(Q', \sigma'\right)$ according to the result of the evaluation.

The rules *Inter* and *Sync* describe, instead, how the synchronisation between two processes evolves. These rules are not directly used for modelling a single contract but they will be used in the next section to specify a more complex model, based on the composition of the models representing the request and the offer, useful for the negotiation process.

The rule *Inter* models the interleaving of the actions of the two processes, while the the *Sync* operator, instead, models the synchronisation between two processes whenever they are willing to execute complementary actions. The function $\gamma(\alpha, \mu)$ is fundamental to model compatibility between the planned actions of the two components.

4.1.3 An Example

We now show how to model a smart contract and a consumer service request, depicted in Fig. 2, with the formalism we devised. Initially we do not consider the synchronisation of offers and requests. The textual representation of the contract is reported in Tab. 8. The *Term* section describes the initial valid terms of the contract, whilst the *Dynamism* describes the set of dynamic actions. The formal specification of this contract is the following:

$\left(P, \sigma_0\right) = \left(D1 + D2 + G1, \sigma_0\right)$ where

$$
\begin{aligned}
D1 = \ & \textbf{if } \sigma(vm) == 2 \\
& \quad \textbf{then } \mu\left(vm := 4, rt := 20\right).\left(P, \sigma\right) \textbf{ else } 0 \\
D2 = \ & \textbf{if } \sigma(vm) == 4 \textbf{ and } \sigma(rt) == 20 \\
& \quad \textbf{then } \mu\left(vm := 4, rt := 30\right).\left(P, \sigma\right) \textbf{ else } 0 \\
G1 = \ & \textbf{if } \sigma(vm) == 4 \textbf{ and } \sigma(rt) == 30 \\
& \quad \textbf{then } \alpha\left(vm := 2\right).\left(P, \sigma\right) \textbf{ else } 0
\end{aligned}
$$

The pair $\left(P, \sigma_0\right)$ specifies the process expression P and the process data σ_0

that associates appropriate values to all process variables. In this example, the process is expressed through the non-deterministic choice among three different processes, each of them representing one action. Thus, considering the format of the actions, we specify each action in our formalism using the conditional operator defined in our language. For example let us consider the action $D1$ in Tab. 8, where is defined a guard condition on the VM term that, if satisfied, allows the consumer to replace the number of VMs and to add a new term in the contract. Therefore, this action can be clearly specified through the conditional operator described in Tab. 6, where the boolean expression $\mathcal{E}(\mathbf{b})\sigma_0$ describes the guard condition evaluated on the value of a variable *vm* storing the current value of the VM term and where the action μ models the action that updates the current process data according to the modifications defined in *D1*. The remaining processes $D2$ and $G1$ have the same structure as $D1$ and similar meaning. Another difference is given by the action name used for the action $G1$ which differs from the others actions in order to point out that the difference from the granted to the demanded actions. Therefore, we can model a dSLA as a process in our formalism, whose behavior is described by the LTS of Fig. 2 that can be obtained by using the inference rules introduced in Tab. 7.

Table 8: Consumer's SLA excerpt.

Terms:
VM:2

Dynamism:
On consumer request:
if *VM==2*
then *replace value of VM with 4*
and *add term RT 20ms*
if *VM==4 and RT==20*
then *replace value of RT with 30ms*

On provider request:
if *VM==4 and RT==30*
then *replace value of VM with 2*
and *remove term RT*

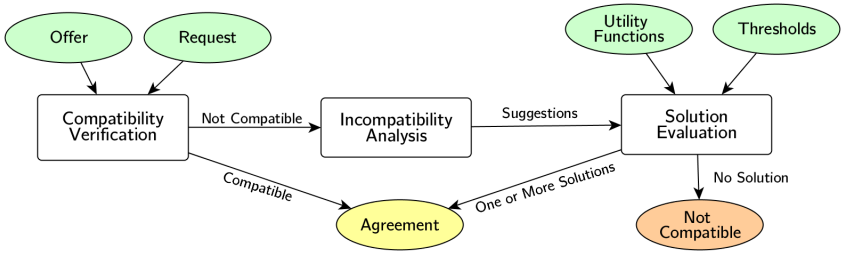

Figure 13: Autonomous negotiation methodology.

4.2 Autonomous Negotiation of Smart Contracts

In the previous section we defined a formal language for modelling the interaction of offers and requests. However, precise matching offer/requests is rarely possible because of the expressiveness of smart contracts and because it requires compatibility at each single state and for each transition. A methodology thus is needed for the autonomous negotiation of smart contracts. The main steps of this methodology are reported in Fig. 13. It demands first (i) to evaluate compatibility of a request and an offer (*Compatibility Verification* - Sec. 4.2.1). In case of incompatibility, (ii) it looks for the motivations and suggests changes to obtain compatibility (*Incompatibility Analysis* - Sec. 4.2.2). Finally, based on a utility functions and on a minimal thresholds for accepting the solution, (iii) the methodology evaluates, according to the suggestions in the previous step, the possible solutions and selects the best one according to the utility functions of the involved parties (*Solution Evaluation* - Sec. 4.2.3).

4.2.1 Compatibility Verification

The compatibility verification has four main steps: generation of the automata for the offer and for the request, consistency and property verification, closure operations on the generated automata and development of a new model for verification of the compatibility between request and offer.

The first step generates two automata, one to model the offer and another to model the request. The initial state of these automata is the same since we assume that a negotiation over the initial terms of the contract has already taken place.

The second step of the compatibility verification looks for errors in the specification of the smart contract, which may generate, for example, a subsets of unreachable states (an undesired behaviour).

The third step performs a closure operation over the transitions of the automata. For each pair of nodes connected by a transition modelling a demanded action, it is checked whether in the counterpart there exists a transition between these nodes representing a granted actions. If such transition is not available, it is checked whether there is a sequence of only granted actions between the two nodes and from all possible intermediate nodes there is no outgoing arc demanding transitions by the counterpart. In case these conditions are satisfied we add a transitions between these nodes, in the counter part automaton. The new transition carries information to be used for checking compatibility of request and offer process.

The last step, develops a *compatibility model* which detects the reasons of incompatibility and produces a new automaton obtained as the parallel composition of the offer and request automata as specified by the *Sync* and the $Inter$ rules in Tab. 7.

Starting from the initial state, the γ function of the *Sync* operator checks the compatibility of demanded and granted actions. If they are compatible the automaton evolves to a new state whose label is given by the output of γ and the new set of possible actions is given according to the set of the possible negotiators' actions in the new state.

If they are not compatible (i.e., demanded but not granted) the $Inter$ operator defines the transition to a new state whose label denotes incompatibility and the new set of actions is given by the possible negotiators' actions in that state.

The obtained model is a new automaton containing all states and transitions of the request and offer automata with new labels on the edges that contain the information needed to check compatibility. Different labels are used to specify if the transitions are matched: missing transitions, i.e., the actions demanded but not granted, are denoted as $-\mathbf{Y}$ where $\mathbf{Y}$ is the action id; $*\mathbf{X}|\mathbf{Y}$ represents the matching of a demanded action $\mathbf{X}$ with the granted action $\mathbf{Y}$; and $+\mathbf{X}|\mathbf{SAT}$ specifies that the action $\mathbf{X}$ is not directly provided by the counterpart, but it is still compatible due to the transitivity closure, which denotes the availability of the counterpart to accept this actions by adding a new edge (**SAT**).

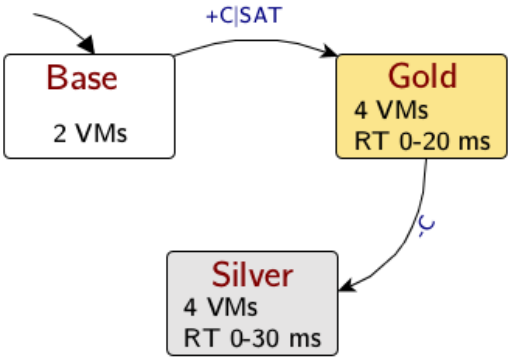

Figure 14: Compatibility model of requests and offers in Fig. 2.

To better understand how this model is computed, let us consider the service request and offer automata in Fig. 2. The resulting compatibility model, obtained as the parallel composition of these two automata, is depicted in Fig. 14. Notably, between "Base" and "Gold" there is no immediate matching of demanded and granted transition. Nevertheless, we can still define a transition matching (+**C**|**SAT**) since we can apply the transitivity closure between these states in the provider's automaton, denoting his willngness to grant the transition even though it was not initially exhibited. However, there is neither a direct nor an alternative matching for the consumer's demanded transition between "Gold" and "Silver". In the model we report this missing passage by relabelling the transition as missing (**-C**). Notably, in the new automaton only the demanded missing actions are included, whereas the granted missing actions are not (e.g., from "Silver" to "Base" in the consumer's automaton). These actions indicate only the party's availability and do not affect the overall party's compatibility if the counterpart does not demand them, therefore, they are not included in the final agreement.

4.2.2 Incompatibility Analysis

This step verifies the main causes of incompatibility and provides a set of suggestions of modifications in the service request and offer to reach an agreement

suitable for both parties. The core component of this process is the compatibility model generated in the previous section. All the *missing paths* in the automaton, i.e., the set of transitions and states requested by a party but not provided by the counter part, are detected. All the model transitions labelled as $-\mathbf{Y}$ are considered and modifications to be applied by either the consumer or the provider to solve the compatibility issue for each missing path are proposed. The proposed modifications are somehow straightforward, they consist in demanding to the party requiring a given missing path to renounce to it or to the counterpart to add it to its proposal.

4.2.3 Solution Evaluation

The set of all valid combinations of modifications, that are the outcome of the incompatibility analysis, are the possible solutions for adapting requests and offers in order to find an agreement between the parties. We evaluate each solution according to the following procedure: we first adapt the compatibility model according to the solution, then use the utility functions provided by each negotiator to evaluate the quality of the new model and verify whether they are higher than the threshold provided by the parties. After verifying all possibilities, we select the best solution, considering both utility functions.

Although any kind function can be implemented by the parties, in Eq.4.1, we propose an example utility function to be used in our experiments. This function provides the overall score of a solution s measuring its quality by a preference score $u(s)$, penalized by an adaptation cost $c(s)$.

$$U(s) = u(s) - c(s) \tag{4.1}$$

The preference score $u(s)$ measures the utility of the current solutions s for the negotiators n proportionally to the number of transitions matched $\mathcal{T}^s_{m,n}$ over the number of the transitions requested $\mathcal{T}^s_{r,n}$ as in Eq. 4.2. In other words, this function measures the similarity of the initial offer or request with the proposed solution.

$$u_n(s) = \frac{|\mathcal{T}^s_{m,n}|}{|\mathcal{T}^s_{r,n}|} \tag{4.2}$$

The adaptation cost $c_n(s)$, instead, penalises the quality of a solution s for the

negotiator n as function of the ratio between the transitions $\mathcal{T}^s_{mod,n}$ to be added or removed in the original automaton over the total number of transitions $\mathcal{T}^s_{o,n}$ it contains. In this way we evaluate the cost of a solution, penalising the ones which have a big impact on the original automaton.

$$c_n(s) = f\left(\frac{|\mathcal{T}^s_{mod,n}|}{|\mathcal{T}^s_{o,n}|}\right) \tag{4.3}$$

The definition of the cost function is crucial for the evaluation process and can vary according to the negotiator's strategy. For instance, the adoption of a cost function that considerably penalises the modifications applied will yield a low overall utility score for solutions providing a high preference score. Moreover, each negotiator defines a preference threshold $W_n \in [0,1]$ that specifies the overall quality that the new agreement, obtained after the application of s, should satisfy to be accepted.

4.2.4 Use Case Example

To illustrate the devised negotiation process, we define a scenario where a broker receives a consumer's service request and attempts to find providers matching his request. The request and offer are depicted in Fig. 2. Let us assume that the consumer adopts a linear cost function in the utility function in Eq. 4.1 and defines the preference score to be satisfied as $W_c = 0.9$. The first step is the compatibility verification of the request and offer. The broker generates and analyses the compatibility model depicted in Fig. 14. The offer and request are incompatible due to the missing transition (labelled $-\mathbf{C}$) between "Gold" and "Silver". Therefore, the initial utility score for the consumer is $U_c = \frac{1}{2} = 0.5$ since only one demanded transition is granted by the provider through the transitivity closure. Thus, the incompatibility analysis process elaborates suggestions to solve this problem. In this case, only two suggestions are formulated: either the consumer removes the transition "Gold" to "Silver" or the provider includes this transition in the offer. Then, the solution evaluator applies both solutions, iteratively, in the compatibility model and evaluates the utility score for the consumer. By removing the missing transition from the consumer's request, the utility score of the new proposal is $U_c = \frac{1}{1} - \frac{1}{3} = 0.66$. This solution maximises the preference score, since

after removing this transition from the original request we have a full matching of the other ones. However, since we use a linear cost function to penalise the modifications of the original request, the preference score is penalised in 0.33 (1 modified transition over the three initially defined). By applying the other solution, instead, the utility score for the provider is $U_c = \frac{2}{2} - 0 = 1$. The preference score is still maximized, since the provider adds the demanded transition and moreover there is no penalisation cost because the solution is not affecting the consumer's request. Therefore, as the provider does not have specific requirements, we can consider the second solution as the optimal one since the utility score satisfies the consumer's preferences.

4.3 Validation of our Approach

In this section, we present the experiments, where we analyse the number of matches between a dynamic request and 100 dynamic offers from different providers. The experiments compare our approach with two additional ones in terms of providers matches. The first one only matches providers, whose offer is 100% compatible with consumer's evolving needs. The second one, based on [28], takes into account only the initial requirements of the consumer and matches with providers that satisfy his requirements at the start without worrying about future needs of the peers. With our experiments, we demonstrate that flexible solutions for the negotiation of smart contracts are necessary and that matching dynamic offers and requests is not only feasible but that, with the right level of flexibility, we can get close to the number of matches of the second approach, which is limited to static smart contracts negotiation.

4.3.1 Scenario

We set up a Cloud scenario where consumers send service requests to a broker, who seeks among 100 providers those that can satisfy the consumer's requirements. The service request and offer are formalized through a smart contract composed of *static* and *dynamic* parts. In the static part we defined 3 negotiable terms (price, response time and number of VMs) and assign, for each term i^n of the negotiator n, a range of acceptable values $[x^n_{i,min}, x^n_{i,max}]$ together with a pref-

erence score w_i^n representing the importance of the value i for the negotiator n. In the dynamic section, instead, we defined what parties demand and grant to the counter parties, as described in Sec. 4.1. It specifies the actions parties permit and require in the agreement, including the conditions to change the valid terms of the agreement. In this experiments, the condition and modification actions are randomly generated. We analyse the provider matching rates of three different negotiation approaches:

1. *Bilateral negotiation (BN).* Only the static part of the smart contracts is considered in the negotiation. Of course, it is easier to find compatible providers when only an agreement on the initial terms is needed. We included this approach in the experiments to provide insights on the differences between negotiating static and dynamic smart contracts. We implemented this solution based on [28], which defines a counteroffer generation mechanism allowing the negotiator's party to modify the term values at each negotiation round, adopting a time-dependent strategy, to maximise the overall utility of the agreement. We chose this approach since it is commonly used as the baseline comparison for negotiation in Cloud, for example, in [18, 85].

2. *Smart contract matchmaking (CM).* The broker verifies whether each providers' offer is 100% compatible with the consumer's requirements of the static and dynamic sections. This approach is used to show that, without a flexible solution, dynamic smart contracts are rarely compatible.

3. *Smart contract autonomous negotiation (AN).* The broker uses the proposed approach to adapt the consumer's requirements and the provider's offer (following the methodology defined in Sec. 4.2) to find a solution which satisfies both negotiator preferences. We evaluated the performance of this approach with respect to different negotiator preference thresholds and different utility functions.

4.3.2 Result Analysis

We execute the previous scenario 100 times. For every execution, we randomly generated both the consumer's request and the 100 offers of the providers. Then,

we assessed the performance of the three aforementioned negotiation approaches by evaluating the number of matched providers. We employed two different versions of the AN approach, which differ on the cost function $c(s)$. The first is characterized by a linear function, which represents a hard cost function, and the second by a quadratic cost function which is more flexible and weighs less the impact of modifications. For each negotiator we specify a preference threshold, as described in Sec. 4.2, representing the consumer's and provider's flexibility to accept a request or offer. Then, we tested the performance of the AN approach for different values of the threshold ranging between 0.1 to 1 with intervals of 0.1, which evaluates the impact of the parties' flexibility in the results.

The results are reported in Fig. 15. Since they do not use a preference threshold, the BN (grey line) and CM (red line) approaches are constant, whereas the AN performance is affected by the adopted preference threshold and cost function. We observe that, for low threshold (flexible peers) the number of matched providers is even larger than BN. This is explained by the fact that, although the BN approach requires matching only the static part of the smart contract, with our approach and with very flexible parties, few transitions must match. For example, in the extreme case of 0.0 threshold, even if not a single transition is equal to the original offer or request, we can obtain a matching. On the other hand, with very high threshold the number of matched providers drastically decreases because the peers require most transitions demanded to be granted by the counter-party, e.g., with threshold 1, no modification is accepted.

Regarding the two cost functions, we can observe that as the threshold values increase the difference between the number of matched providers also increases. This reflects a choice of behaviour from each party, which means that, even if the parties use the same threshold, with different cost function they can still be more or less prone to accept modifications on the offer/request. In our case, with the linear cost function (brown line) negotiators are less prone to change their requirements than negotiators using a quadratic cost function (blue line), since the values are always between 0 and 1.

Overall, almost no offer is 100% compatible with the consumer's requirements, as can be seen from the results of the CM approach, which calls for more flexible solutions for the negotiation of dynamic smart contracts. Moreover, despite the complexity faced with the introduction of dynamic smart contracts,

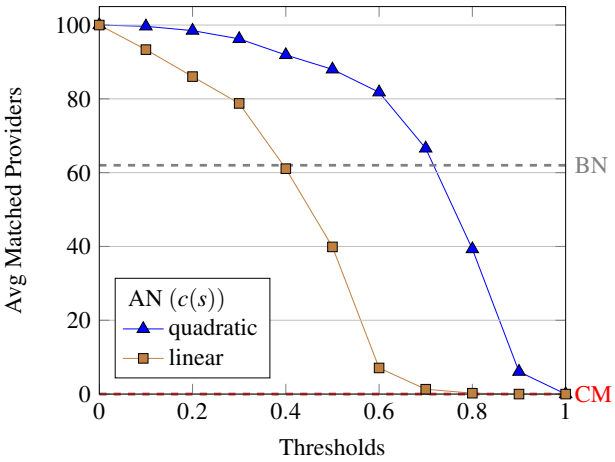

Figure 15: Negotiation results of the BN, CM and AN approaches.

the average number of providers matched with our approach is relatively close to providers matched by the BN approach, which only considers the static part of the agreement. Therefore, we can conclude that these results emphasise the effectiveness of our autonomic negotiation approach.

Chapter 5

Scheduling Latency-Sensitive Services in Edge Computing

The lack of solutions for scheduling of latency sensitive applications, we noticed during the study of possible scenarios suitable for the application of SLAC, motivated the work described in this chapter, also stated in third research question of the introduction.

Essentially, the main factors that lead us to this work are two: we thought that the dynamism characterizing latency-sensitive applications could make this type of services a very good use case for putting SLAC at work; we could not find any work on scheduling this kind of applications in a Edge scenario which represents the most suitable deployment platform for latency-sensitive applications.

The main contributions of this chapter are:

1. A novel Edge scheduling framework for latency-sensitive services, that is latency, bandwidth and resource aware.

2. The analysis of performance of different deployment solutions, namely Cloud, CDN and Edge, for latency-sensitive services, which shows the advantages of Edge computing in this context.

3. The evaluation of the proposed scheduling framework in comparison to a solution for latency-sensitive services in Cloud, which shows the benefits of using an algorithm specifically devised for this architecture.

Experimental results show that Edge-based deployment solutions actually guarantee lower end-to-end latency compared with the alternatives Cloud and CDN. However, we also proved that, to really exploit the advantages of Edge, specific scheduling algorithms need to be adopted, which take into account specific infrastructure related characteristic such as latency, bandwidth and resource availability.

The structure of this chapter is organised a follows. In the next section, we present the related works; in Sec. 5.1 we provide a motivating example discussing the benefit of Edge-based deployment approach for latency sensitive applications such as a live video streaming service; in Sec. 5.2 we present the scheduling approach we devised; in Sec. 5.3 we describe the experimental setup used for the evaluation of our approach together with the results we obtained.

5.1 Motivating Example

To better motivate the research question we are dealing with in this chapter, we consider a simple scenario, based on live video streaming services, that will be used also for the experimental analysis.

Live video streaming services allow to stream a video recorded by any device equipped with camera in nearly real-time to a large number of mobile or desktop audience globally. Currently, due to the advances in networking technologies these services are becoming very popular and attracting the attention of big companies such as Twitter, Facebook and Google who developed their own live video streaming services. To better understand what are the challenges underlying this kind of services, let us first quickly describe the live streaming service workflow depicted in Fig. 16.

The first step is the encoding of input video in a high quality stream, using either local camera encoder or an external one installed in remote server. Afterwards, the encoded video is given as an input to the transcoding operations which create streams in different resolutions and bitrates on the fly. This process is fundamental in order to reach a very broad audience. Indeed, creating new streams at various bitrates and resolution allows an adaptive distribution of the video content according to the device used to access the stream and bandwidth condition, guaranteeing a high quality of experience for many users. The

multiple streams in output from the transcoding process are then processed by a media server responsible of packaging the video streams, according to the specifications (e.g., chunk length, codecs and container) defined by the streaming protocols (e.g., HLS, HDS, etc.) supported. The video chunks are then ready to be directly delivered to end users according to their location, bandwidth and streaming protocol implemented by the media player they use, or first disseminated on multiple servers closer to them.

Although the service workflow may look quite simple, all the steps described are characterized by some critical issues that will affect the overall user engagement to the video stream. Factors impacting user engagement, as reported in [23], are the join time (i.e., the duration from the player initiates a connection to a video server till the time the play starts), the buffering ratio (i.e., percentage of the total streaming session time spent in buffering) and the video quality. All these metrics are directly affected by the transcoding and distribution steps of the streaming workflow, which strictly rely on the service deployment solution adopted. Currently, streaming service providers adopt Cloud-based deployment solutions in order to face the high and quickly changing processing resource and bandwidth requirements. In the simplest Cloud-based architecture, as depicted in Fig. 18, the incoming video stream is usually encoded in a high quality video locally and then uploaded to a cloud server for transcoding and packaging operations.

In this way, exploiting the elasticity provided by the Cloud infrastructure, it is possible to dynamically manage the resource needs according to the current service load. However, this deployment solution is not the best approach to maintain a low buffering ratio and a high video quality. Whereas extensive computation required for the transcoding operations can be managed according to the current needs, the best effort delivery of the packets from a cloud server to final users can still incur downgraded video experience. Indeed, the path between users and the stream origin server in the Cloud usually involves multiple hops and then the probability of experiencing link congestion along the route is high. Congested links incur a low throughput and high packet loss and delay jitter, introducing high network delay and discontinuous packet delivery rate, causing a drop in the video quality.

To face these network issues, the most widely adopted solution currently is

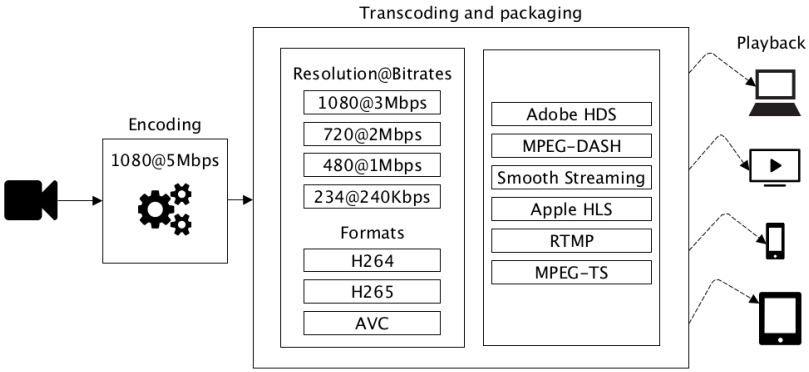

Figure 16: Live video streaming workflow.

content delivery networks (CDNs) for the distribution of streaming contents. In this scenario, as depicted in Fig. 19, all the transcoding and packaging operations are still executed on the Cloud, but the media contents are then distributed over a network of cache servers located closer to end users. In this way, user requests are automatically directed to the closest cache node, which forward to the Cloud only the requests whose content is not already available on that node. Therefore, an improved bandwidth consumption is provided and then the risk of network congestion is lower, reducing the buffering ratio while increasing the video quality experienced.

Nevertheless, CDNs have originally been designed for the distribution of static contents and adopting them for the distribution of dynamic contents as live video streams is not straightforward. The main issues arise from the volume and requirements of live video streaming users. Indeed video content accounts for 70% of whole Internet traffic [8], which is much higher compared to the traffic generated by other web-applications. Moreover users demand high quality video, instant start up times and low buffering ratio, requirements which bring to high demand of bandwidth, whereas CDN aims to minimize costs and they may not meet these high-demanding requirements. Finally, a CDN-based distribution can easily become too costly as the number of streams and viewers increases, due to the high bandwidth costs.

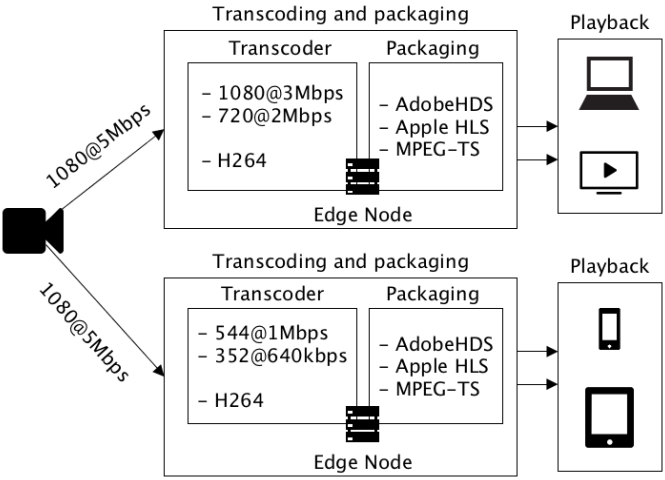

Figure 17: Edge-based platform for live video streaming services.

To face the issues of the approaches described above, the set of innovative technologies introduced by the Edge paradigm may provide the solution needed. Indeed, Edge solutions can be used to tackle various challenges currently faced by media and video streaming applications in order to improve the service quality experienced by end users. A possible Edge-based video streaming platform is depicted in Fig. 17.

With Edge computing, incoming video is first encoded in high quality, either with the camera encoder or with an encoder installed in the closest edge node, and then distributed over a set of edge nodes responsible for both video transcoding and content distribution. In this way, video will be encoded on the fly and delivered to the viewer, removing all the delay introduced by fetching video content from central Cloud. We also assume that there are different transcoder configurations related to different device type (e.g., smartphone, tablet, IPTV, etc.) that will transcode the input video in multiple streams suitable for a given specific device. Therefore, even though edge nodes cannot provide the same computing power of cloud nodes, distribution of individual encoders, each for a specific type of device, makes computational requirements of transcoding oper-

ation suitable for edge nodes.

Overall, the main advantages of this approach are the reduction of end-to-end latency and bandwidth consumption. Indeed, strict proximity of end users to the nodes delivering video content reduces delay due to both a shorter physical distance and a reduced probability to face network congestion, since the number of links and hops along the route is smaller compared to the Cloud- and CDN-based approaches. Moreover, bringing transcoding process to the Edge further contribute to lower delay since video is processed and packaged on that node without the need of fetching missing content from an origin server in a cloud data-center.

5.2 A Scheduling Framework for Edge Computing

Before describing the scheduling framework we devised, we provide a general overview of the system we considered, while reminding that, in the considered scenario, the Edge infrastructure is essentially an extension of a Cloud infrastructure with a set of edge nodes deployed at the edge of the network.

Moreover, these edge nodes are geographically distributed according to the model described in [33], in proximity of multi Radio Access Technology (RAT) base stations (BSs), which provide access to the core network to both user equipment and edge nodes. We assumed that each node is a micro data-center that,

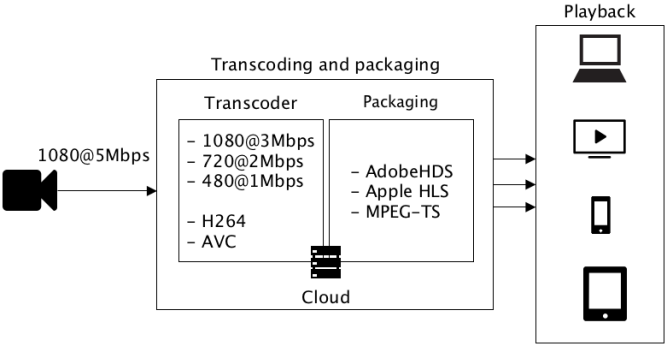

Figure 18: Cloud-based solution for live streaming services.

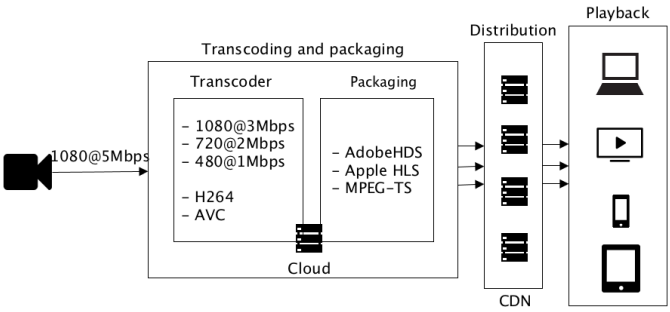

Figure 19: Delivery networks for streaming media contents.

leveraging on virtualization technologies, grants access to its resources by means of virtual machines (VMs). Therefore, each node defines a virtualized environment that allows the deployment of a service instance within an individual VM. Since different services have different computing requirements, in this scenario we assumed that a node can provide different type of VMs. To provide Edge services, the application service provider (ASP) requests to a provider the instantiation of a service in a VM compatible with the service requirements. Since in our work we considered only latency-sensitive services, the main service requirement the provider has to guarantee is the service response time, that is, the time elapsed from a user sending a request and receiving a reply. In this scenario, the network delay and the processing time are the main factors that affect this metric. The former refers to time necessary to a user request from the user device to reach the edge node of the service and it is determined by user-node physical distance, queuing and processing delay of each hop in the network route and the route's available bandwidth. The request processing time is strictly related to the VM and service specifications and refers to the time to elaborate a request. In this context with different VM and service specifications, further analysis of computing performance is needed to select a VM type that can optimize this metric.

Therefore, for the scenario described above, we designed a scheduling framework, summarized in Fig. 20, that takes into account network, bandwidth and computing capabilities of the Edge infrastructure to maximize the service qual-

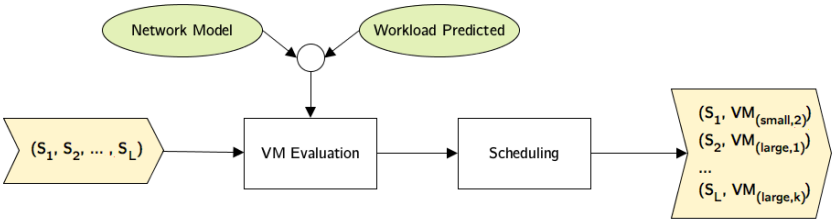

Figure 20: Scheduling framework.

ity experienced by end users of a latency-sensitive application.

5.2.1 Virtual Machine Evaluation

The first component of our scheduling framework performs the evaluation of the available VMs in the edge network. Essentially, it evaluates both network and computational capabilities of the available VM types for each incoming service *s*, by computing a quality score used to determine the eligibility to host *s*. A step-wise description of the devised algorithm is reported in Alg. 1 and more details of the evaluation steps characterizing this process are provided in the rest of this section.

Low-latency path detection

Predicting network delay between users and edge nodes provides useful insights to understand if a given node can meet the network delay requirements of the service to be scheduled, and at runtime to select the communication routes that minimize the delay. Since several monitoring techniques are available for the network latency estimation, for example, by sending probes between nodes and then measuring the latency (i.e., active monitoring) or capturing information about network paths directly from the network devices (i.e., passive monitoring) [81], we assume that network delays between edge nodes can be measured and we use them in our scheduling approach.

We model our edge network as a weighted graph were each node denotes a edge data-center in the network, the link between two nodes represents the actual connection between data-centers and the link weight describes the estimated

network latency between them. We assume, then, that each user connects to the closest BS and we group together users connecting to the same BS (line 2). Therefore, for each user group g we detect the lowest latency path from g to a every node n applying the Dijkstra algorithm on the graph modelling our network, considering as source node the node representing the edge data-center co-located with the BS of the group (line 5). This set of low latency paths will be used for the VM evaluation process described in the next section.

Network and resource evaluation

To identify the most suitable VM to a given service, we defined an evaluation process which computes, for each type of VM, a quality score q_v combining connectivity $q_{n,l}$, bandwidth $q_{n,bw}$ and resource $q_{v,res}$ scores.

The *connectivity score* $q_{n,l} \in [0,1]$ assesses the quality of the connectivity of a VM v by evaluating the quality of the network routes connecting user groups to the node n running v. The input data for this process is the set of low-latency paths computed in Sec. 5.2.1. Therefore, for each network path connecting a given user group g to the n, we evaluate the delay according to the network delay requirements by computing a quality score $q_{n,l,g} \in [0,1]$ using a utility function previously defined by the provider (line 7). The output of this path-based evaluation process is a set of quality scores $\{q_{n,l,g_1}, q_{n,l,g_2}, \ldots, q_{n,l,g_n}\}$, which will be used to compute the connectivity final score $q_{l,n}$ as the mean value of these scores weighted by the number of users belonging to each group (line 9).

The *bandwidth score* $q_{n,bw} \in [0,1]$ assesses the available bandwidth of the node n running the given VM v. The available bandwidth on paths connecting the users to n represents one of the main factors affecting the overall service quality, as already motivated in Sec. 5.1, therefore information about the node capacity can improve the service scheduling. We compute a per path *bandwidth score* $q_{n,bw,g} \in [0,1]$, assessing the quality of the bandwidth of each low latency route in input (line 8). Similarly to the latency evaluation process, we compute the final bandwidth quality score $q_{n,bw}$ as the average of the single path quality scores q_{n,bw,g_i} weighted by the number of users in each group (line 10).

The *VM resource evaluation* is carried out by measuring the service load that a given VM type v can handle and the expected overall service load. Indeed, for latency sensitive applications, underestimating the computational resource nec-

essary for executing the service may increase considerably the service processing time and the overall end-to-end response time. Our approach, then, evaluates the computational resources of a VM v computing a score $q_{v,res} \in [0,1]$ by means of a utility function defined by the provider, comparing the number of user requests that v can handle $\tilde{w}$ with the overall number of requests expected w (line 12).

Finally, the overall *quality score* of a given VM type $q_v \in [0,1]$ is calculated as the harmonic mean of connectivity, bandwidth and resource quality scores (line 13). Despite the fact that the harmonic mean behaves as the arithmetic mean by giving equal weight to both scores when they are similar, it favours the smaller value when the gap between them increases. This ensures that VMs with a very high and a very low score are penalized, which emphasises the need of sufficient network and computational resources. The evaluation output is the set $Q = \{q_{v_1}, \ldots, q_{v_k}\}$ of VM type quality scores that will be used by the scheduler for service deployment (line 14).

5.2.2 Scheduling Approach

The second component of the framework we devised is the scheduler, that is in charge to select, for each service $s \in \mathcal{S}$ received as input, the VM type $v \in \mathcal{V}$ which most likely will fulfil the established latency requirements.

We compute for each v a quality score $q_{s,v}$ using the approach described in Sec. 5.2.1, which is based on the VM computing specifications and the network capabilities of the node hosting that VM. Then, the service instances are scheduled to maximize the overall quality of the selected VMs, guaranteeing an enhanced service quality to end users.

We model the optimisation problem as a binary integer linear programming problem as reported in formulation below. Binary variables $x_{s,v}$ model the placement of a service s on the VM type v, and assume value 1 only when the service is actually scheduled on that VM. The coefficients are the VM quality scores computed by the VM evaluation process previously described, which measures the suitability of the VM type v in hosting the service s. The cost function (I) aims to maximize the quality of the final scheduling, assigning at each service s the most suitable VM.

$$\underset{\bar{x}}{\text{maximize}} \quad \sum_{s\in\mathcal{S}}\sum_{v\in\mathcal{V}} q_{s,v}x_{s,v} \qquad \text{(I)}$$

$$\text{s. to} \quad \sum_{v\in\mathcal{V}} x_{s,v} = 1 \qquad \forall s \quad \text{(II)}$$

$$\sum_{s\in\mathcal{S}} x_{s,v} \leq k_v \qquad \forall v \in \mathcal{V}_n \quad \text{(III)}$$

where

$$x_{s,v} = \begin{cases} 1 & \text{if } s \text{ is scheduled on } v \\ 0 & \text{otherwise} \end{cases}$$

Furthermore, we assume that each service has to be scheduled on a single VM, as expressed in (II), and that the number of times a VM type is assigned to new services cannot exceed the number k_v of available VM instances, defined in (III).

5.3 Validation of our Approach

In this section we present the experiments we carried out to analyse the effectiveness of the proposed framework in terms of service quality experienced by the end users.

5.3.1 Experimental Setup

The experiments evaluate how different deployment solutions affect the live video streaming service time experienced by end users and analyse the impact on its components, namely the network delay and the request processing time, with different number of users joining the video stream in input. A comprehensive description of the different scenarios considered in our experiments is provided in Sec. 5.3.2. We simulated these scenarios using the EdgeCloudSim simulator [64]. This simulator extends the CloudSim toolkit [11] providing extra functionalities granting the modelling of networking resources and edge nodes, allowing a accurate simulation of real edge infrastructures. The results obtained, not only show the benefit of the Edge platform in terms service time, but denote the need of a Edge-specific scheduling solution to obtain optimal results.

Algorithm 1: Latency-Sensitive Scheduling

Data: Service s to be scheduled
Data: Latency constraint t requested by the service provider
Data: Users' locations $U = \{u_1, u_2, \ldots, u_m\}$
Data: Nodes $N = \{n_1, n_2, \ldots, n_k\}$
Data: Nodes' location $L = \{l_1, l_2, \ldots, l_k\}$
Data: VM types $VM = \{v_{i,1}, v_{i,2}, \ldots, v_{i,q}, \forall i \in N\}$
Data: Array Q of quality scores for each VM $v \in VM$
Result: The vm_s scheduled for s.

begin
  Define a set of users' groups $G = \{g_i, |u_j - l_i| < \varepsilon \quad \forall u_j \in U, \forall l_i \in L\}$;
  forall the $n \in N$ **do**
    forall the $g \in G$ **do**
      Estimate the network path $p_{g,n}$ with the lowest latency $\tilde{l}_{g,n}$ between g and n;
      Estimate the available bandwidth $\tilde{bw}_{g,n}$ between users in g and the node n and the required bandwidth bw_g on $p_{g,n}$;
      Compute the latency score $q_{n,l,g} = \text{LatencyScore}(t, \tilde{l}_{g,n})$;
      Compute the bandwidth score $q_{n,bw,g} = \text{BandwidthScore}(bw_g, \tilde{bw}_{g,n})$;
    $q_{n,l} = \frac{\sum_{i=1}^{n} |g_i| q_{n,l,g_i}}{\sum_{i=1}^{n} |g_i|}$;
    $q_{n,bw} = \frac{\sum_{i=1}^{n} |g_i| q_{n,bw,g_i}}{\sum_{i=1}^{n} |g_i|}$;
    forall the $v \in n$ **do**
      $q_{v,res} = \text{ComputingScore}(w, \tilde{w}_v)$;
      Compute the q_v quality score $\mathsf{Q}[q_v] = \frac{3}{\frac{1}{q_{v,res}} + \frac{1}{q_{n,l}} + \frac{1}{q_{n,bw}}}$
  $vm_s = Scheduler(\mathsf{Q})$;
  return vm_s;

5.3.2 Scenarios

We defined a set of simulation scenarios where we analysed the impact of different deployment solutions and scheduling policies on the service quality experienced by end users, in the context of live video streaming services. In the remainder of this section we provide a detailed presentation of the different network designs and scheduling approaches in each scenario devised.

Cloud. We consider a centralized deployment solution where both video processing and content distribution processes are carried out on a cloud data center. Essentially, given an incoming video to be streamed, all the encoding/transcoding operations are executed on a cloud data center. User requests to access the live stream are also directly forwarded to the cloud server responsible of stream distribution.

We modelled cloud resources as set of infinite VM instances whose specifications are taken from the Amazon m2.4xlarge and are reported in Tab. 9. The users-data center connection has been defined as a single link, whose capacity has been fixed at 1000 Mbps, representing the aggregation of single links connecting user access BSs to the cloud. We defined also a communication delay $\delta_{u,c} = 0.09$, which models the delay related to the queuing and processing operations and the physical distance characterizing the network route between the user u and the cloud data center c. This value has been estimated by averaging latency of ICMP requests between hosts in Europe and the Amazon Web Service instances in the same time zone.

Content delivery network (CDN). We defined a two-tier network architecture, characterized by an origin server in a massive cloud data-center and 10 geographically distributed replica servers. In this scenario the encoding/transcoding operations are executed in the cloud servers, while the content is distributed among the replica nodes. Users are randomly distributed in proximity of all replica nodes and the content distribution follows the schema designed in [51]. Therefore, a user request is first redirected to the closest replica server and then, if the content is already cached there, it is directly returned. Otherwise, the request is forwarded to the cloud server to fetch the content.

In this scenario origin and replica servers have different purposes and then different hardware configurations. We assumed that CDN (replica) nodes are

Algorithm 2: FIXED provisioning algorithm

Data: Set of QoS requirements offered by the ASP: $S = \{QoS(x), x \geq 0\}$
Data: Estimated mean inter-arrival times: $\frac{1}{\lambda}$
Data: Estimated mean requests durations: $\frac{1}{\mu}$
Result: Number of VMs to acquire: V

begin
  calculate the smallest number of V such that $\frac{\lambda}{V\mu} \leq 1$;
  $t = \min(QoS(x)), QoS(x) \in S$;
  $P(w(r_i) > t) = 1$;
  $\rho = \frac{\lambda}{V\mu}$;
  while $P(w(r_i) > t) > p$ **do**
    calculate $\Pi_W = \frac{(V\rho)^V}{V!}((1-\rho)\sum_{n=0}^{V-1}\frac{(V\rho)^n}{n!} + \frac{(V\rho)^V}{V!})^{(-1)}$;
    calculate $P(w(r_i) > t) = \Pi_W e^{(-V\mu(1-\rho))^t}$;
    if $P(w(r_i) > t) > p$ **then**
      $V = V + 1$
  return V

small data centers located only in strategic points, such as ISP point of presences (PoPs) or at internet exchange points (IXPs). CDN servers' access bandwidths are distributed as a Pareto distribution with mean value $\mu = 500$ and each connection user-server is characterized by communication delay $\delta_{v,s} = 0.013$s. Similar to the Cloud scenario, $\delta_{v,s}$ measures the expected delay due to the physical characteristics (e.g., number of hops and distance) of the route between the host and the closest CDN server. We modelled the connection between CDN servers and the cloud origin server as a high capacity link with an average bandwidth of $\mu = 750$Mbps, since we assumed that they are directly connected to the ISP backbone. We also defined a link communication delay as $\delta_{e,c} = 0.03$s, modelling the delay along the path from the CDN and the origin server based on the number of hops and the physical distance between them. Moreover, since the main purpose of CDN networks is to deliver content, the VM instances used in this scenario are storage optimized. Therefore we used the Amazon i3.large specifications, as reported in Tab. 9.

Edge. In this scenario, according to the Edge-based deployment solution described in Sec. 5.1 the service is entirely deployed on the edge nodes. Therefore, both encoding/transcoding and distribution operations are executed on the edge nodes, whereas the Cloud has only management functionalities.

The designed network infrastructure is composed of 20 edge nodes, each colocate with a BS. The computing power of each node is rather limited. Each node provides 2 types of VMs, namely the Amazon m1.large and m1.xlarge instance types, but, due to limited physical resources, only 10 instances that can actually instantiated on each node. The access bandwidth of each edge node has been modelled instead as a Pareto distribution with average value $\mu = 375$Mbps. In this scenario we also assume that the distance between edge nodes is small, in the order of 20km, allowing the deployment of high speed inter-node connections through either dedicated links or single-hop connections. Thus, we modelled the inter-node bandwidth whose capacity is distributed also as a Pareto distribution with mean value $\mu = 400$Mbps. The communication delay between the nodes i and j has been modelled, instead, by mean of a uniform distribution $\mathcal{U}[0.006, 0.009]$.

In this scenario, we assume that users can access the stream video with 3 different type of devices, namely smartphone, laptop and tablet. Therefore, according to the Edge-based service deployment described in Sec. 5.1, we assume that 3 different and lightweight streaming engines (i.e., packages responsible of encoding/transcoding operations and content distribution) have to be deployed, each one for a specific device. Each instance is responsible for transcoding the video input into multiple bit rates and resolutions suitable for a given device. We used two different approaches for the scheduling of the instances, that is Cloud-based and Edge-based, described below.

In the Edge-based scenario, we schedule the services using the approach presented in Sec. 5.2.2. For the VM evaluation process, described in Sec. 5.2.1, we defined three utility functions as in Eq. 5.1, Eq. 5.2 and Eq. 5.3 for the evaluation of the network delay, the available bandwidth and VM resource respectively.

$$u_{v,\delta}(\delta_{g,v}, \tilde{\delta}) = S\left(1 - \frac{\delta_{g,v}}{\tilde{\delta}}\right) \tag{5.1}$$

$$u_{v,B}(B_{g,v},\tilde{B}) = S\left(\frac{B_{g,v}}{\tilde{B}} - 1\right) \tag{5.2}$$

$$u_{v,RPS}(RPS_v,\tilde{W}) = S\left(\frac{RPS_v}{\tilde{W}} - 1\right) \tag{5.3}$$

For the evaluation of the network route delay between a user group g and a VM v, we compute a utility score using a sigmoid function S, that takes in input the route delay $\delta_{g,v}$ and the delay requirement $\tilde{\delta} = 50ms$. Therefore, the utility function in Eq. 5.1 evaluates the margin between the actual delay $\delta_{g,v}$ and the requirements $\tilde{\delta}$ returning a score as close to 1 as the current delay is significantly lower than the requirements. Otherwise, low values close to 0 are returned. Bandwidth and resource evaluation follow the same approach defined for the delay evaluation. Essentially, given the predicted available bandwidth $B_{g,v}$ on the network route from g to v the utility function returns a value as close to 1 as the value of $B_{g,v}$ is higher then the needed bandwidth $\tilde{B}$ due to the number of users in g. The resource evaluation is achieved, instead, by comparing the computing capability of v expressed in terms of request per second *RPS* with the expected number of requests $\tilde{W}$ generated by the expected workload. Finally, the scheduling is defined by the scheduling approach in Sec. 5.2.2.

In the Cloud-based approach, instead, we adopted a Cloud scheduling approach [26] that estimates the number of streaming engines instances needed to minimize the user waiting time. This approach aims to minimize the processing time deploying a number of instances based on the expected workload. The Alg. 2 shows the pseudo-code of their approach. They define a M/M/V[5] queuing model to determine the number of VMs to be instantiated in advance to guarantee a user waiting time compliant with the requirements. Three inputs are required: (i) the estimated mean inter-arrival time of requests $1/\lambda$, (ii) the estimated mean duration of requests $1/\mu$ and (iii) and the target probability p that a request has to wait more than its QoS requirement. The first step of the algorithm (line 2) computes the initial number of VMs V, such that the system utilization is lower than one. Then, the smallest waiting time t to be guaranteed by the service provider is determined (line 3). Afterwards, the algorithm updates the number of VMs to be instantiated until the probability that a new service request $w(r_i)$ has to wait more than t ($P(w(r_i) > t)$) is lower than the threshold defined by p. Finally, for each streaming engine type the number of replica is

Table 9: Virtual machines specifications.

Specs	m1.large	m1.xlarge	m2.4xlarge	i3.large
CPUs	4	8	26	2
CPU MIPS	2400	4800	20000	2400
RAM(GB)	8	16	70	15
Storage(GB)	2x420	4x420	2x840	unlimited
Price (USD/h)	0.17	0.35	0.98	0.15

computed and then randomly distributed among the edge nodes.

Similarly to the CDN scenario, also in this scenario, users are randomly spread in proximity of all edge nodes. Therefore, a user request is first sent to the BS co-located to the closest edge node and then forwarded to the node containing the VM which hosts the encoder related to the type of device user adopted to access the video stream.

5.3.3 Experimental Results

The results in Fig. 21(a) show that deploying a service on the Edge allows to achieve a considerable network delay reduction with respect to all the others deployment solutions. Highest reduction is around 4.5-fold with respect to cloud, and lowest is more than 2-fold with respect to CDN. Moreover, this network delay reduction is not guaranteed by only the platform itself and a suitable service scheduling algorithm is necessary as shown in Fig. 22 (a). Indeed, the adoption of scheduling algorithms that do not take into account the joint information of network conditions, user requirements and workload only partially exploit the advantages introduced by the edge infrastructure, resulting in suboptimal results. Additionally, processing time represents another critical factor for the Edge platform, as depicted in Fig. 21 (b) and in Fig. 22 (b). The values plotted represent the time spent by the streaming engines to package the video content to be delivered. These results show how in the Edge scenario we obtain the highest processing time due to the limited computational resources provided by the edge nodes. Obviously the processing time on the Cloud scenario is the lowest among the three due to the high computing power characterizing cloud resources. In the CDN scenario, instead, we obtain smaller values then the Edge scenario, since

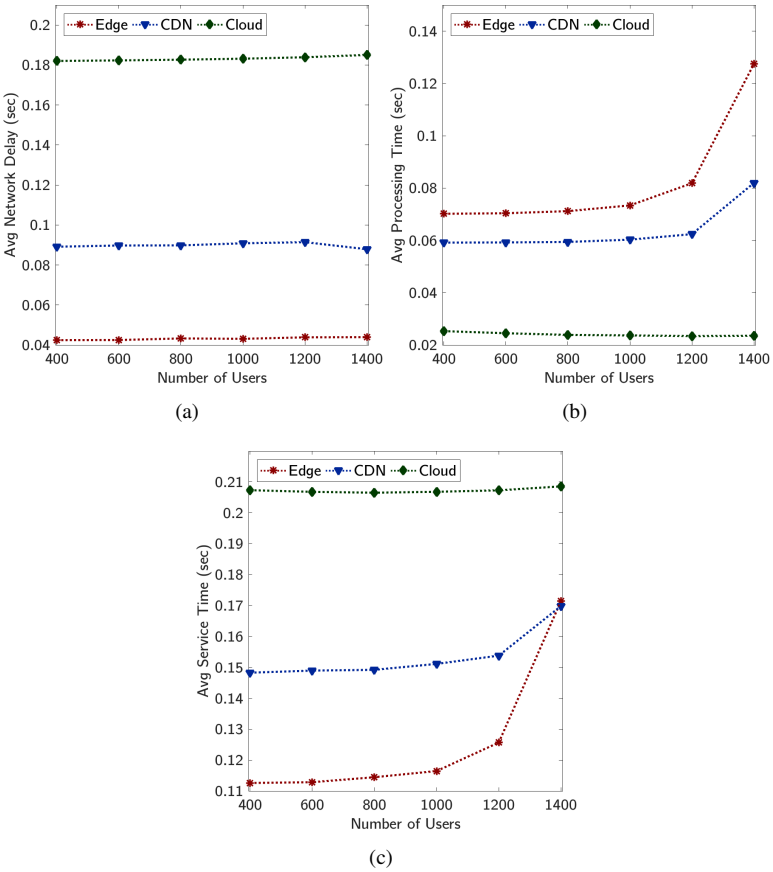

Figure 21: Performance analysis of Cloud, CDN and Edge scenarios.

distributing the requests on the multiple replica server avoids the server overloading. Moreover, the limited edge node resources not only already provide the highest processing time, but they may become the system bottleneck, as the load increases. Edge results in Fig. 21 (b) and Fig. 22 (b) describing the processing time obtained using our approach, denote this increasing trend according to the growth of the number of input devices. Essentially, in our approach where each streaming engine instance has to process all the incoming requests from the associated user device, we experience a faster raise of the processing time.

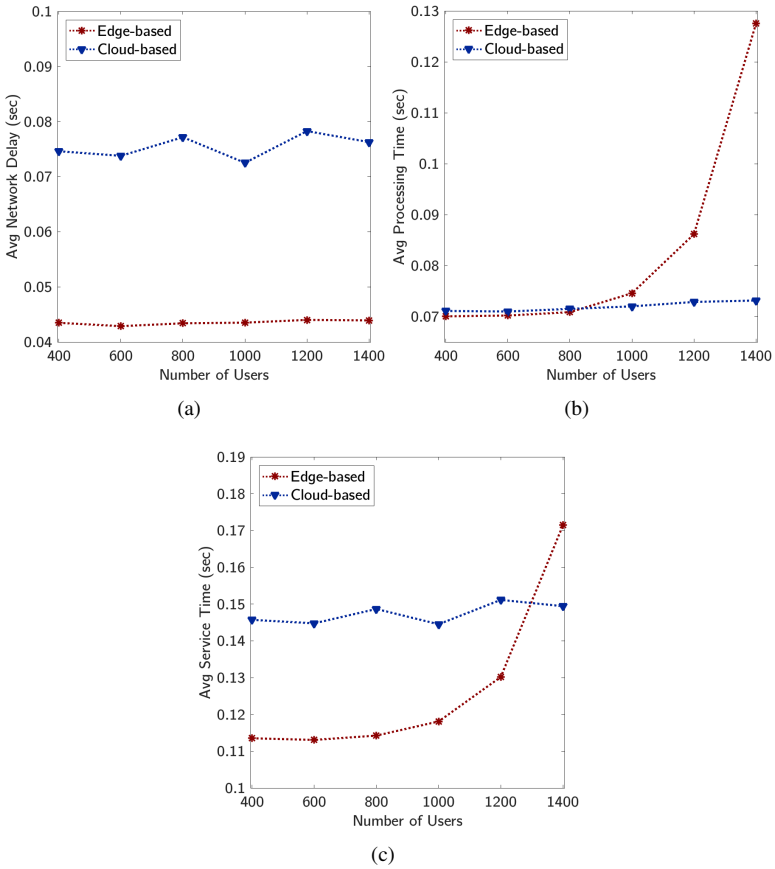

Figure 22: Performance comparison of Cloud and Edge scheduling approaches.

Therefore, to reduce this high processing time in the Edge scenario, provider can horizontally/vertically scale the servers in the edge data centers or create new edge data centers in the overloaded locations. Results in Fig. 22 (b) confirm this assumption showing that predicting in advance, the number of VM instances for each streaming engine enhances the performance, reducing the average processing time needed to elaborate each user request. However, due to the significant network delay reduction provided by the edge-based solution, higher process-

ing time does not affect the overall service time, if the processing capacity of the servers is not saturated, as depicted in Fig. 21 (c) and in Fig. 22 (c). For very high number of users, instead, higher processing time of Edge introduces a system bottleneck that considerably affects the final service time experienced by end users (Fig. 22 (c)). Nevertheless, results in Fig. 22 (c) confirm the need for scheduling solutions that take Edge specific features into account and for an infrastructure with processing capacity compatible with number of users in the edge data centers to obtain good performance in Edge scenario.

Chapter 6

Related Works

In the area of Cloud Computing many works have already been carried out for the development of frameworks for the management of service quality throughout the whole service life-cycle [5, 6, 15, 17, 77]. However, they are mainly focused on the definition of theoretical technical frameworks without providing any real implementation of the methods actually responsible of SLA management tasks.

Nonetheless, many studies have been accomplished for the development of point wise solutions dealing with the challenges introduced by each stage in the service life-cycle.

In the remainder of this section we provide a comprehensive overview of the most relevant works related to SLA specification languages, negotiation and an analysis of the solutions currently adopted for service scheduling in Cloud.

6.1 SLA Definition Languages

To provide an overview of the most important languages currently adopted, we extend the study of [40] to cover additional languages and new aspects of the already evaluated ones. Below we briefly describe the evaluation criteria. The results of our evaluations are summarised in Tab. 10 and show the advantage of SLAC over other formalisms. Important components of the Cloud domain like dynamicity, multi-party agreements, and brokerage are supported only by

SLAC, and the supporting software framework allows an easy deployment of SLAC SLAs in a wide-range of real-world scenarios.

For our assessment, we consider the following formalisms: WSLA [37], WS-Agreement [3], WSOL [66], RBSLA [50], Linked USDL (LUA for short) [52], SLALOM [16], SLAng [63], SLA* [38], CSLA[58] and, of course, SLAC. We evaluate them by first considering *General* features of the languages and then assessing their impact on the different phases of SLAs lifecycle. In the table, we use yes ✓ for indicating that the feature is fully supported, ✸ for indicating that the feature is partially supported, and ✗ for indicating that the feature is not supported.

General considers the evaluation of different aspects that we list below:

- *Cloud Domain* considers whether a SLA language has been specifically designed for Cloud;
- *Service Models* evaluates whether all service models (including specific vocabularies) are directly supported or, in some cases, extensions are needed;
- *Formalisation* refers to the level of formality in the definition of syntax and semantics of the language;
- *Dynamicity* considers the capacity to express possible changes of the terms of agreement at runtime.
- *Confidence or Fuzziness* is the capacity to deal with QoS uncertainty, with confidence defining the percentage of compliance of clauses, and fuzziness referring to an acceptable interval around the threshold of a metric; *Reusability* refers to the possibility of reusing constructs defined in a template or in a SLA across different SLAs;
- *Composability* is the ability to express composite SLAs;
- *Extensibility* evaluates whether the language terms and metrics can be extended;
- *All parties* considers whether it is possible to describe all parties involved in the service provision and in all actions (monitoring, verification, provision, etc.);

- *Price Model* is the level of coverage of price schemes and of computation model;
- *Consistency check* considers whether consistency of SLAs is verified; we write ✸ if only syntactic checks are offered, ✓ if also semantic aspects are considered, and ✗ if no check is performed.

Definition, Discovery and Negotiation considers the possibility of using:

- *Editor* for writing SLAs, they can be generic (✸), domain-specific(✓) or absent (✗);
- *Broker* may have different levels of support when taking decisions;
- *Metric Definition* refers to the possibility offered for defining quality metrics;
- *Alternatives* evaluates the ability to specify alternative levels of service;
- *Soft Constraints* is concerned with the use of soft-constraints to address over-constrained requirements;
- *Matchmaking Metric* enables the specification of how to match equivalent metrics or metric units;
- *Negotiability* is the ability to indicate how, and to which extent, quality terms are negotiable.

Deployment and Monitoring considers the offering of:

- *Metric Schedule* indicates how often the terms of agreements are measured;
- *Metric Provider* indicates the possibility of specifying the party responsible for monitoring each term of agreement;
- *Automatic Deployer* refers to the provision of tools for automatic deployment of the service;
- *Integrated Monitoring* is concerned with the capacity to automatically configure the monitoring system relying on the SLA specification.

Billing and Penalty Enforcement considers the presence of:

- *Penalties* and *Rewards* to be enforced under specified conditions. In this case, ✓ stands for fine-grain support (at the level of service level objectives), ✸ for coarse-grain, and ✗ for no support.
- *Actions* triggered by contracts violations;
- *Conditions Evaluator* refers to the possibility of specifying the party in charge of auditing each term;
- *Assessment Scheduler* specifies when each term is assessed.

Termination considers the

- *Automatic Undeployment* of the used resources.

Phase	Criteria	WSLA	WS-A	WSOL	RBSLA	LUA	SLALOM	SLAng	SLA*	CSLA	**SLAC**
General	Cloud Domain	✗	✗	✗	✗	✗	✗	✗	✓	✓	✓
	Service Models	✸	✸	✸	✸	✸	✸	✸	✸	✓	✸
	Formalisation	✗	✗	✗	✸	✸	✸	✓	✓	✸	✓
	Dynamicity	✗	✗	✗	✸	✗	✗	✗	✗	✗	✓
	Confidence or Fuzziness	✗	✗	✗	✗	✗	✗	✗	✗	✓	✗
	Reusability	✓	✓	✓	✓	✓	✓	✗	✗	✗	✓
	Composability	✗	✸	✗	✗	✗	✗	✓	✓	✸	✓
	Extensibility	✓	✓	✓	✗	✗	✗	✗	✓	✗	✓
	All Parties	✗	✗	✗	✗	✗	✗	✗	✗	✗	✓
	Price Model	✗	✗	✗	✗	✓	✗	✸	✓	✸	✓
	Consistency check	✸	✸	✸	✓	✓	✸	✓	✓	✗	✓
Definition Disc./Nego.	Editor	✸	✸	✸	✸	✸	✗	✗	✓	✓	✓
	Broker	✗	✸	✗	✗	✗	✗	✗	✗	✗	✓
	Metric Definition	✓	✗	✗	✓	✗	✗	✗	✓	✗	✓
	Alternatives	✸	✸	✸	✸	✗	✗	✗	✗	✗	✓
	Soft Constraints	✗	✓	✗	✗	✗	✗	✗	✗	✗	✓
	Matchmaking Metric	✗	✗	✗	✗	✗	✗	✗	✓	✗	✓
	Negotiability	✗	✸	✗	✗	✗	✗	✗	✓	✗	✓
Deploy. Mon.	Metric Schedule	✓	✗	✗	✓	✓	✗	✓	✓	✓	✓
	Metric Provider	✓	✗	✓	✗	✓	✗	✗	✓	✗	✓
	Automatic Deployer	✓	✗	✗	✓	✓	✗	✗	✸	✗	✓
	Integrate Monitoring	✓	✗	✗	✓	✓	✗	✗	✓	✗	✓
Billing and Penalty Enfor.	Penalties	✗	✓	✸	✸	✓	✓	✓	✓	✓	✓
	Rewards	✗	✓	✗	✸	✓	✗	✗	✗	✗	✓
	Actions	✓	✗	✗	✓	✗	✗	✓	✓	✗	✓
	Condition Evaluator	✓	✗	✓	✗	✓	✓	✗	✗	✓	✓
	Assessment Schedule	✸	✸	✸	✸	✗	✗	✓	✓	✗	✓
Ter.	Automatic Undeployment	✗	✗	✗	✗	✗	✗	✗	✸	✗	✓

6.2 SLA Negotiation

Still the in area of SLA management many works have been carried out for the autonomic negotiation of SLAs. Many of them are mainly based on the approach adopted for the negotiation of traditional contracts.

Indeed, they are mainly divided into: game theoretic models, heuristic and argumentation-based methods [36]. Here, we focus on heuristic approaches since, even for static contracts: (i) the game theoretic techniques assume that negotiators have full knowledge of peers' preferences [36], which is not suitable for Cloud due to the large number of participants, SLA terms and strategies [48]; (ii) applying the argumentation-based approaches in automated negotiation for cloud is challenging [32] and few works actually employ it [12].

Heuristic approaches define protocols and strategies to look for good solutions, which do not necessarily correspond to the optimal one [36] or do satisfy all the requirements of the involved parties. They are based on counter-offer generation where each party decides, according to some heuristic, to accept-reject the offer or to create counter-proposal. The core element of this approach is the agents' decision-making process which defines the offer's evaluation and the counteroffer generation. Two mechanism are usually adopted for decision-making: *trade-off* and *concession*. Essentially, the trade-off mechanism (used in [18, 25, 78, 85]) defines counterproposals similar to the offer in terms of negotiator's utility, where multiple negotiation terms are traded-off to fit better the opponent's requirements [27]. The concession approach (for example, [13, 15, 27, 42, 80]) defines an offer generation mechanisms where, at each negotiation step, the negotiator is willing to decrease the overall utility score of the agreement. This is done by decreasing the utility value of one or more terms using different functions [48]: (i) time-dependent, which compute the next value to be offered according the remaining negotiation time, (ii) resource-dependent, which consider the available resource during the negotiation process as main parameter; and (iii) behaviour-dependent negotiation functions, which formulate the new offer based on the previous opponent's attitude [29].

Nevertheless, the *trade-off* and *concession* approaches cannot be used for smart contracts, since applying them implies the execution of the counteroffer generation mechanism in each possible state of the contract. Therefore, consid-

ering that both provider's and consumer's contract may have thousands of states, it becomes quickly infeasible.

6.3 Service Scheduling in Cloud

Also in the area related to service management, many works have been developed which provide effective solutions for SLA-aware service deployment. An example is given by scheduling approaches which aim to optmize specific metrics defined within the SLA.

For instance, many works have been developed for the scheduling of latency-sensitive services, that are characterized by more strict requirements on the service-user latency.

In the area of distributed Cloud Papagianni et al. [49], proposed a framework for an efficient mapping of VM user requests on the aggregate set of connected clouds. They modelled the mapping problem as mixed integer programming aiming to minimize the mapping costs and the number of hops among the VMs. Similarly, Aral et al. in [4] proposed a novel approach for the problem of mapping virtual networks defined by the set of interconnected VMs (i.e., service replicas) on a real infrastructure in a distributed cloud. The solution devised is relies on a topology based mapping approach and allocates the virtual network defined by the connections among service components on a cloud subnet whose topology is isomorphic to the virtual one. They aim to reduce the network latency and optimize the bandwidth utilization. Still in the area of distributed Cloud, in [39] the authors propose a novel approach to tackle the problem of service allocation in a federated Cloud environment for horizontally scalable services. They defined a method which allocates service component replicas taking into account the maximum service requirements in terms of computing, networking and storage resources, a set of affinity and anti-affinity rules for deploying replica in the same node or subnet and the federated infrastructure costs. The allocation problem is then formulated as Mixed-Integer Linear Programming optimization problem and implemented, then, a heuristic solver that yields to near-optimal solutions in very short time. Pittaras et al. [54], instead, developed an approach for efficient mapping of virtual networks onto a multi-domain network substrate. They devised a semantic based approach for the mapping of

requests to real network subnets which minimizes the number of hops between selected nodes.

In the context of single Cloud providers, authors in [53, 82] developed service component placement methodologies that, similarly to the distributed scenario described before, optimize the service placement among servers in a single data center minimizing the transfer time between service components. Therefore, also these approaches do not take into account users information which would make their application suitable for service scheduling in Edge. Authors in [26], however, integrated user information in the service allocation methodology they proposed. Indeed, they defined a provisioning algorithm based on queuing theory to identify the number of VMs to be deployed in order to minimize the user waiting time. However, this approach, intended for IaaS Cloud, only define the number of VMs needed to cope with the incoming load and is still missing VM placement policies which would lead to sub-optimal results in the case of Edge Computing.

In the area of Edge Computing, instead, no works are currently available, to the best of our knowledge, for service scheduling on edge nodes. Works as [31, 44, 84] face edge computation offloading problem, that is the decision of scheduling a task on mobile device or local/internet Cloud. Nevertheless, these works do not take into account the actual service scheduling between edge nodes.

In the area of Fog Computing, in [62], authors defined a scheduling approach for the placement of service modules on fog nodes, focusing only on the optimization of response time among components, without taking into account the end-to-end service time. Aazam et al. in [1], instead, defined a resource estimation and pricing model for IoT, that estimates the amount of resources to be allocated for a given service, again without providing an solution for the selection of a node where to allocate the service.

Chapter 7

Conclusions

Service quality in Cloud Computing has a considerable impact on the whole service life cycle. For example, given a set of functionally equivalent services, during the service acquisition phase, the consumer choice is mainly driven by the quality guarantees and terms of agreements they offer. Moreover, the violation, during the service execution, of previously agreed terms certainly affects the trustworthiness of the consumers, that might ask the termination of the service execution or decide to not renew the service provision once it is terminated.

Therefore, given the relevance of this area for the success of Cloud paradigm, many works have already been developed in the context of service quality specification and management. However, in the devised solutions, the dynamism characterizing cloud services is not properly taken into account, even though it represents one of the distinguishing feature of these services.

We have proposed a framework which addresses service quality in the first phases of the service life cycle, namely specification, negotiation and service deployment.

Specifically, for the service acquisition phase we have defined (i) SLAC, a novel specification language, that allows the definition of dynamic agreements and (ii) a negotiation framework for the negotiation of the SLAs formulated with the language we devised.

The SLAC language, presented in Chap. 3, introduces the definition of *dynamic SLAs* allowing the modification of the quality of service terms at run time

when specific conditions are met, guaranteeing the adaptation of service performance. In this way, consumers and providers can associate service quality term values to specific service conditions, that can trigger automatic modification of those values when these conditions are reached. The presented experimental results show the advantages of the dynamism offered by SLAC by evidencing a reduction of agreements violations and thus of penalties as well as an increase of the revenue for the provider.

Nevertheless, the adoption of this new specification language introduces new research challenges related to negotiation of the dynamic SLAs. Therefore, we developed a novel framework (Chap. 4) for the negotiation of this new type of agreements. It provides insights for the modification to be applied to service offers and request to increase their compatibility and, consequently, to find a final agreement between the parties.

To this aim, we proposed a formal framework for the specification of this interaction and a framework for the autonomous negotiation of smart contracts. The main steps of the solution we devised are checking of specification consistency, analysing the compatibility between offers and requests and elaborating the best possible agreement (if any). The outcome of the experimental evaluation shows that if the parties are willing to accept modifications of their initial proposals, the number of reached agreements improves considerably.

We also addressed the issues related to the scheduling of latency sensitive applications by defining a novel scheduling algorithm aiming at guaranteeing the latency requirements in a Edge scenario.

The main factors that lead us to address this last challenge were the facts that latency-sensitive services are highly dynamic and thus a good test bed for the application of our service specification and negotiation framework; and currently there is no available solutions for addressing the scheduling problem in Edge Computing, which represents the most suitable deployment solution for this type of services.

Indeed, the latency characterizing Cloud platforms, cannot cope with the latency requirements of these services, and thus the currently adopted cloud-based deployment solutions for these services turn out to not fully suitable. The Edge paradigm, instead, extends the cloud infrastructure by adding computing resources in close proximity of end users and considerably reduces the end-to-

end latency. Moreover, the considered scenario represents a good use case for the application of the solutions for service specification and negotiation; its high dynamism can benefit from the specification of dynamic requirements.

In Chap. 5, we defined a two stages score-based algorithm that, given a set of available Virtual Machines (VMs), first evaluates the eligibility of each VM type to host a given service, assigning to each VM type a quality score, and then schedules the services in order to maximize the total score of the chosen VMs, while guaranteeing higher service quality for end users. To evaluate our approach we evaluated the impact of different deployment solution, namely Edge, CDN, and Cloud. The obtained results suggest that Edge is the best solution in this context. Moreover, to validate our framework, we compared average response time, processing time and network delay experienced by the users of our approach with similar work by other researchers and showed that our solution offers much better performances.

7.1 Research Findings

Research Question 1

How to properly describe service quality terms in Cloud Computing?

The needs of both cloud computing consumers and providers are service-driven, therefore when specific conditions are met, the service level objects need to be adapted accordingly. We propose a novel framework that allows the specification of service level agreements that permits adapting the terms at run time, according to the conditions defined. The results we obtained demonstrate the advantages of this autonomous approach in reducing services violations, and in guaranteeing enhanced quality of service for final user as well as higher revenues for the service provider.

Research Question 2

How to negotiate SLAs for Cloud Computing with the new dynamic language?

One of the main challenges introduced by the use of dynamic SLAs is represented by the negotiation process of offer/requests. Therefore we devised a ne-

gotiation framework, which autonomously executes the compatibility analysis of the parties' proposals and seeks for modifications to be applied to both of them, to increase the compatibility and to eventually sign a final agreement. Experimental results show that with our approach, if parties are flexible in accepting modifications of their original proposals, a valid agreement can be reached.

Research Question 3

How to enhance specific service quality in distributed environments?

In this work we focused on Edge computing, a highly-distributed environment, to enhance the quality latency sensitive services. Specifically, we defined an innovative scheduling algorithm that aims at maximizing service quality experienced by end users and considers in the scheduling process the Edge computing characteristics, such as, latency, bandwidth, and computational capabilities. The experimental results show that Edge-based deployment solutions offer good performance in terms of end-to-end latency. However, they also show that, for effective exploitation, scheduling algorithms specifically devised for this platform are needed.

7.2 Limitations of our Study

In this section we summarize the main limitations of the work presented in this thesis:

- Our final goal was to provide a framework for service quality management in Cloud Computing focusing on the improvement of service quality specification and on the development of novel techniques for service management. However, while we provided a comprehensive framework for SLA specification and negotiation, we addressed only partially the management phase. Indeed, we developed a solution for the initial deployment of the services in Edge that does not consider the possible dynamic requirements expressed with SLAC in the decision process.

- The SLAC language we devised, defines a formalism for enhanced spec-

ifications, however it introduces new challenges in service discovery, negotiation, scheduling and management processes that are only partially addressed in this thesis. We mainly focused on the discovery and negotiation processes, whereas the analysis on how scheduling and management tasks are affected by dynamic SLAs has not been carried out yet.

- The decision model devised in the negotiation framework does not allow parties to define a customized policy to be used during the negotiation process. Currently, the process is only based on the compatibility of SLA states and transitions and it does not give the possibilities to the parties to weight differently the states and the transitions during compatibility analysis. Moreover, costs and revenues of states and transitions are not considered in the decision model.

- The service scheduling solution we developed is only responsible for the initial placement of service instances on edge nodes and does not provide a run time support for the service management during the service provision.

7.3 Future Works

In the previous section we outlined some limitations of our work which automatically suggest new challenges and further research topics:

- The scheduling approach we developed does not take into account dynamic requirements specified with SLAC. Therefore, to fully exploit the advantages introduced by this new formalism, we plan to extend our solution considering also the dynamic requirements in the decision process.

- The information provided by the conditions and modifications expressed with the dynamic agreements provide useful information that can be used also for service and resource management processes. We plan also to define innovative management processes that take into account dynamic requirements in the decision tasks.

- The module for decision making of our scheduling component can be improved for estimating also the number of service instances needed to handle the expected workload. Currently we only considers one instance per

service to be deployed, and rely on the (limited) scaling capabilities of edge nodes for the activation of new instances.

- Besides new research challenges, there are also technical aspects to be improved. Indeed, the SLA verification module, defined in the negotiation framework, is currently based on a brute force approach. We plan to study alternative approaches based, e.g., on genetic algorithms.

References

[1] Mohammad Aazam and Eui-Nam Huh. Fog computing micro datacenter based dynamic resource estimation and pricing model for iot. In *Advanced Information Networking and Applications (AINA), 2015 IEEE 29th International Conference on*, pages 687–694. IEEE, 2015. 85

[2] Smart Contracts Alliance. Smart contracts: 12 use cases for business and beyond. *Chamber of Digital Commerce*, page 56, 2016. 18

[3] Alain Andrieux, Karl Czajkowski, Asit Dan, Kate Keahey, Heiko Ludwig, Toshiyuki Nakata, Jim Pruyne, John Rofrano, Steve Tuecke, and Ming Xu. Web Services Agreement Specification (WS-Agreement). Technical report, Open Grid Forum, 2007. 79

[4] Atakan Aral and Tolga Ovatman. Network-aware embedding of virtual machine clusters onto federated cloud infrastructure. *Journal of Systems and Software*, 120:89–104, 2016. 84

[5] Elarbi Badidi. A framework for software-as-a-service selection and provisioning. *International Journal of Computer Networks & Communications*, 5(3):189, 2013. 78

[6] Elarbi Badidi. A broker-based framework for integrated sla-aware saas provisioning. *arXiv preprint arXiv:1605.02432*, 2016. 19, 78

[7] Eric Bauer and Randee Adams. *Service quality of cloud-based applications*. John Wiley, 2013. 2, 15

[8] Kashif Bilal and Aiman Erbad. Edge computing for interactive media and video streaming. In *Fog and Mobile Edge Computing (FMEC), 2017 Second International Conference on*, pages 68–73. IEEE, 2017. 61

[9] Leo Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001. 33

[10] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599–616, 2009. 1, 9

[11] Rodrigo Calheiros, Rajiv Ranjan, Anton Beloglazov, César De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and Experience*, 41(1), 2011. 68

[12] Jen-Hsiang Chen, Fahmida Abedin, Kuo-Ming Chao, Nick Godwin, Yinsheng Li, and Chen-Fang Tsai. A hybrid model for cloud providers and consumers to agree on qos of cloud services. *Future Generation Computer Systems*, 50:38–48, 2015. 83

[13] Robert Coehoorn and Nicholas Jennings. Learning on opponent's preferences to make effective multi-issue negotiation trade-offs. In *Proc. of the 6th Intl Conference on Electronic commerce*, pages 59–68. ACM, 2004. 83

[14] Fog Computing. The internet of things: Extend the cloud to where the things are. *Available on: http://www. cisco. com/c/dam/en_us/solutions/trends/iot/docs/computingoverview. pdf*, 2015. 12

[15] Marco Comuzzi and Barbara Pernici. An architecture for flexible web service qos negotiation. In *Proc. of the 9th IEEE Intl Enterprise Computing Conference*, pages 70–79. IEEE, 2005. 78, 83

[16] Anacleto Correia, Fernando Brito e Abreu, and Vasco Amaral. SLALOM: a language for SLA specification and monitoring. *CoRR*, abs/1109.6740, 2011. 79

[17] Asit Dan, Doug Davis, Robert Kearney, Alexander Keller, Richard King, Dietmar Kuebler, Heiko Ludwig, Mike Polan, Mike Spreitzer, and Alaa Youssef. Web services on demand: Wsla-driven automated management. *IBM systems journal*, 43(1):136–158, 2004. 78

[18] Amir Vahid Dastjerdi and Rajkumar Buyya. An autonomous reliability-aware negotiation strategy for cloud computing environments. In *Proc. of the 12th IEEE/ACM CCGrid*, pages 284–291. IEEE, 2012. 55, 83

[19] Amir Vahid Dastjerdi, Harshit Gupta, Rodrigo Calheiros, Soumya Ghosh, and Rajkumar Buyya. Fog computing: Principles, architectures, and applications. In *Internet of Things*, pages 61–75. Elsevier, 2016. 12

[20] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Proc. of TACAS*, pages 337–340, 2008. 32

[21] Rocco De Nicola. Process algebras. In David A. Padua, editor, *Encyclopedia of Parallel Computing*, pages 1624–1636. Springer, 2011. 44

[22] Sharaf Djemame. Enabling service-level agreement renegotiation through extending WS-Agreement specification. *SOCA*, pages 177–191, 2015. 35

[23] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. Understanding the impact of video quality on user engagement. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 362–373. ACM, 2011. 60

[24] Koustabh Dolui and Soumya Kanti Datta. Comparison of edge computing implementations: Fog computing, cloudlet and mobile edge computing. In *Global Internet of Things Summit (GIoTS), 2017*, pages 1–6. IEEE, 2017. 12, 14

[25] Eric Dubois, Kyriakos Kritikos, and Sylvain Kubicki. An automatic requirements negotiation approach for business services. In *Proc. of the 9th IEEE ECOWS*, pages 133–140. IEEE, 2011. 83

[26] Ta Nguyen Binh Duong, Xiaorong Li, Rick Siow Mong Goh, Xueyan Tang, and Wentong Cai. Qos-aware revenue-cost optimization for latency-sensitive services in iaas clouds. In *Distributed Simulation and Real Time Applications (DS-RT), 2012 IEEE/ACM 16th International Symposium on*, pages 11–18. IEEE, 2012. 73, 85

[27] Peyman Faratin, Carles Sierra, and Nicholas Jennings. Using similarity criteria to make issue trade-offs in automated negotiations. *artificial Intelligence*, 142(2):205–237, 2002. 83

[28] Peyman Faratin, Carles Sierra, and Nick Jennings. Negotiation decision functions for autonomous agents. *Robotics and Autonomous Systems*, 24(3-4):159–182, 1998. 54, 55

[29] Peyman Faratin, Carlos Sierra, Nick Jennings, and Phil Buckle. Designing responsive and deliberative automated negotiators. In *Proc. of AAAI Workshop on Negotiation: Settling Conflicts and Identifying Opportunities*, 1999. 83

[30] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE'08*, pages 1–10. IEEE, 2008. 9

[31] Xueying Guo, Rahul Singh, Tianchu Zhao, and Zhisheng Niu. An index based task assignment policy for achieving optimal power-delay tradeoff in edge cloud systems. In *Communications (ICC), 2016 IEEE International Conference on*, pages 1–7. IEEE, 2016. 85

[32] Stella Heras, Fernando de la Prieta, Sara Rodriguez, Javier Bajo, Vicente J Botti, and Vicente Julián. The role of argumentation on the future internet: Reaching agreements on clouds. In *AT*, pages 393–407, 2012. 83

[33] Yun Chao Hu, Milan Patel, Dario Sabella, Nurit Sprecher, and Valerie Young. Mobile edge computing—a key technology towards 5g. *ETSI White Paper*, 11(11):1–16, 2015. 63

[34] Michaela Iorga, Larry Feldman, Robert Barton, Michael Martin, Nedim Goren, and Charif Mahmoudi. Fog computing conceptual model. Technical report, NIST, 2018. 12

[35] Sung Ho Jang, Tae Young Kim, Jae Kwon Kim, and Jong Sik Lee. The study of genetic algorithm-based task scheduling for cloud computing. *International Journal of Control and Automation*, 5(4):157–162, 2012. 21

[36] Nicholas Jennings, Peyman Faratin, Alessio Lomuscio, Simon Parsons, Michael Wooldridge, and Carles Sierra. Automated negotiation: prospects, methods and challenges. *Group Decision and Negotiation*, 10:199–215, 2001. 83

[37] Alexander Keller and Heiko Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *JNSM*, 11(1):57–81, 2003. 79

[38] Kearney Keven, Torelli Francesco, and Kotsokalis. Constantinos. SLA: An abstract syntax for Service Level Agreements. *GRID*, 2010. 79

[39] Kleopatra Konstanteli, Tommaso Cucinotta, Konstantinos Psychas, and Theodora Varvarigou. Elastic admission control for federated cloud services. *IEEE Transactions on Cloud Computing*, 2(3):348–361, 2014. 84

[40] Kyriakos Kritikos and Rafael Brundo Uriarte. Semantic sla for clouds: Combining slac and owl-q. In *Proc. of CLOSER*, pages 432–440. ScitePress, 2017. 2, 26, 78

[41] Kyriakos Kritikos, Barbara Pernici, Pierluigi Plebani, Cinzia Cappiello, Marco Comuzzi, Salima Benrernou, Ivona Brandic, Attila Kertész, Michael Parkin, and Manuel Carro. A survey on service quality description. *ACM Computing Surveys (CSUR)*, 46(1):1, 2013. 14

[42] Khaled Mahbub and George Spanoudakis. Proactive sla negotiation for service based systems: Initial implementation and evaluation experience. In *Proc. of the 8th IEEE Services Computing (SCC)*. IEEE, 2011. 83

[43] Redowan Mahmud, Ramamohanarao Kotagiri, and Rajkumar Buyya. Fog computing: A taxonomy, survey and future directions. In *Internet of Everything*, pages 103–130. Springer, 2018. 12

[44] Yuyi Mao, Jun Zhang, and Khaled Letaief. Dynamic computation offloading for mobile-edge computing with energy harvesting devices. *IEEE Journal on Selected Areas in Communications*, 34(12):3590–3605, 2016. 85

[45] Peter Mell and Tim Grance. The NIST Definition of Cloud Computing. *National Institute of Standards and Technology*, 53(6):50, 2009. 9, 11

[46] S Nagadevi, K Satyapriya, and D Malathy. A survey on economic cloud schedulers for optimized task scheduling. *International Journal of Advanced Engineering Technology*, 4(1):58–62, 2013. 21

[47] Radheshyam Nanduri, Nitesh Maheshwari, Reddyraja, and Vasudeva Varma. Job Aware Scheduling Algorithm for MapReduce Framework. In *Proc. of CloudCom*, pages 724–729, 2011. 36

[48] Aya Omezzine, Saïd Tazi, Narjes Bellamine, Ben Saoud, Khalil Drira, and Gene Cooperman. Towards a dynamic multi-level negotiation framework in cloud computing. In *Proc. of 1st Cloud Technologies and Applications (CloudTech)*, pages 1–8. IEEE, 2015. 83

[49] Chrysa Papagianni, Aris Leivadeas, Symeon Papavassiliou, Vasilis Maglaris, Cristina Cervello-Pastor, and Alvaro Monje. On the optimal allocation of virtual resources in cloud computing networks. *IEEE Transactions on Computers*, 62(6):1060–1071, 2013. 84

[50] Adrian Paschke. RBSLA A declarative Rule-based Service Level Agreement Language based on RuleML. *Proc. of CIMCA-IAWTI*, 2:308–314, 2005. 79

[51] Al-Mukaddim Khan Pathan and Rajkumar Buyya. A taxonomy and survey of content delivery networks. *Grid Computing and Distributed Systems Laboratory, University of Melbourne, Technical Report*, 4, 2007. 70

[52] Carlos Pedrinaci, Jorge Cardoso, and Torsten Leidig. Linked USDL: A vocabulary for web-scale service trading. In *Proc. of ESWC*, pages 68–82, May 2014. 79

[53] Jing Tai Piao and Jun Yan. A network-aware virtual machine placement and migration approach in cloud computing. In *Grid and Cooperative Computing (GCC), 2010 9th International Conference on*, pages 87–92. IEEE, 2010. 85

[54] Chariklis Pittaras, Chrysa Papagianni, Aris Leivadeas, Paola Grosso, Jeroen van der Ham, and Symeon Papavassiliou. Resource discovery and allocation for federated virtualized infrastructures. *Future Generation Computer Systems*, 42:55–63, 2015. 84

[55] Charles Reiss, John Wilkes, and Joseph Hellerstein. {Google} cluster-usage traces: format + schema. Technical report, Google Inc., Mountain View, USA, nov 2011. 36

[56] Vincenzo Scoca, Atakan Aral, Ivona Brandic, Rocco De Nicola, and Rafael Brundo Uriarte. Scheduling latency-sensitive applications in edge computing. In *Proceedings of the 8th International Conference on Cloud Computing and Services Science*, pages 158–168, 2018. vii

[57] Vincenzo Scoca, Rafael Brundo Uriarte, and Rocco De Nicola. Smart contract negotiation in cloud computing. In *Cloud Computing (CLOUD), 2017 IEEE 10th International Conference on*, pages 592–599. IEEE, 2017. vii, 31, 32

[58] Damián Serrano, Sara Bouchenak, Yousri Kouki, Frederico Alvares de Oliveira Jr, Thomas Ledoux, Jonathan Lejeune, Julien Sopena, Luciana Arantes, and Pierre Sens. Sla guarantees for cloud services. *Future Generation Computer Systems*, 54:233–246, 2016. 79

[59] Ram Kumar Sharma and Nagesh Sharma. A dynamic optimization algorithm for task scheduling in cloud computing with resource utilization. *International Journal of Scientific Engineering and Technology, Volume*, 2:1062–1068, 2013. 20

[60] Sudhir Shenai and Vijindra. Survey on scheduling issues in cloud computing. *Procedia engineering*, 38:2881–2888, 2012. 21

[61] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016. 12

[62] Olena Skarlat, Matteo Nardelli, Stefan Schulte, and Schahram Dustdar. Towards qos-aware fog service placement. In *Fog and Edge Computing (ICFEC), 2017 IEEE 1st International Conference on*, pages 89–96. IEEE, 2017. 85

[63] James Skene, Davide Lamanna, and Wolfgang Emmerich. Precise service level agreements. In *Proc. of ICSE*, pages 179–188, 2004. 79

[64] Cagatay Sonmez, Atay Ozgovde, and Cem Ersoy. Edgecloudsim: An environment for performance evaluation of edge computing systems. In *Fog and Mobile Edge Computing (FMEC), 2017 Second International Conference on*, pages 39–44. IEEE, 2017. 68

[65] Nick Szabo. Smart contracts: building blocks for digital markets. *EXTROPY: The Journal of Transhumanist Thought,(16)*, 1996. 18

[66] Vladimir Tosic, Bernard Pagurek, and Kruti Patel. WSOL - A Language for the Formal Specification of Classes of Service for Web Services. In *Proc. of ICWS*, 2003. 79

[67] Guido Urdaneta, Guillaume Pierre, and Maarten van Steen. Wikipedia workload analysis for decentralized hosting. *Elsevier Computer Networks*, 53(11):1830–1845, July 2009. `http://www.globule.org/publi/WWADH_comnet2009.html`. 40

[68] Rafael Brundo Uriarte. SLAC project website. 35

[69] Rafael Brundo Uriarte, Vncenzo Scoca, Francesco Tiezzi, and Rocco De Nicola. SLAC: Formal Definitions. Technical report, IMT, 2017. `http://sysma.imtlucca.it/tools/slac/`. 24

[70] Rafael Brundo Uriarte, Francesco Tiezzi, and Rocco De Nicola. Dynamic slas for clouds. In *Proc. of the 5th European Conference on Service-Oriented and Cloud Computing (ESOCC)*, pages 34–49. Springer, 2016. vii, 2

[71] Rafael Brundo Uriarte, Francesco Tiezzi, and Rocco De Nicola. Slac: A formal service-level-agreement language for cloud computing. In *Proceedings of the 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing*, pages 419–426. IEEE Computer Society, 2014. vii, 2, 43

[72] Rafael Brundo Uriarte, Sotirios Tsaftaris, and Francesco Tiezzi. Service clustering for autonomic clouds using random forest. In *Proc. of CCGrid*, pages 515–524, 2015. 33

[73] Rafael Brundo Uriarte, Sotirios Tsaftaris, and Francesco Tiezzi. Supporting autonomic management of clouds: Service clustering with random forest. *TNSM*, 2016. 31, 33

[74] Rafael Brundo Uriarte and Carlos Becker Westphall. Panoptes: A monitoring architecture and framework for supporting autonomic Clouds. In *Proc. of NOMS*, pages 1–5, Poland, 2014. IEEE. 31

[75] Gérard Verfaillie and Thomas Schiex. Solution reuse in dynamic constraint satisfaction problems. In *Proc. of AAAI*, volume 94, pages 307–312, 1994. 28

[76] Shuo Wang, Xing Zhang, Yan Zhang, Lin Wang, Juwo Yang, and Wenbo Wang. A survey on mobile edge networks: Convergence of computing, caching and communications. *IEEE Access*, 5:6757–6779, 2017. 5, 12, 13

[77] Philipp Wieder, Joe M Butler, Wolfgang Theilmann, and Ramin Yahyapour. *Service level agreements for cloud computing*. Springer Science & Business Media, 2011. 78

[78] Linlin Wu, Saurabh Kumar Garg, Rajkumar Buyya, Chao Chen, and Steve Versteeg. Automated sla negotiation framework for cloud computing. In *Proc. of the 13th IEEE/ACM CCGrid*, pages 235–244. IEEE, 2013. 83

[79] Anagha Yadav and Suresh Rathod. Study of scheduling techniques in cloud computing environment. *International Journal of Computer Trends and Technology (IJCTT)*, 2015. 20

[80] Edwin Yaqub, Ramin Yahyapour, Philipp Wieder, Constantinos Kotsokalis, Kuan Lu, and Ali Imran Jehangiri. Optimal negotiation of service level agreements for cloud-based services through autonomous agents. In *Proc. of the 11th IEEE Services Computing (SCC)*, pages 59–66. IEEE, 2014. 83

[81] Curtis Yu, Cristian Lumezanu, Abhishek Sharma, Qiang Xu, Guofei Jiang, and Harsha V Madhyastha. Software-defined latency monitoring in data center networks. In *International Conference on Passive and Active Network Measurement*, pages 360–372. Springer, 2015. 65

[82] Lingfang Zeng, Bharadwaj Veeravalli, and Qingsong Wei. Space4time: Optimization latency-sensitive content service in cloud. *Journal of Network and Computer Applications*, 41:358–368, 2014. 85

[83] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010. 9

[84] Tianchu Zhao, Sheng Zhou, Xueying Guo, Yun Zhao, and Zhisheng Niu. A cooperative scheduling scheme of local cloud and internet cloud for delay-aware mobile cloud computing. In *Globecom Workshops (GC Wkshps), 2015 IEEE*, pages 1–6. IEEE, 2015. 85

[85] Farhana Zulkernine and Patrick Martin. An adaptive and intelligent sla negotiation system for web services. *IEEE Transactions on Services Computing*, 4(1):31–43, 2011. 55, 83