Disclaimer

This copy is a preprint of the article self-produced by the authors for personal archiviation. Use of this material is subject to the following copyright notice.

IEEE Copyright notice

Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works, must be obtained from the IEEE. Contact: Manager, Copyrights and Permissions / IEEE Service Center / 445 Hoes Lane / P.O. Box 1331 / Piscataway, NJ 08855-1331, USA. Telephone: + Intl. 908-562-3966.
A Proposal of a Simulation-based Approach for Service Level Agreement in Cloud

Massimiliano Rak  
Seconda Università di Napoli,  
Dip. di Ingegneria dell’Informazione,  
Aversa, Italy  
massimiliano.rak@unina2.it

Antonio Cuomo, Umberto Villano  
Università degli Studi del Sannio,  
Dip. di Ingegneria,  
Benevento, Italy  
{cuomoant,villano}@unisannio.it

Abstract—Clouds offer resources as services, which are charged on the basis of their actual usage. An approach that is currently spreading, to face possible problems linked to a low quality of service offered, is the adoption of Service Level Agreements (SLAs). In this paper we propose the adoption of a simulation-based approach for supporting a framework in charge of SLA management in cloud applications. The idea stems from the consideration that to face the inevitable impact on QoS of the cloud elasticity, it is necessary to evaluate in a predictive way the performance properties of the many different configurations among which the automatic cloud management can choose to allocate resources to customers. This paper proposes the architecture of the framework for SLA management in clouds, identifying the requirements implied by the simulation to be performed and suggesting the adoption of a simulation engine that fits with the proposed requirements.

I. INTRODUCTION

One of the key features to which the emerging cloud computing paradigm owes its success is the adoption of the pay-per-use business model. Clouds offer every kind of resource in terms of services, which are charged on the basis of their actual usage. The main economic issue with this model is linked to the quality of such services. As a matter of fact, unsatisfactory performance of offered resources (or services) may lead to increased service time and so to higher costs charged to the cloud user. The provision of services with a given quality of service level is a particularly tough problem due to the intrinsic characteristics of clouds. In order to offer all resources through a service-oriented approach, clouds heavily rely on the adoption of virtualization techniques and on physical resource sharing among different customers. This implies that performance can hardly be predicted with good accuracy. Moreover, nowadays a widely diffused approach is to develop cloud services using federation frameworks (like mOSAIC [1], Contrail[2], OPTIMIS [3], Cloud@Home [4]) which can rent resources from a provider, possibly reallocating the services dynamically, to help reduce the costs. In this case, it is almost impossible to obtain a clear evaluation of the performance of the offered services.

An approach that is currently spreading to face possible problems linked to a low quality of service offered on the cloud, is the adoption of Service Level Agreements (SLAs). An SLA is a contract among customers and providers that guarantees the quality level of offered services. Notwithstanding the recent research efforts on this topic (full projects as SLA@SOI [5], SLA-related tasks in projects as CONTRAIL, OPTIMIS, mOSAIC, independent research both from academia and industry), at the stage of the art the SLAs offered by cloud providers are typically natural language documents that give only a few simplistic guarantees, such as a generic availability of 99% on an annual evaluation basis. In this paper we propose the adoption of a simulation-based approach for supporting a framework in charge of SLA management in cloud applications. The idea stems from the consideration that to face the inevitable impact on QoS of the cloud elasticity, it is necessary to evaluate in a predictive way the performance properties of the many different configurations among which the automatic cloud management can choose to allocate resources to customers. This evaluation should be reasonably accurate and fast, in order to avoid adding non-negligible overhead. In our opinion, a possible solution to perform fast predictive evaluations is the use of concurrent simulation [6], which makes it possible to launch multiple independent simulation in parallel and to obtain reasonable response time even for large clouds. Simulation-based predictions should be exploited at every step of SLA life cycle (Negotiation, Monitoring, Enforcement) for taking optimized decisions.

This is an idea paper, where we intend to propose the general architecture of the framework for SLA management in clouds, identifying the requirements implied by the simulation to be performed and suggesting the adoption of a simulation engine, which fits with the proposed requirements. The rest of the paper is organized as follows: the next section illustrates the SLA life cycle, outlining the requirements for each phase and sketching the architecture of the SLA management framework based on simulation. A subsection is devoted to each phase to illustrate in detail how to use the simulation-based predictions at each step of the SLA life cycle. Section III describes the requirements for simulation
and proposes the adoption of a concurrent simulation engine. The paper closes with a section dedicated to related work and with our conclusions.

II. SLA LIFE CYCLE AND FRAMEWORK ARCHITECTURE

Service Level Agreements (SLAs) aim at offering a simple and clear way to build up an agreement between final users and the service provider in order to establish what is effectively granted. ITILv3 defines Service Level Agreement (SLA) as: “An agreement between an IT Service Provider and a Customer. The SLA describes the IT Service, documents Service Level Targets, and specifies the responsibilities of the IT Service Provider and the Customer. A single SLA may cover multiple IT Services or multiple Customers”.

In cloud environments, where every resource is offered as a service, SLAs assume a relevant role, being the only way to clearly state what the provider actually offers. In recent years, many studies have been done aiming at building solutions that enable a cloud provider to offer its services through an SLA-based approach. However, the majority of them focuses on complex frameworks, as SLA@SOI [5], which are to be integrated into the provider infrastructure.

Almost all the proposed SLA systems assume that providers have full control on their infrastructure and are able to write down SLA templates that represent the class of guarantees they are able to provide to customers. Stated another way, they are provider-centric. In [7], the authors show that in clouds this is not always correct, as in most cases service providers rent resources from external providers and offer them as as a service to end users. In such contexts, SLAs are user-centric, in that they start the negotiation process from user needs and not from what providers are able to offer.

For both user-centric or provider-centric SLA management, it is possible to identify a single SLA life-cycle, which represents the different phases an SLA passes through: Negotiation, Monitoring, Enforcement (Figure 1).

In the Negotiation phase the SLA is not fully defined. Customer(s) and provider(s) follow a negotiation protocol in order to find an agreement on what the SLA should actually offer. During the negotiation, customers evaluate the trade-off among what they aim at being granted and the cost of such grants. Service providers, instead, have to evaluate the services offered, trying to predict resource consumptions, to evaluate the cost of acquired resources and what effectively can be granted. Service providers have also to evaluate the risks related to incorrect evaluations, which may lead to breach of the SLAs and to consequent penalties (which must be stated into the SLA).

During the Monitoring phase, a signed SLA is checked in order to assure that it is correctly respected. Customers aim at checking SLAs in order to be sure that providers respect them and in order to apply penalties when they are not respected. It should be noted that at the state of the art there exists no practical solution to support users in such monitoring functions, which usually are performed manually. From the service provider point of view, instead, monitoring has two different roles: to verify that the SLAs are respected, exactly as the customers will like to do, but by exploiting the access to underlying infrastructure which is usually not allowed to end users, and to generate alerts before SLAs are broken, in order to activate actions that could prevent the violation of the SLA.

The last step of the SLA life Cycle is the Enforcement, which is the step in which the actions needed to respect the SLA are effectively taken. This may imply activation of software modules, acquisition of resources (in the correct amount), and possibly even a dynamic reconfiguration of resources after an alert is generated. From the user point of view, Enforcement is just the application of the service requirements explicitly requested in the SLA. From the service provider point of view, instead, the Enforcement phase is the phase in which SLA requirements are actually applied on the acquired resources.

The three phases are strictly related: the Negotiation cannot be performed without taking in consideration how the SLA can be granted, i.e., how the enforcement will take place. Enforcement needs Monitoring in order to evaluate the real state of the solution before applying the policies and the predefined procedure, whereas Monitoring need the results of Negotiation in order to know what to monitor and which alerts should be generated. On the other hand, each phase takes in consideration different aspects of the SLA, and so existing techniques and solutions often aim at solving only one of such aspects, not taking in consideration the others.

The main problem with SLA in cloud environments is linked to the elasticity that the cloud approach implies. A cloud service is offered to a large number of end users and runs consuming resources that are dynamically acquired and/or released depending on user requests. The cloud elasticity makes the adoption of formal and analytical model hard to apply: changes in the class and type of resources may lead to a completely different behaviour, which implies a complete rewriting of models. Simulation models, instead, can be built following the same schema of the real environment, enabling the adaptation of the models
Figure 2. Simulation-based SLA Framework

as the system changes.

The approach we propose is to build a SLA Framework exploiting performance predictions obtained by simulations at each step of the SLA life-cycle. Figure 2 sketches the proposed architecture: the framework runs, consuming resources from cloud, and offers SLA-oriented services to end users. The framework has three main modules dedicated to the SLA phases. Each of them uses the same simulation engine to make decisions about SLA (in order to accept or to refuse, to generate alerts and/or to activate reactions).

In the following, we show how to use simulation in the three steps of the life-cycle, prior to outlining the requirements for the simulation engine in Section III. Our solution is based on the use of the mOSAIC SLA framework, previously successfully used in similar conditions [7], [8].

A. Simulation-based Negotiation

As mentioned before, the main goal of the Negotiation phase is to define the SLA, taking into account both the user needs and the service provider capabilities. In Negotiation, we identify the following simple concepts:

- **Negotiation Peers**: they are the peers involved in the negotiation, searching for an agreement. Negotiation peers may be human or software agents.
- **Negotiation Model**: it is a generic interaction model that is adopted in order to find an agreement among the peers of the negotiation. The negotiation model may assume a one-to-one relationship among peers, but even more complex interactions.
- **Negotiation Protocol**: it is the set of rules that peers of the negotiation obey in order to find an agreement. Negotiation protocols are usually represented in terms of messages exchanged and state of the peers. Negotiation protocols implement Negotiation models.
- **Negotiation Applications**: applications that use negotiation protocols, possibly offering a simple interface to human actors.

We assume that the goal of the negotiation is reaching an agreement upon a given service level, which can be represented in many different ways. We assume that SLAs are represented in terms of WS-Agreement\(^1\), even if the assumptions that follow can be easily applied to every kind of SLA representation format (like [9], [5]). Software negotiation peers need solutions to automatize the decision process. We aim at using simulation to support such decisions. We will focus on Negotiation models that assume that one of the peers is human (the cloud end users) and the other is a software agent that adopts our simulation-based SLA solution. The negotiation model we consider is, in the following example, very simple: a template-based Accept or Refusal negotiation. The service providers offer SLA in terms of templates, end users fill the template with their needs and the provider accepts or refuses the SLA requested, according to the status of his resources. The role of simulation will be to offer a fast support for evaluating the effect on performance of acquiring the resources, evaluating the trade-off among the effect of granting SLA and the cost of resources. In such a context, the simulation-based negotiation module should act as in listing 1.

**Data**: Service Request

**Result**: Refusal if service is not simulable or request is not satisfiable, acceptance if request is satisfiable

- Extract the service and the parameters;
- Extract the guarantee terms;
- if Simulations models are available for service and terms then
  - Define simulation scenarios;
  - Invoke simulations;
  - Apply decision criteria;
  - if A simulation matches the criteria then
    - Accept
  - else
    - Refuse
  - end
- else
  - Refuse;
end

**Algorithm 1**: Accept or Refuse algorithm

Such approach assumes a full control of resources by systems administrators and cannot be applied to generic cloud services where resources are dynamically acquired, depending on user needs, and are modeled as “infinite”. However, the proposed approach can be adapted when the Negotiation Model, and the related Negotiation protocol, become more complex, a thing that will be discussed in future work.

The final result of the negotiation process is an agreed SLA, stored in a secure way and available to Monitoring and Enforcement components.

\(^1\)http://www.forge.ogf.org/documents/GFD.192.pdf
B. Monitoring

SLA Monitoring has the key role of generating alerts when the SLA are effectively violated and, most important, when there are conditions that may lead to violations of the agreement. In this way enforcement components may (re)act in order to avoid the penalties. The Monitoring components have as an input the collection of agreed SLA (generated by SLA Negotiation modules). Such SLA contains the terms to be guaranteed, which represent the parameters to be monitored. When the Negotiation module signs a new SLA it notifies the Monitoring module of the new document. The role of the Monitoring system is twofold: to collect real measurements from the target services and to generate alerts for the target environment. In order to collect real measurements, the system should use standard monitoring solutions, possibly coordinating them at multiple levels, as proposed in [10].

At the state of the art, such monitoring/alerting solutions have been proposed for cloud environments ([11]), but they use a simple static model to identify risky conditions. We propose that the monitoring system relies on simulation to predict possible SLA violations. At predefined (possibly regular) intervals of time, the Monitoring system can launch a predefined set of simulations, using as input the measurements made in the last interval(s) of time. Simulations may predict the future evolution of the system, and put in evidence possible forthcoming SLA violations. If violations are foreseen, the Monitoring system generates an alert, which must be managed by the Enforcement module.

Listing 2 summarizes the approach described above. The monitoring system executes an event-based loop in which it reacts to two predefined events (new SLA and timeout) that determine the execution of simulations.

```
Result: Generation of Alert event when needed
Input: New SLA event
begin
  Monitoring module extract service, parameters and guarantee terms;
  Set Monitoring agents;
  Set Timeout;
  Startup Monitoring;
end
Input: Timeout event
begin
  Monitoring Module collects state data;
  Start Simulations using collected data as input;
  Compare Simulation results against alerting conditions and possibly generate alerts;
end
```

Algorithm 2: Monitoring Two Phase Behavior

C. Enforcement

The role of the Enforcement module in SLA systems is to perform all the actions needed to grant SLAs. Enforcement takes place in two different phases of SLA life cycle: (i) when a new SLA is signed, and (ii) during the life cycle of the offered service. In the first case, after the negotiation process, the Enforcement module has the role of correctly configuring the offered service, following the SLA constraints. In the second case, the Enforcement module has to react to alerts generated by the Monitoring module and to perform changes that affect the system behavior (reconfiguration) so that SLA violations are avoided. The first case (enforcement of a newly signed SLA) is strictly related to the result of the negotiation process, and results in the execution of the negotiated service with the right configuration: no decisions are taken. At the state of the art, this is the most common solution (if not the only one) supported by existing frameworks.

In the solution we propose, the Enforcement module aims at offering support for both cases, exploiting simulation to make decisions about the actions to be taken. So the Enforcement module can be thought of as composed of (i) Execution submodule which starts up the (remote) services with the right parameters and configurations and (ii) Evaluation submodule which reacts to monitoring alerts and identifies the actions to be taken.

The first module is used immediately after the negotiation: in the first case (enforcement of newly signed SLA) no new evaluations are needed, as in the negotiation phase all the needed decisions have been made. The second module, instead, is the most innovative one. It has the role of evaluating the possible choices, comparing them and generating the resulting actions.

The approach we propose is similar to the one adopted by autonomic engines as MAWeS [12] and CHASE [13], which use simulation to identify the actions to be taken. When a monitoring alert arrives, the Evaluation module starts up a set of concurrent simulations with different configurations and parameters. Then it collects the simulation results, identifies the new actions to be taken and invokes the Execution module in order to reconfigure the target services.

III. SIMULATION ENGINE REQUIREMENTS

The approach described in the above sections assumes that the simulation engine is able to predict the behaviour of the target cloud services and to help making decisions. In this section we outline the characteristics that it should have to provide the decision support for the SLA framework.

All the SLA modules need to launch simulations in order to compare the performance impact of different choices and to make decisions. Moreover, the Negotiation module has to identify the parameters to be granted, the Monitoring to generate alerts, and the Enforcement to identify the right reactions. As a consequence, the main requirement of the
simulation engine is the capability to manage a high number of simulations, corresponding to different sets of parameters, representing different configurations of the services.

The execution of an high number of simulations is obviously time consuming. This implies that the engine should be able to optimize the trade-off among the number of simulations (i.e., how many different scenarios should be evaluated) and the correctness of the predictions (i.e., the granularity of the models and the repetitions of the basic simulation run needed for statistical validation). Moreover, the engine should be able to run in parallel as many simulations as possible in order to reduce the time needed to make a single choice.

Using simulation to support fast cloud management decisions requires that a number of suitable models should be prepared beforehand to compare easily different choices, taking into account the different features to compare. As an example, the target service (i.e., the service whose behavior is granted by SLA) can run using multiple virtual machines. The simulation engine should be able (i) to describe how each request affects the consumption of resources (like memory and CPU of the virtual machine), (ii) to simulate different workloads (i.e., collections of concurrent requests of the service) and (iii) to take into account the state of the available resources (number of virtual machines, memory and CPU usage).

This implies that simulation models should be easy to compose and adapt to scenarios that differ only slightly one from another. The approach we propose is to build models structured in three logical levels: (i) the logical behavior of the system (i.e., sequence of memory accesses, cpu bursts, communication messages and how they are nested all together) which is independent from the resources actually available, (ii) the mapping of logical components over available resources, i.e., which resources are consumed by a given action and (iii) the time parameters derived from the monitoring system that represent the actual state of the system, e.g., the amount of available CPU cycles or memory.

It should be noted that the tool needs a simulation model for each new service on which SLAs have to be granted. This implies that the simulation engine should be able to dynamically load new models and reuse them. Moreover, it should be able as much as possible to automatize as much as possible the process of the generation of new configurations from model parameters, in order to ease the comparison of different choices.

We propose to use the JADES library [14] for simulation and the mJADES concurrent simulation system [6] as a basis to build up the solution. The above cited tools already have some of these features (dynamic loading of models, concurrent simulation in the cloud, ...) and can be enriched in order to be the core solution for the simulation engine described above.

IV. RELATED WORK

Service Level Agreements have widespread diffusion in the context of grid computing. For example, the BEinGRID project\(^2\) adopted a SLA-based approach to manage resources, offering grants to grid users. The use of SLAs is commonly related to the adoption of good practices and clear resource management, as proposed and supported by gSLM project ([15]). The activity of IBM in this field led to the development of the language WSLA ([16]), dedicated to describe, to negotiate and to manage SLAs. SLAng [9] is an alternative language, developed in an academic environment with similar purposes. These activities converged in the definition of the WS-Agreement standard, which offers a standard XML-based language for SLA description and a negotiation protocol based on Web Services.

The main goal of such efforts was the description of SLAs, along with the development of frameworks aimed mainly at helping to negotiate with users the terms to be granted. All such frameworks assume that system (grid) administrators set up templates of acceptable SLA, which end users fill with their specific needs. SLA systems have to check the templates against the real state of resources. In grid environments, the assumption is that resources are well known and can be easily monitored through suitable tools (e.g., Ganglia).

The SLA@SOI project [5] aims at porting the ideas for SLA definition and management developed in the grid context to generic service-oriented architectures. It proposes an architecture based on custom SLA managers that can model the system behavior in terms of service interactions. The project proposes a language compatible with WS-Agreement, but with additional features, and a framework for the development of SLA managers. SLA@SOI negotiation and resource management also relies on the use of templates, but offers no solution to support decisions. An SLA infrastructure is set up, but the criteria to be used to make decisions are delegated to system administrators.

In cloud environments, SLA@SOI results can be exploited to build a Cloud Provider Datacenter (there are examples of integration with OpenNebula, as in the Contrail project ([2])). But no clear solutions have been proposed to manage cloud elasticity, which reduces the predictability of the system. Examples of SLA-based solutions that tackle the problems raised by cloud intrinsic characteristics can be found in the results of the Cloud@Home project [17], [4], where a SLA manager based on WSAG4J is used to manage federated resources. Buyya et al. propose a solution that relies on the Cloudsim simulator for the support decisions ([18]). The two projects above are probably the only examples of the use of simulation to support SLA management. However, in all the mentioned works, simulation supports only SLA

\(^2\)http://www.it-tude.com/projects/beingrid
negotiation, and it is not fully integrated in the SLA life cycle as proposed in this paper.

V. CONCLUSIONS

In this idea paper we have proposed a framework for Service Level Agreement management for cloud applications based on simulation-based predictions. The novelty of the approach is in the adoption of simulation in each phase of SLA life-cycle, that implies an high number of concurrent simulations to be performed. This is made possible by the use of the mJADES simulation engine. We have outlined the requirements for the simulation engine, showing the complexity of such approach from the simulation point of view. The framework is to be offered as a third-party set of services, which can be used by end users to have grants about the resources they are using and/or by providers to offer their service with SLA-based features. In our future work, we intend to build prototypes of the framework to be used for simple use cases, to demonstrate the feasibility of the approach proposed.

REFERENCES


