Dynamic SLAs for Clouds

Rafael Brundo Uriarte[1(✉)], Francesco Tiezzi[2], and Rocco De Nicola[1]

[1] IMT School for Advanced Studies Lucca, Lucca, Italy
{rafael.uriarte,rocco.denicola}@imtlucca.it
[2] University of Camerino, Camerino, Italy
francesco.tiezzi@unicam.it

Abstract. In the Cloud domain, to guarantee adaptation to the needs of users and providers, Service-Level-Agreements (SLAs) would benefit from mechanisms to capture the dynamism of services. The existing SLA languages attempt to address this challenge by focusing on renegotiation of the agreement terms, which is a heavy-weight process, not really suitable for dealing with cloud dynamism. In this paper, we propose an extension of SLAC, a SLA language for clouds that we have recently defined, with a mechanism that enable dynamic modifications of the service agreement. We formally describe this extension, implement it in the SLAC framework and analyse the impacts of dynamic SLAs in some applications. The advantages of dynamic SLAs are demonstrated by comparing their effect with that of static SLA and of the "renegotiation" approach.

1 Introduction

The cloud paradigm is inherently dynamic from both the consumer and the 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 may vary considerably when, e.g., clouds are used to outsource internal services or to complement the computing capacity through a hybrid cloud. Such dynamism might change providers and consumers requirements during the service provision period. Providers might need to change the agreements, e.g., to avoid the violation of agreements and to maximise revenues by serving consumers willing to pay for immediate use of the service [6]. On the other hand, consumers may modify the service, e.g., to respond to unexpected demands, to extend the expiration date of a contract or to change the amount of resources to be provided.

Clouds commonly use Service-Level-Agreements (SLAs) to regulate the provision of services. A SLA is the formalisation of the service provision characteristics, which are composed of obligations, rights and guarantees for the involved parties. In clouds, where consumers entrust crucial data and processes to other

M. Aiello et al. (Eds.): ESOCC 2016, LNCS 9846, pp. 34–49, 2016.
DOI: 10.1007/978-3-319-44482-6_3

parties, SLAs are necessary and reflecting cloud's dynamism in contracts is a crucial open issue. The need for dynamicity can be perceived by considering a situation in which a cloud provider has overbooked its resources and the load unexpectedly raises. In such case, the provider to avoid breaching SLAs, paying fines and violating the consumers' trust, might want to activate a clause in the contract that allows him to reduce the resources provided to some consumers (e.g., number of VMs) offering monetary discounts to compensate this reduction. Unfortunately, none of the existing SLA definition languages offers this possibility. Two solutions are commonly employed to mitigate this problem.

In the first, only a generic specification of the service and its quality is defined. Providers specify generic terms, such as service availability, or even service classes, such as Silver and Gold, under which new instances are created. However, this approach provides only a high-level account of the service, which may be a source of ambiguity for the verification of the service quality. Moreover, the addition of resources or changes in the provided service are subject to the prices and availability at the moment of the request, which may vary considerable since the original agreement does not impose them any restriction.

The second approach to mitigate the dynamicity problem in clouds is the renegotiation of the SLA. However, automatic (re)negotiation of SLAs is complex and time consuming [8,9,14,16]; it entails the costs of formulating, taking decision and analysing the proposed SLA modifications [9]. It does not offer the flexibility of acting/planning without the authorisation of the other parties, and does not guarantee elasticity to the service because requests can always be refused. Furthermore, renegotiation cannot replace contracts specified in natural language because they may include conditional clauses which trigger automatic changes.

To overcome the lack of support for dynamic changes in the SLA definition languages, we introduce a conceptual framework, devised for the cloud domain, that enables the specification of conditions and events in which changes (triggered by, e.g., violations or requests from parties) are explicitly permitted by the SLA. We propose two mechanisms to perform changes in the SLA. The first mechanism allows unilateral changes, where the authorisation by the involved parties is not necessary if the conditions defined in the SLA are satisfied. The second mechanism enables changes only with the explicit authorisation of the involved parties. Differently from the renegotiation approach, in this case, the modifications are defined in the contract, which allows the parties to predict possible changes and speed up the decision-making process. We implement this framework as an extension of SLAC [17], a SLA definition language for clouds which, as the other existing definition languages, did not include mechanisms to support dynamic SLAs.

The main contributions of this paper are: (i) an innovative approach to capture the dynamism of clouds in the SLAs through the definition of cases in which the terms of the SLA can be changed; (ii) a formal extension of the SLAC language to support dynamic SLAs, and (iii) the implementation and comparison of our approach against the traditional SLA renegotiation. Before presenting the

extension of SLAC to support dynamism (Sect. 2), some experiments (Sect. 3) and the impact of dynamic SLAs in different fields (Sect. 4), we briefly discuss the related works.

A generic SLA framework that supports the reservation of long term capacity at pre-specified prices is proposed in [7]. The focus is on the service admission control from the provider side and on providing solutions for capacity allocation in scenarios in which consumers reserve resources for fixed prices. This approach is problem specific and the authors clarify that the work is a pragmatic first step towards more dynamic pricing scenarios. Neither the framework, nor the SLA definition language to support this feature, nor the mechanism to reserve the resources are discussed. Similarly, other works provide solutions considering implicit changes in the services quality level, but, to the best of our knowledge, none of them defines a conceptual framework with an actual implementation or studies the contractual nature of such changes and its implications in the SLA.

An interesting discussion on the management perspective of dynamic SLAs is reported in [9] where the authors stress that, due to rapidly changing requirements of consumers and providers, clouds require dynamic SLAs. Also possible design choices for such systems are surveyed and the main phases of SLA management, such as admission control, monitoring, SLA evaluation and enforcement are clearly introduced. The focus, however, is on the discussion of the requirements of management systems, in particular of Openstack, to provide this flexibility in clouds but the issues of creating dynamic SLAs for clouds are not addressed.

The WS agreement language is extended in [3,4] to support modifications at run time using renegotiation; this involves a party requesting the desired modification and the other one accepting it. An on-line renegotiation extension for WS-Agreement is instead proposed in [5]; renegotiation templates are introduced which specify the terms that can be modified during the renegotiation (dynamic or static Service-Level-Objectives). Similarly, many other works propose renegotiation of SLAs (e.g., [6,11,13]). However, they are not suitable for the cloud domain since they do not enable the specification of changes in the agreement without a negotiation process, which involves requests, proposals and decision-making from the involved parties.

2 Supporting Dynamism in SLAC

The requirements of the parties involved in the service provision in clouds change rapidly. This changes are not limited to elasticity but can range from business aspects, e.g., expiration date and payment model, to the quality-of-service, e.g., response time. However, SLAs are commonly non-modifiable documents, unless a renegotiation process is carried out and the involved parties agree on new terms. Aiming to reflect the dynamism of clouds into SLAs without the weight of a renegotiation process, we have developed a mechanism to enable changes in SLAs inspired by contracts specified in natural language. The intuition behind this mechanism is the specification in the contract itself of the conditions and the terms which can be changed.

Table 1. Syntax of the SLAC language (an excerpt from [17]).

SLA ::=	`id:` *Id* `parties:` *PartyDef* *PartyDef*$^+$ `term groups:` *Group** `terms:` *Term*$^+$ `guarantees:` *Guarantee**
Group ::=	*GroupName* `:` *Term*$^+$
Term ::=	*Party* `->` *Party*$^+$`:` *Metric* | `[`*Expr*`,`*Expr*`]` `of` *GroupName*
Party ::=	*Role* | *PartyName*
Metric ::=	*NumericMetric* `not`$^?$ `in` *Interval* *Unit* | ...
NumericMetric ::=	`cCPU` | `RT_delay` | `response_time` | `RAM` | `price` | ...
Interval ::=	`]`*Expr*, *Expr*`[` | `]`*Expr*, *Expr*`]` | `[`*Expr*, *Expr*`[` | `[`*Expr*, *Expr*`]`
Unit ::=	`GB` | `#` | `ms` | `EUR/Hour` | ...
Event ::=	`violation`

We designed an extension of SLAC [17], named dSLAC, to support this mechanism and enable parties to modify the terms during the service provision. The SLA itself contains the specification of the conditions and terms which can be modified and is divided into two sections: the *Dynamism*, where the triggers for changes, the conditions and the modifications are specified; and the *Invariants*, which defines fixed bounds for the SLA terms.

2.1 Syntax

SLAC is a language for the specification of SLA for clouds, which focusses on: *(i)* formal aspects of SLAs; *(ii)* supporting multi-party agreements; *(iii)* business and utility aspects; and *(iv)* proactive management of the agreement as well as the cloud system. For the sake of readability and self-containedness, we report in Table 1 (an excerpt of) the syntax of the the SLAC core language (the complete syntax of SLAC and its extensions can be found in the project's web page [1]). The syntax is formally defined in the Extended Backus Naur Form (EBNF), in which *italic* denotes non-terminal symbols, while `teletype` denote terminal ones.

The *description* of a *SLA* comprises a unique identification code (*Id*) and at least two involved parties. The *terms* of the agreement express the characteristics of the service together with their respective expected values. Each SLA requires the definition of at least one term, which can be either a *Metric* or a *Group* of terms (which enables the re-use of the same terms in different contexts). Each term defines the party responsible to fulfil the term (a single party) and the contractors of the service (one or more). SLAC supports different *metrics*, e.g., the *NumericMetric*, which is constrained by open or closed *Intervals* of values and a particular *Unit*. The specification of intervals in numeric metrics relies on the evaluation of expressions (*Expr*). Finally, *Guarantees* ensure that the terms of the agreement will be enforced or, in case of a violation *Event*, define the actions that will be taken.

Table 2. Syntax of the dSLAC language.Syntax of the dSLAC language.

$$
\begin{aligned}
\mathit{SLA} \mathrel{+}= {} & \texttt{Dynamism:}\ \mathit{Dynamism}^{*}\ \ \texttt{Invariants:}\ \mathit{Metric}^{*} \\
\mathit{Dynamism} \mathrel{+}= {} & \texttt{on}\ \mathit{Event}\ \texttt{:}\ \mathit{ConditionModification} \\
\mathit{ConditionModification} ::= {} & (\texttt{if}\ \mathit{ExprModification}\ \texttt{then}\ \mathit{Modifications})^{+} \\
& (\texttt{else}\ \mathit{Modifications})^{?} \mid \mathit{Modifications} \\
\mathit{ExprModification} ::= {} & \mathit{Expr} \mid \mathit{Party}\ \texttt{authorises} \\
\mathit{Modifications} ::= {} & \mathit{Modification}\ \texttt{and}\ \mathit{Modifications} \\
& \mid \mathit{Modification}\ \texttt{or}\ \mathit{Modifications} \mid \mathit{Modification} \\
\mathit{Modification} ::= {} & \texttt{add term}\ \mathit{Term} \mid \texttt{delete term}\ \mathit{RefTerm} \\
& \mid \texttt{replace value of}\ \mathit{RefTerm}\ \texttt{with}\ \mathit{MetricValue} \\
\mathit{RefTerm} ::= {} & (\mathit{Party}\ \texttt{=>}\ \mathit{Party}^{+}\texttt{:})^{?}(\mathit{GroupName}\texttt{:})^{*}\ \ \mathit{ComposedMetric} \\
\mathit{ComposedMetric} ::= {} & \mathit{NumericMetric} \mid \mathit{ListMetric} \mid \mathit{BooleanMetric} \\
& \mid \mathit{GroupName} \\
\mathit{MetricValue} ::= {} & \mathit{Interval}\ \mathit{Unit} \mid \mathit{Boolean} \mid \{\mathit{ListElement}^{+}\} \\
& \texttt{or}\ \{\mathit{ListElement}^{+}\}^{*} \\
\mathit{Event} \mathrel{+}= {} & \mathit{Party}\ \texttt{request} \mid \texttt{SLA\_expiration} \mid \ldots
\end{aligned}
$$

The syntax of dSLAC is obtained by extending that of *SLA* and is presented in Table 2. The symbol ::= is used when a new rule is added to the syntax, while += is used to extend an existing syntactic rule.

The dynamic changes cover two sections of the SLA: *Dynamism* and *Invariants*. The former specifies events, conditions and changes, whilst the latter provides fixed rules that regulate these changes, i.e. the modification of the agreements specified in the *Dynamism* section are only performed when they are compliant with the conditions defined in the *Invariants* section. Although the invariants could be specified as conditions in the *Dynamism* part of the agreement, we opted to specify them as separated sections to have clearer specification and semantics.

The changes in the *Dynamism* section are based on *Events*, such as requests from parties or SLA violations. Then, one or more conditions can be defined (*ConditionModification*), which can be an expression *Expr*, as defined in the core language, or express the need of the authorisation from one or more parties.

Afterwards, the modification are specified using three actions: *add term*, *delete term* and *replace value*. The *add term* operation inserts a new term, which is not specified in the initial SLA. It requires a *Term*, which includes the involved parties, metric and value. The *delete term* removes a SLA term, which requires only the reference to an existing term (*RefTerm*). The *replace value* operation substitutes the value of an existing term with a new value (*MetricValue*), which can be, according to the metric type, an *Interval* and its *Unit*, a *Boolean* or a set of lists. Replacements can use reserved variable names, which refer to, e.g., the current values of the existing term, of another term or the number of times the

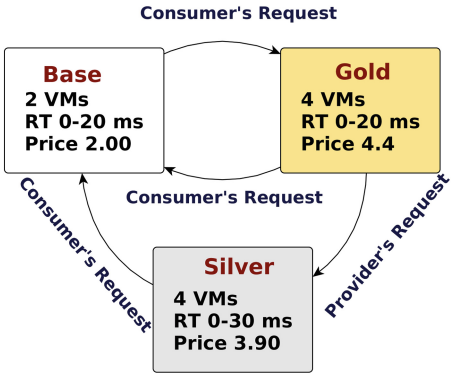

Fig. 1. Automaton of an excerpt of the example in Table 4.

SLA was violated. In case of multiple terms using the same metric (e.g., in different groups) the reference to the metric must be complete, including involved parties and groups, to avoid ambiguity.

Finally, the *Invariants* section defines rules that cannot be changed by dynamic actions, i.e. fix bounds for terms of the contract. They can define, for example, that a user cannot request more than 1000 Virtual Machines (VMs) and never less than 1 or that the provider must always backup the data of the consumer. Events that trigger changes in the contract will not be applied unless they are compliant with the terms defined in this section.

2.2 Semantics

The semantics of a SLA specified via SLAC is formulated as a Constraint Satisfaction Problem (CSP) that verifies: (i) at negotiation-time, whether the terms composing the agreement are consistent; and (ii) at enforcement-time, whether the characteristics of the service are within the specified values.

In the formal SLAC semantics [17] a SLA was given by means of a function $[\![SLA]\!]$ that returned a pair composed of a set of group definitions and a constraint representing the semantics of SLA's terms. This pair constituted the CSP associated to the agreement; details can be found in [17].

In dSLAC the possibility of changing the set of valid constraints calls for a different approach. The agreement is represented by an *automaton* in which each state is labelled by a set of constraints and each transition by the event that modifies the constraints. The initial state of the automaton is created using the original semantics of a SLA, which converts the agreement without the *Dynamism* and *Invariants* sections into a CSP. Then, all possible new states are created considering events, conditions and the triggered changes to the SLA constraints specified in the *Dynamism* section, as well as the invariants of the agreement, which are defined as additional conditions to every transition of the automaton.

Table 3. IaaS Example: consumer can add and remove any type of VM up to 200 VMs.

SLA
...
Term Groups:
Small_VM:
Imt → Rafael:cCpu in [1,1] #
Imt → Rafael:RAM in [4,4] GB
Imt → Rafael:RT_delay in [0,10] ms
Imt → Rafael:price in [0.22,0.22] EUR/Hour
Terms:
[2,2] of Small_VM
Dynamism:
on consumer request:
replace value of Small_VM with [#1+1, #2+1] *or*
replace value of Small_VM with [#1-1, #2-1]
Invariants:
Small_VM in [0,200]

Figure 1 illustrates this approach based on the second example of dynamic SLA, discussed in the next section and presented in Table 4. For the sake of simplicity, the automaton does not include the dynamic part that requires authorisation from the consumer. In this case, the consumer can request 2 additional VMs when the SLA state is the initial one (corresponding to the service quality level *Base* and passing to state *Gold*) and remove 2 of them when the total number of VMs is 4 (from state *Gold* back to sate *Base*). Also, the provider can request an increase in the Response Time in state *Gold* compensating the consumer with a reduction in the price (passing to the state *Silver*). Even if in the state *Silver*, the consumer can reduce the number of VMs to 2 and return to state *Base*.

2.3 Examples

We illustrate the dynamism of SLAs using two examples. The first is presented in Table 3; it is an instance of an Infrastructure-as-a-Service SLA from a provider named *IMT* which initially delivers 2 VM to consumer *Rafael*. In this scenario, the consumer adds and removes VMs according to his needs, if the number of VMs does not exceed 200. The #1 represents the current value of the lower bound of the interval of the term and #2 the upper bound.

In the second example, presented in Table 4, we use the same scenario and base SLA as the first example, modifying only the *Dynamism* and *Invariant* parts. In this case, the consumer adds and removes VMs for a fixed price but must respect the limits defined in the Invariants, that is, in this case, he can have only 2 or 4 VMs at the same time. The provider may request an increase in the response time if the number of VMs is equal to 4. If so, as a compensation to the consumer, the price is then reduced by 50 cents. Moreover, on provider's

request, a consumer may accept an increase in the response time of 5 ms, for a reduction of 10 cents in the price, always considering the ranges defined in the Invariants section that, in this example, limits the response time between 0 and 35 ms.

3 Experiments

To illustrate the SLAC extension and the benefits of dynamic SLAs we present a case study in a cloud testbed for the execution of cloud services and simulates the interaction between a provider and multiple consumers.

For the sake of simplicity, apart from the Dynamism and Invariant sections, the SLAs have only: the definition of the service execution deadline, the price to execute the service and the penalty in case of violation of the deadline. We compare three different approaches: *Static SLAs*, in which SLAs do not change during their lifetime; *Renegotiation*, in which the parties can renegotiate the existing SLA; and our approach, *Dynamic* SLAs, which enables the definition of modifications in the agreements. For this comparison, we analyse the number of violations, penalty and the total revenue of the provider.

The need for dynamism in this use case is demonstrated in three cases:

– Request for modification from the consumers, commonly caused by change of requirements, which we simulated by randomly selecting services in execution;

Table 4. Example of a more complex dynamic SLA.

SLA
...
Dynamism:
on consumer request:
 if Small_VM == 2
  replace value of Small_VM with [#1+2,#2+2] *and*
  replace value of Price with [#1+2.4,#2+2.4]
on consumer request:
  replace value of Small_VM with [#1-2,#2-2] *and*
  replace value of Price with [2.0, 2.0]
on providers request:
 if Small_VM == 4 and RT_delay == [0,20]:
   replace value of RT_delay with [0,30] *and*
   replace value of Price with [#1-0.5,#2-0.5]
on providers request:
 if consumer authorises:
   replace value of RT_delay with [0,#2+5] *and*
   replace value of Price with [#1-0.1,#2-0.1]
Invariants:
 RT_delay in [0,35]
 Small_VM in [2,4]

– High violation risk, which is detected during the service execution to enables the provider to change the agreement and avoid violations;
– Low violation risk, which is also detected during the service execution and enables the provider to increase its revenue, for example increasing the price but reducing the deadline of the service.

The results demonstrate the flexibility dSLAC and its capacity to reduce the number of SLA violations and to improve the revenue of the involved parties.

3.1 Use Case Model

The framework developed for the coordination of the execution of services has several components, as depicted in Fig. 2. The *SLAC Framework* parses and evaluates SLAs, which contain the service specification and requirements, and sends this description to the *Service Execution Framework*. This latter component is specifically designed to guarantee the correct deployment and execution of services and to manage the cloud infrastructure and employes schedules services using the approach presented in [18]. The Panoptes *Monitoring System* [19] provides the status of the system and services to the Violation Risk Analyser and to the SLAC Framework and is directly configured by the Service Execution Framework. The *Violation Risk Analyser* measures the risk of the running services of not meeting the deadline specified in the SLA and updates the Renegotiation Decision System. Finally, the *Renegotiation Decision System* creates proposals of modifications for the SLA and decides, when using the Renegotiation approach, whether to accept changes in the services. The violation risk analysis is performed using Supervised Random Forest [2], a machine learning technique, and is based on the monitoring information of services and the SLA itself.

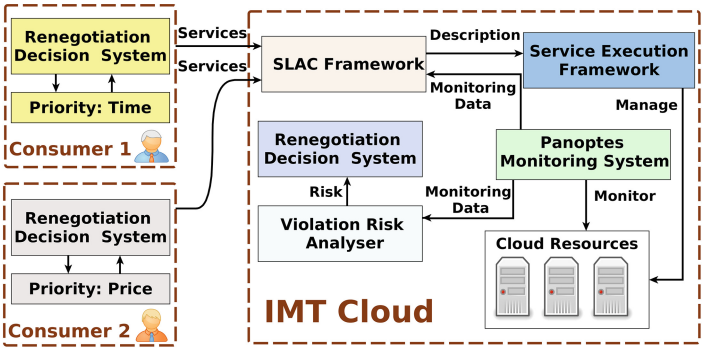

Fig. 2. Components of the use case framework.

The algorithm defined for the experiments is used for each service regardless the evaluated approaches and is depicted in Fig. 3. Each service is evaluated only

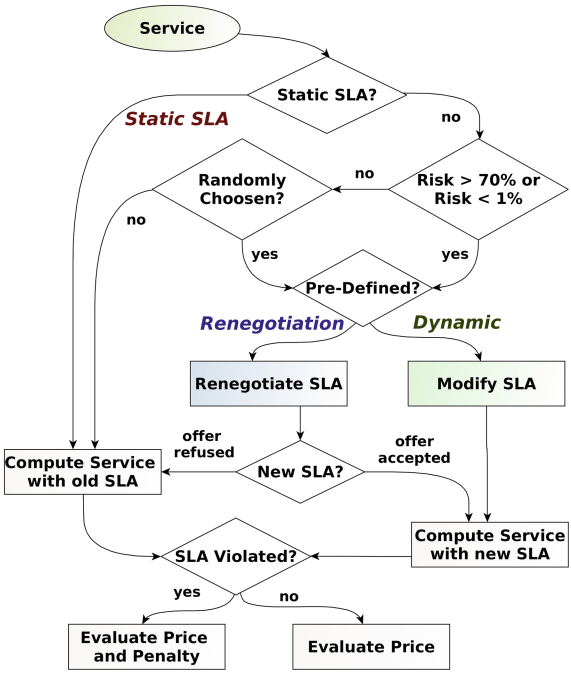

Fig. 3. Flow diagram of the tested approaches: Static, Renegotiation and Dynamic.

once during its execution lifetime, in time t_r, which is a random point between the initial time and the deadline of the service defined in the SLA. In the case of *Static SLAs*, the services are computed employing the SLA defined at design time and, when the service ends or the deadline is achieved, the system verifies whether the SLA was violated and then assesses the price paid and the possible penalties. In the *Renegotiation* approach, first the violation risk is measured. If it is not higher or lower than a specific value, the system verifies whether the service was randomly chosen. If not, the service is computed normally with the SLA defined at design time. Renegotiation take place in case of high violation risk to avoid penalties and customer insatisfaction; in case of the payment of low violation risk, to raise provider revenues by, e.g., shortening the deadline; and to simulate changes of requirements from consumers when the service was randomly chosen. A party sends a SLA proposal to the other party that analyses it according to its priorities using a Fuzzy Decision System, as described in the next section. If the new agreement is accepted, the service continues and is evaluated considering the new SLA, otherwise the initially defined agreement is the valid one till the end of the service.

In this use case, compared to Renegotiation, the only difference of the *Dynamic* approach is that the renegotiation and consentment of the involved parties is not necessary, that is, in case of low violation risk or high violation risk or the service being randomly chosen, the agreement is modified automat-

ically since the changes are pre-defined in the SLA. In both cases a bonus is given to the other when a change is made during the service execution in order to motivate or compensate the other party for the changes. 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.

3.2 Fuzzy Decision System

The Renegotiation approach requires the analysis of the difference between the initial agreement (pre-runtime) and the SLA proposed for renegotiation, and the assessment of the benefits before deciding whether accept or refuse a new agreement. In our use case, to simulate this process that is typically carried out by a human being or autonomous decision systems, we designed a fuzzy logic decision support system inspired by the approach presented in [5].

Our decision system takes as input the rate (positive or negative) of change for the considered parameter; for example, if in the renegotiation process the provider requests and increases of 20 % on the price, one of the parameters is 20. With these inputs, the system decides whether the new SLA is beneficial, neutral or not beneficial to the party. In the case of consumers, the system also takes into account the priorities of each consumer, for example, if the deadline is the priority of *consumer 1* the influence of this variable is highlighted in the decision.

Fuzzy rules interpret the relationships between the inputs and outputs and are constructed in the logical form. In the 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). Table 5 exemplifies some rules exploited in the use case and applied by the provider's fuzzy decision system. Despite being fixed for the provider, the rules change according to the priorities of each consumer. For a complete account of the fuzzy rules and the framework used in the experiments, we refer to the website of the SLAC project [1].

Table 5. Fuzzy rules of the provider decision system.

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

3.3 Evaluation

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

In the experiments, services are generated based on the distribution of a trace of real-world cloud environment, the Google's cloud dataset [15], and the same services are executed using all three described approaches. Each service has an associated SLAC 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 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, machine learning algorithms, number generation and format conversion, which are close to real-world applications [10]. Service's penalty and price are generated along with the SLA and are based on the service execution time and a randomly defined number. Penalties are always higher than the price, since the price is paid even if a service is violated.

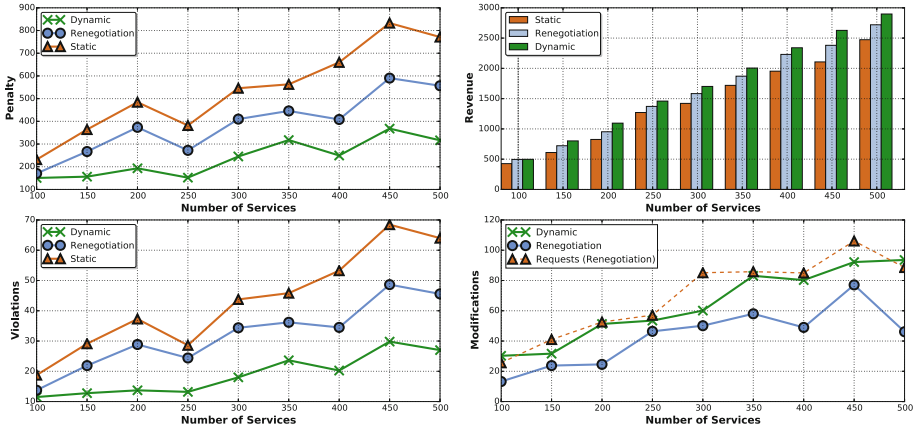

Fig. 4. Performance analysis

The training set for the SLA Risk assessment is built in every round of experiments 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 of experiments, 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, in our case, a service arrives in average every 0.7 s.

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 fulfil the need of the requester, e.g., if the violation risk is high, the provider requests more time to finish the service but offers it with a discount on the price and higher penalty. The

definition of the exact parameters of the considered metrics of the SLA modification proposal, which are used by Renegotiation and Dynamic approaches (though the latter applies changes without requesting the approval of the other party), are randomly generated in a predefined range.

The results of these experiments are illustrated in Fig. 4 considering different number of services. Table 6 presents the overall results, relative to the Renegotiation and Dynamic approaches, expressed as percentages: in the case of Penalties and Revenue characteristics, the results correspond to a comparison with the Static approach, whilst in the case of the other measured characteristics 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 machining 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, in this case, were considered high-risk. In the Renegotiation and Dynamic approaches, 14 % and 19 % of the SLAs classified as high-risk and more than 10 % of the Randomly selected were violated. 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.

3.4 Discussion

In the experiments, the use of the renegotiation and the dynamic mechanisms of SLAs heavily depend on the accuracy of the violation risk analyses approach. The results show that, although the penalties were reduced by 64 %, the impact

Table 6. Experimental results

	Renegotiation	Dynamic
Modification Requests	24 %	0 %
Modifications	13 %	21 %
High Risk	11 %	19 %
Low Risk	0.1 %	0.2 %
Random	1.8 %	1.1 %
Violated High Risk	19 %	14 %
Violated Low Risk	0 %	3 %
Violated Random	10 %	12 %
Penalties	−31 %	−64 %
Revenue	13 %	22 %

on the total revenue was an increase of around 22 %. The main reasons for this difference are: the limited impact of the penalties on the total revenue due to the average number of violations; 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; and the number of modified SLAs which were violated since most of the modification requests increase the penalty as a compensation to increasing the service completion time, which 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 opportunities the Dynamic and Renegotiation mechanisms were used to improve the revenue of the parties (only around 1.4 % of the services were considered low-risk or randomly selected). In most scenarios, these mechanisms can be more aggressively employed to improve the revenue, mainly when a better accuracy is reached by the risk analysis.

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.

Finally, the results demonstrate that dSLAC provides flexibility for the parties and significantly improves the optimisation of the SLA management. Moreover, it can always be used together with the renegotiation approach in case not all relevant modifications are included in the SLA.

4 Conclusions and Future Works

To address the lack of support for the cloud dynamism in the existing SLA definition languages we have introduced a new approach for the specification of dynamic SLAs for clouds. This dynamism is achieved through the specifications, in the SLA itself, of the conditions and of the modifications that will be applied to the SLA once the related conditions are met. We have introduced syntax and semantics of an extension of the SLAC language and described its implementation and possible usage. We have provided evidence of the advantages of our approach in comparison to static SLAs and the use of renegotiation.

Since our approach is devised also for business, it may be used to back-up legal disputes. In fact, in designing dSLAC, we took the legal aspects of contract formation into account. Our approach is compliant with the norms defined in [12], where the authors discuss the provision of services from the legal standpoint using the European Union directives for E-Commerce as reference. Thus, dynamic SLAs, i.e. SLAs with pre-defined changes based on events and conditions, can be used in legally binding contracts.

We conclude by discussing the impact of dynamic SLAs in the cloud domain and the related challenges that we plan to address as future work. Indeed, this approach impacts in areas related to the creation and management of services and poses new important challenges in the field while looking at them from different perspectives.

The *negotiation* process needs considerable changes to support this feature. The first challenge is the matching of offers and requests of services since the possible changes must be considered. This matching mechanism needs to verify whether the requirements of the parties comply with all possible states of the SLA, taking into account the conditions for such changes. Depending on the changes defined on the SLA, a large number of states need to be analysed, which is a computation intensive process, and new techniques are needed to address possible problems, such as explosion of states. Moreover, crisp solutions that simply verify whether there exist a single state that is not compatible with the specification of the parties may imply low matching rates, whilst an algorithm considering the probability of reaching a non-desired state could improve this rate. Considering the complexity of this process, the negotiation can hardly be performed by humans and new algorithms, for example, based on model checking, that take into account the priorities of the involved parties should be develop to define SLA proposals and to assess their benefits.

The *scheduling*, *service admission* and *resource reservation* areas also need to consider the possible changes in the services. As the modifications in the agreement do not need the authorisation from the provider, the consumer can request changes in unexpected times and the provider must cope with them. Although most of the existing methodologies already employ statistical methods to predict the load of the system, the agreement with pre-defined changes is valuable source of knowledge as it contains the explicit definitions of the changes which are more likely to happen or that are expected by the parties. Moreover, the methodologies can also use the conditions pre-defined in the SLAs to adjust the load of the system. This process is complex and requires multi-objective solutions to find the optimal (or a better) scenarios for such cases since the possibilities in large scale are numerous. For example, even if a consumer requests a large number of new VMs, the provider can avoid violations of SLAs by removing VMs from other consumers, by e.g., providing a discount to them.

Service and infrastructure *Monitoring* are essential for the decision-making and SLA management. With dynamic SLAs, this process needs to adapt the knowledge generation methods to use the collected data before and after the changes. Finally, management systems must track the changes in the SLA for billing purposes and for legal reasons (e.g., in case of disputes).

References

1. SLAC: A Formal Service-Level-Agreement Language for Cloud Computing (2016). http://sysma.imtlucca.it/tools/slac/
2. Breiman, L.: Random forests. Mach. Learn. **45**(1), 5–32 (2001)
3. Di Modica, G., Regalbuto, V., Tomarchio, O., Vita, L., Doria, V.A.: Dynamic re-negotiations of SLA in service composition scenarios. In: SEAA, pp. 359–366. IEEE (2007)
4. Di Modica, G., Tomarchio, O., Vita, L.: Dynamic SLAs management in service oriented environments. J. Syst. Softw. **82**(5), 759–771 (2009)
5. Djemame, S.S.K.: Enabling service-level agreement renegotiation through extending WS-Agreement specification. SOCA **9**, 177–191 (2015)

6. Galati, A., Djemame, K., Fletcher, M., Jessop, M., Weeks, M., Hickinbotham, S., McAvoy, J.: Designing an SLA protocol with renegotiation to maximize revenues for the CMAC platform. In: Haller, A., Huang, G., Huang, Z., Paik, H., Sheng, Q.Z. (eds.) WISE 2011 and 2012. LNCS, vol. 7652, pp. 105–117. Springer, Heidelberg (2013)
7. Garg, R., Saran, H., Randhawa, R., Singh, M.: A SLA framework for QoS provisioning and dynamic capacity allocation. In: IWQoS, pp. 129–137. IEEE (2002)
8. Green, L.: Service level negotiation in a heterogeneous telecommunication environment. In: I4CT. IEEE (2004)
9. Lee, C.A., Sill, A.F.: A design space for dynamic service level agreements in OpenStack. J. Cloud Comput. **3**(1), 17 (2014)
10. Nanduri, R., Maheshwari, N., Reddyraja, A., Varma, V.: Job aware scheduling algorithm for mapreduce framework. In: CloudCom, pp. 724–729. IEEE (2011)
11. Omezzine, A., Tazi, S., Bellamine, N., Saoud, B., Drira, K., Cooperman, G.: Towards a dynamic multi-level negotiation framework in cloud computing. In: CloudTech. IEEE (2015)
12. Parkin, M., Kuo, D., Brooke, J., MacCulloch, A.: Challenges in EU grid contracts. In: Exploiting the Knowledge Economy: Issues, Applications and Case Studies, pp. 67–75. IOS Press (2006)
13. Parkin, M., Hasselmeyer, P., Koller, B.: An SLA re-negotiation protocol. In: NFPSLA-SOC (2008)
14. Pichot, A., Wäldrich, O., Ziegler, W., Wieder, P.: Towards dynamic service level agreement negotiation: an approach based on WS-Agreement. In: WEBIST, pp. 107–119. SCITEPRESS (2009)
15. Reiss, C., Wilkes, J., Hellerstein, J.L.: Google cluster-usage traces: format + schema. Technical report, Google Inc. November 2011. http://googleclusterdata.googlecode.com/files/Googlecluster-usagetraces-format+schema(2011.10.27external).pdf
16. Shen, W., Li, Y., Ghenniwa, H., Wang, C.: Adaptive negotiation for agent-based grid computing. JASA **97**(457), 210–214 (2002)
17. Uriarte, R.B., Tiezzi, F., De Nicola, R.: SLAC: a formal service-level-agreement language for cloud computing. In: UCC, pp. 419–426. IEEE (2014)
18. Uriarte, R.B., Tsaftaris, S., Tiezzi, F.: Service clustering for autonomic clouds using random forest. In: CCGrid, pp. 515–524. IEEE (2015)
19. Uriarte, R.B., Westphall, C.B.: Panoptes: a monitoring architecture and framework for supporting autonomic clouds. In: NOMS. IEEE (2014)