Towards QoS-Oriented SLA Guarantees for Online Cloud Services

Damián Serrano, Sara Bouchenak, Yousri Kouki, Thomas Ledoux, Jonathan Lejeune, Julien Sopena, Luciana Arantes, Pierre Sens

HAL Id: hal-00780000
https://hal.archives-ouvertes.fr/hal-00780000v2

Submitted on 2 Feb 2016

Towards QoS-Oriented SLA Guarantees for Online Cloud Services

Damián Serrano, Sara Bouchenak
University of Grenoble – LIG
Grenoble, France
{Firstname.Lastname}@imag.fr

Yousri Kouki, Thomas Ledoux
EMN – INRIA – LINA
Nantes, France
{Firstname.Lastname}@inria.fr

Jonathan Lejeune, Julien Sopena,
Luciana Arantes, Pierre Sens
LIP6 – INRIA, Paris, France
{Firstname.Lastname}@lip6.fr

Abstract—Cloud Computing provides a convenient means of remote on-demand and pay-per-use access to computing resources. However, its ad hoc management of quality-of-service and SLA poses significant challenges to the performance, dependability and costs of online cloud services. The paper precisely addresses this issue and makes a threefold contribution. First, it introduces a new cloud model, the *SLAaaS* (*SLA aware Service*) model. *SLAaaS* enables a systematic integration of QoS levels and SLA into the cloud. It is orthogonal to other cloud models such as SaaS or PaaS, and may apply to any of them. Second, the paper introduces *CSLA*, a novel language to describe QoS-oriented SLA associated with cloud services. Third, the paper presents a control-theoretic approach to provide performance, dependability and cost guarantees for online cloud services, with time-varying workloads. The proposed approach is validated through case studies and extensive experiments with online services hosted in clouds such as Amazon EC2. The case studies illustrate SLA guarantees for various services such as a MapReduce service, a cluster-based multi-tier e-commerce service, and a low-level locking service.

Keywords-SLA; QoS; Cloud Computing; Specific Language; Online Control;

I. INTRODUCTION

Cloud Computing is a paradigm for enabling remote, on-demand access to a set of configurable computing resources [1]. This model aims to provide hardware and software services to customers, while minimizing human efforts in terms of service installation, configuration and maintenance, for both cloud provider and cloud customer. A cloud may have the form of an Infrastructure-as-a-Service (IaaS), a Platform-as-a-Service (PaaS) or a Software-as-a-Service (SaaS). However, cloud's ad-hoc management in terms of quality-of-service (QoS) and Service Level Agreement (SLA) poses significant challenges to the performance, availability, energy consumption and economical costs of the cloud. Existing public clouds provide very few guarantees in terms of performance and dependability [2]. This is the case for Amazon EC2 compute service and Amazon S3 storage service [3], Rackspace Cloud Servers compute service and Rackspace Cloud Files storage service [4], Azure Compute and Azure Storage [5].

We believe that a differentiating element between Cloud Computing environments will be the QoS and the SLA provided by the cloud. This raises the following questions: (i) How to consider SLA in a general way for different cloud

environments? (ii) How to describe the SLA terms between cloud provider and cloud customer, such as service levels, penalties in case of SLA violation, etc. (iii) How to provide guarantees on cloud QoS and provide better than best-effort behavior for clouds?

The contributions of this paper are as follows:

- A novel cloud model is proposed: *SLAaaS* (*SLA-aware-Service*). The SLAaaS model enriches the general paradigm of Cloud Computing, and enables systematic and transparent integration of service levels and SLA into the cloud. SLAaaS is orthogonal to IaaS, PaaS and SaaS clouds and may apply to any of them.
- A specific language is introduced to describe QoS-oriented SLA associated with cloud services, the *CSLA* (*Cloud Service Level Agreement*) language.
- A control-theoretic approach is described to provide performance, dependability and cost guarantees for online cloud services, with time-varying workloads.
- Three case studies running on private clusters and Amazon EC2 public cloud illustrate the soundness of the proposed approach. These include the first SLA-oriented dynamically provisioned MapReduce service, a multi-tier e-commerce service, and a SLA-oriented locking service.

The rest of the paper is organized as follows. Section II introduces the proposed SLAaaS cloud model, CSLA language and online cloud control. Section III presents the experimental case studies. Section IV reviews the related work, and Section V draws our conclusions.

II. SLAaaS CLOUD MODEL

A. Background

In this section, we first provide preliminary definitions before introducing the SLAaaS cloud model. A cloud provides a set of services. A cloud service exposes a functional interface with operations to call on the cloud. For instance, an IaaS cloud as Amazon EC2 exposes a functional interface that allows users to acquire compute instances, to run software on these instances or to release instances. Amazon S3 IaaS cloud service exposes a functional interface that allows users to store, read or delete any amount of data. Amazon RDS PaaS cloud provides a relational database

service that makes it easy to set up, operate, and scale a relational database. Google Apps SaaS cloud provides a set of services with functional interfaces, such as Google Drive that allows users to create, update and share documents.

Besides the functional aspects of a cloud service, there are also non-functional aspects related to the quality-of-service. There are different QoS aspects, such as *performance*, *availability*, *reliability*, *cost*, etc. For each QoS aspect, multiple QoS metrics may be considered. Examples of performance metrics are service *response time* that is the necessary time for a user request to get served, service *throughput* that reflects cloud service scalability, etc. Examples of availability metrics are service *abandon rate* that is the ratio of accepted service requests to the total number of requests, or service *use rate* that is the ratio of time a cloud service is used to the total time. Examples of reliability metrics are *mean time between failures* which is the predicted elapsed time between inherent failures of the service, or *mean time to recover* which is is the average time that a service takes to recover from a failure. Finally, examples of cost metrics are the *energetic cost* that reflects the energy footprint of a service, or the *financial cost* of using a cloud service.

Thus, a QoS metric is a means to quantify the service level with regard to a QoS aspect. One might want a service level to attain a given objective that is the *Service Level Objective (SLO)*. A SLO has usually one of the following forms: provide a QoS metrics with a value higher/lower than a given threshold, maximize/minimize the QoS metrics, etc. Therefore, a *Service Level Agreement (SLA)* is a set of SLOs to meet and is negotiated between two parties, the cloud service provider and its customer.

B. SLAaaS Model

We introduce *SLA-aware-Service* (SLAaaS), a new cloud model that defines a non-functional interface which exposes the SLA associated with a cloud functional service. Figure 1 illustrates the SLAaaS model at three cloud levels: an Infrastructure-as-a-Service cloud, a Platform-as-a-Service cloud and an example of a Software-as-a-Service cloud that represents here a business intelligence system. The example of this figure shows four levels: an end-user is a client of the SaaS cloud, which is itself a client of the PaaS cloud, which is itself a client of the IaaS cloud.

Roughly speaking, the functional interface of a cloud exposes operations that allow a cloud customer to get new resources from the cloud, to access/use resources in the cloud or to release resources that he/she does not use anymore. With SLAaaS, the cloud also exposes the SLA non-functional interface. Furthermore, SLAaaS aims to provide SLA-oriented cloud reconfiguration, and SLA governance. Due to space limitation, we focus on the former in the rest of the paper.

SLAaaS first allows a user to select the QoS aspects he/she is interested in (e.g. performance, cost), and the
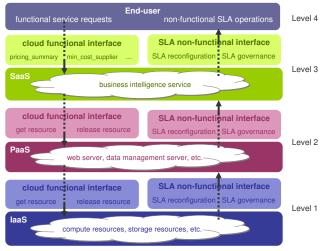

Figure 1. SLAaaS cloud model

QoS metrics for these aspects (e.g. service response time, financial cost). The user can then choose the SLOs he/she wants to apply on the QoS metrics. For instance, the SLO for the service response time might be to guarantee that the response time never exceeds a given threshold, and the SLO for the financial cost might be to guarantee that the cost is minimized. Then, the SLA is defined as the combination of SLOs. Furthermore, the SLA between a cloud service and its customer may include additional information, such as the agreed confidence level (e.g. SLOs are guaranteed with a confidence of 95%), or the penalties applied in case of SLA violation. Figure 2 presents three examples of SLAs that apply at three different cloud levels, between the end-user and the SaaS, between the SaaS and the PaaS, or between the PaaS and the IaaS.

SLA between the end-user and the SaaS cloud	
SLOs	For a maximum financial cost of US$ 0.10/request to the SaaS business intelligence service, response time must be less than 1 minute
Confidence	SLOs guaranteed on at least 95% of requests to the SaaS service
Penalty	If more than 5% of requests to the SaaS service violate SLOs, a penalty of US$ 0.20/violated request is applied
SLA between the SaaS and PaaS cloud	
SLOs	For a maximum financial cost of US$ 0.01/request to the PaaS data management service, response time must be less than 1 second
Confidence	SLOs guaranteed on at least 98% of requests to the PaaS service
Penalty	If more than 2% of requests to the PaaS service violate SLOs, a penalty of US$ 0.02/violated request is applied
SLA between the PaaS and IaaS cloud	
SLOs	For a maximum financial cost of US$ 0.12/resource.hour of the IaaS service, at least 7 GB of memory and at least 4 compute units must be available
Confidence	SLOs guaranteed on at least 99% of the time the client uses the IaaS service
Penalty	If during more than 1% of the time the IaaS service violates SLOs, a penalty of US$ 0.24/violated resource.hour is applied

Figure 2. Examples of SLAs at different cloud levels

In SLAaaS, the cloud SLA is defined with the CSLA language introduced in Section II-C, and the SLA is guaranteed following a control-theoretic approach, as described in Section II-D.

C. CSLA Specific Language

CSLA, the Cloud Service Level Agreement language, allows to describe a SLA between a cloud service provider and its customer by defining QoS guarantees in the form of SLO clauses [6]. The clauses are combined using *"and"* and *"or"* operators. Previous efforts to define SLA for web services (WSLA) [7] and service oriented architectures (SLA@SOI) [8] have influenced the design of this language.
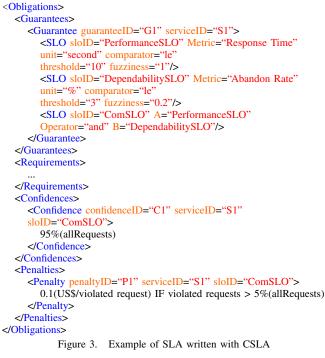
Among its novelties, CSLA integrates features dealing with QoS uncertainty and cloud fluctuations: *fuzziness*, *confidence*, and *penalty*. Fuzziness defines the acceptable margin degree around the target value of a SLO. Confidence is the compliance percentage of SLO clauses. Lastly, penalties are applied in case of SLA violations to compensate cloud service customers, i.e. penalties reduce the service price. The reduction can be applied either as a constant or variable rate. In the latter case, the request price is modeled as: $P = \alpha - \beta \cdot dt$ [9]; where α is the price with no violations ($\alpha > 0$), β is the penalty rate ($\beta > 0$) and dt is the absolute difference between the actual value and the SLO threshold. For example, if a SLO indicates a maximum response time of 3 s per request, with a request price $\alpha = \$0.8$, and a penalty rate $\beta = 0.5$, a request with response time of 4 s costs $P = 0.8 - 0.5 \cdot |4 - 3| = \$0.3$.

The CSLA syntax is defined according to the grammar generated from the meta-model in [6]. Based on the CSLA meta-model, the SLA can be defined in any language for any cloud service. In this paper, we use XML as a representation format. Figure 3 presents an example of a CSLA file describing the *Obligations* section for a SLA between a SaaS provider and its customer. Two SLOs are composed using the *"and"* operator, one is a performance SLO and the other, a dependability SLO. The performance SLO specifies that the request response time must be below 10 seconds, with an acceptable margin less than 1 second. The dependability SLO specifies that service abandon rate should not exceed 3% of incoming requests, with an acceptable margin of 0.2%. SLOs are guaranteed on at least 95% of requests to the cloud service (confidence). Thus, if more than 5% of the cloud service requests violate the SLOs, a reduction of $0.1 is applied in the price of each request violating the SLA.

According to a cloud service, a SLA template is generated with pre-defined parameters to ensure that offered QoS guarantees are realistic and realizable. Finally, once a SLA is described with CSLA and established between a cloud service provider and a cloud customer, it is passed to an online cloud controller as described in the next subsection.

D. Online Cloud Control

The online control of cloud services is based on a general feedback control loop as described in Figure 4. To manage cloud SLA in a principled way, we follow a control-theoretic approach to design fully autonomic SLA-oriented cloud services. The general approach consists in three main steps.

```
<Obligations>
    <Guarantees>
        <Guarantee guaranteeID="G1" serviceID="S1">
            <SLO sloID="PerformanceSLO" Metric="Response Time"
            unit="second" comparator="le"
            threshold="10" fuzziness="1"/>
            <SLO sloID="DependabilitySLO" Metric="Abandon Rate"
            unit="%" comparator="le"
            threshold="3" fuzziness="0.2"/>
            <SLO sloID="ComSLO" A="PerformanceSLO"
            Operator="and" B="DependabilitySLO"/>
        </Guarantee>
    </Guarantees>
    <Requirements>
        ...
    </Requirements>
    <Confidences>
        <Confidence confidenceID="C1" serviceID="S1"
        sloID="ComSLO">
            95%(allRequests)
        </Confidence>
    </Confidences>
    <Penalties>
        <Penalty penaltyID="P1" serviceID="S1" sloID="ComSLO">
            0.1(US$/violated request) IF violated requests > 5%(allRequests)
        </Penalty>
    </Penalties>
</Obligations>
```
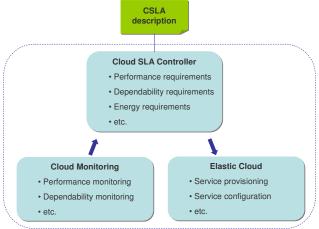
Figure 3. Example of SLA written with CSLA

Figure 4. Cloud autonomic reconfiguration

First, a utility function is defined to precisely describe the set of SLOs as specified in the cloud SLA, the weights assigned to these SLOs if any, and the possible trade-offs and priorities between the SLOs. The cloud service configuration (i.e. how many resources, what is their combination) with the highest utility is the best regarding SLA guarantees.

Then, control theory techniques are applied to model cloud service behavior, and propose control laws and algorithms for fully autonomic SLA-oriented cloud services. The challenges for modeling cloud services are to build accurate models that are able to capture the non-linear behavior of cloud services, and that are able to self-calibrate to render the variations of service workloads. The challenges for controlling cloud services are to propose accurate and efficient algorithms and control laws that calculate the best service configuration, and rapidly react to changes in cloud

service usage. The next section illustrates this approach to control online cloud services to guarantee their SLA.

III. Case Studies

We illustrate in this section how to build SLAaaS cloud services with three use cases: a MapReduce service, a multi-tier service, and a distributed locking service.

A. Experimental Environment

The experiments presented in this section were conducted in a cluster running on Amazon EC2 [3], and in two clusters running in Grid'5000 [10], see the hardware configuration in Table I. The underlying software configuration is as follows. Amazon EC2 instances run Fedora Linux 8 with kernel v2.6.21. Nodes in Grid'5000 (i.e. G5K I and G5K II) run Debian Linux 6 with kernel v2.6.32. Experiments of Section III-B use Apache Hadoop v1.0 MapReduce framework, Java 6, and the high-level MRBS benchmark suite [11]. Experiments of Section III-C use Apache Tomcat v7 web server, MySQL v.5.5.1 database server, and the TPC-W benchmark [12]. Finally, experiments of Section III-D are based on C++ and OpenMPI, and use micro-benchmarks.

Table I
HARDWARE CONFIGURATIONS

Cluster	CPU	Memory	Storage	Network
Amazon EC2	large instances, 4 EC2 Compute Units in 2 virtual cores	7.5 GB	850 MB	10 Gbit Ethernet
G5K I	4-core 2-CPU 2.5 GHz Intel Xeon E5420 QC	8 GB	136 GB SATA	1 Gbit Ethernet
G5K II	4-core 1-CPU 2.53 GHz Intel Xeon X3440	16 GB	278 GB SATA II	Infiniband 20G

B. SLAaaS-Oriented MapReduce PaaS

MapReduce is a programming model and a software framework to support distributed computing and large data processing on clusters of commodity machines [13]. High performance and fault-tolerance are two key features of MapReduce. They are achieved by automatic task scheduling in MapReduce clusters, automatic data placement, partitioning and replication, and automatic failure detection and task re-execution. A MapReduce job, i.e. an instance of a running MapReduce program, is automatically divided into multiple tasks scheduled by the MapReduce framework to run in parallel on cluster nodes. MapReduce is usually provided as a Platform-as-a-Service by cloud providers, such as Amazon and Azure. The functional interface of such a service includes operations such as starting a MapReduce cluster of a given size (i.e. #nodes), running a job on a MapReduce cluster, or stopping a MapReduce cluster.

We consider the case of a MapReduce PaaS that follows the SLAaaS model to illustrate the proposed approach. Thus, a SLA is contracted between the MapReduce PaaS and its customer. Figure 5 provides an example of the SLA. It specifies that the MapReduce job response time should

```
<Guarantees>
    <Guarantee guaranteeID="G1" serviceID= "MapReduce" >
        <SLO sloID="Rt" Metric="ResponseTime"
            unit="second" comparator="le" threshold= "90" fuzziness="0" />
        <SLO sloID="Rc" Metric="ResourceCost"
            unit="#nodes" comparator="min" />
        <SLO sloID="CompSlo" A="Rt" Operator="and" B="Rc" />
    </Guarantee>
</Guarantees>
```

Figure 5. SLA for MapReduce PaaS in CSLA language

not exceed 90 seconds, while the MapReduce cluster size (i.e. #nodes) should be kept as small as possible.

In order to guarantee the SLA, we applied a control-theoretic approach to provide a SLA-oriented self-elastic MapReduce cluster. Although some initiatives exist to add elasticity to MapReduce [14], [15], as far as we know, this is the first attempt to provide fully self-elastic MapReduce that is able to automatically adapt cluster size to workload variations in order to guarantee the SLA. To this purpose, the SLA is translated into a utility function in an ad hoc manner. First, the following boolean expression is defined to reflect whether the service performance SLO is met at a given time t:

$$PO(t) = \ell(t) \leq \ell_{max} \qquad (1)$$

where $\ell(t)$ is the average MapReduce job latency (i.e. response time) at time t, and ℓ_{max} is the maximum job latency not to exceed. Note that $\forall t, PO(t) \in \{0,1\}$, depending on whether Eq. (1) holds or not. Then, the utility function combines both performance and cost (cluster size) objectives:

$$\theta(t) = \frac{PO(t)}{\omega(t)} \qquad (2)$$

where $\omega(t)$ is the MapReduce cluster size at time t. Here, $\forall t, \theta(t) \in [0,1]$. Intuitively, the MapReduce cluster with the highest utility is the one that guarantees the performance SLO (if possible) with minimal cluster size.

Then, the MapReduce cluster is modeled following a queuing network approach, where each queue represents a cluster node and is modeled as an M/M/c queue. Here, client communication with the MapReduce service is modeled as a closed loop to reflect the synchronous communication model that underlies this service, that is a client waits for a request response before issuing another request. Moreover, multiple clients may concurrently request the service (i.e. execute MapReduce jobs). The model predicts the average request latency based on the monitored workload and service cluster size. The workload is defined as the number of concurrent clients and the average response time for the requests. Then, a capacity planning is applied to calculate the MapReduce cluster size with the highest utility, and to apply it to the online MapReduce service. The used model and capacity planning are adaptations from our previous work on Internet services to MapReduce services [16]. The model allows the capacity planning to find the exact number of nodes needed to guarantee the SLA, instead of adding/removing nodes one by one. Moreover, monitoring windows to measure the workload and to react to changes in the workload are

ensured to be long enough to let the system stabilize after adding/removing nodes.

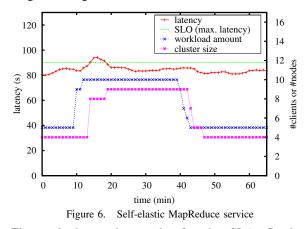Figure 6.   Self-elastic MapReduce service

Figure 6 shows the results for the SLAaaS-oriented MapReduce PaaS running on Amazon EC2. MRBS [11] was used to stress the MapReduce service. The setup consists of a set of nodes hosting the MapReduce cluster, an additional node hosting MRBS emulating MapReduce clients, and another node for the cloud SLA controller. Among the different benchmarks provided by MRBS, the movie recommender system benchmark was used in this use case. It builds upon a set of movies, a set of users, and a set of ratings and reviews users give for movies to indicate whether and how much they liked or disliked the movies. A client can request the top-10 recommendations for him/her, all the ratings given to a movie, how much the client would like a movie, or all the ratings given by another client. Each request is randomly chosen, and after receiving the response, the client submits another request. Our experiments are based on the following set of real data: 1700 movies, 1000 users, 100,000 ratings [17].

The MapReduce service initially runs on a four node cluster, and the service is warmed up for 10 minutes with 5 clients and then, measures are taken during 65 minutes as shown in Figure 6. The service state (#clients and average response time) is monitored every minute and the capacity planning is executed every 3 minutes taking the average of the service states in the last 5 minutes. We vary the number of concurrent clients over time between 5 and 10 as shown in the figure. When additional clients access the service, client request response times increase until the SLA is violated at time 13 minutes. Nevertheless, the automatic self-elastic MapReduce service adapts and increases its capacity to guarantee the SLA again. Finally, when the workload decreases after minute 40, the automatic self-elastic service releases underused nodes. Thus, this experiment shows that SLAaaS successfully applies to an online MapReduce service to guarantee performance- and cost-oriented SLA. In this case study, we considered the resource cost metrics (i.e. #nodes) in the SLA. In a future work, we will consider the financial cost which relies on both resource cost and units of time (usually hours) during which the resource is used.

C. SLAaaS-Oriented Multi-Tier Bookstore SaaS

In this case study, we apply the SLAaaS model to the TPC-W online bookstore Software-as-a-Service [12]. This service follows a multi-tier architecture consisting of a front-end web tier and a back-end database tier. For scalability purposes, each tier may consist of many server instances. Intuitively, the higher the number of instances in each tier, the better the performance and availability of the service. However, the number of instances hosting a cloud service has a direct impact on the service cost, and actually depends on the current service workload.

Figure 7 presents an example of SLA established between the multi-tier bookstore SaaS and its customers. This contract combines performance, availability and cost SLOs as follows: request response time should not exceed 500 ms and at least 95% of client requests should be served, with a number of instances hosting the service as small as possible.

```
<Guarantees>
  <Guarantee guaranteeID="G1" serviceID="MultiTierBookstore" >
    <SLO sloID="Rt" Metric="ResponseTime"
      unit="millisecond" comparator="le" threshold="500" fuzziness="0" />
    <SLO sloID="Av" Metric="Availability"
      unit="% of req." comparator="ge" threshold="95" fuzziness="0" />
    <SLO sloID="Rc" Metric="ResourceCost"
      unit= "#nodes" comparator="min" />
    <SLO sloID="PerfAvailSlo" A="Rt" Operator="and" B="Av"/>
    <SLO sloID="CompSlo" A="PerfAvailSlo" Operator="and"B="Rc" />
  </Guarantee>
</Guarantees>
```

Figure 7.   SLA for multi-tier bookstore SaaS in CSLA language

First, a utility function is drawn ad hoc from the SLA. The following boolean expression reflects whether the service performance SLO and the service availability SLO are met at a given time t:

$$PAO(t) = (\ell(t) \leq \ell_{max}) \cdot (\alpha(t) \leq \alpha_{max}) \qquad (3)$$

where $\ell(t)$ is the average request latency (i.e. response time), ℓ_{max} the maximum request latency not to exceed, $\alpha(t)$ the service availability (i.e. ratio of non-rejected requests), and α_{max} is the minimum service availability to be guaranteed. Note that $\forall t, PAO(t) \in \{0, 1\}$, depending on whether Eq. (3) holds or not. Then, the utility function combines performance, availability and cost (#nodes) objectives:

$$\theta(t) = \frac{T \cdot PAO(t)}{\omega(t)} \qquad (4)$$

where $\omega(t)$ is the number of nodes that host the multi-tier SaaS at time t, and T is the number of tiers of the multi-tier service ($T = 2$ in TPC-W). T is used in Eq. (4) for normalization purposes. Here, $\forall t, \theta(t) \in [0, 1]$, since $\omega(t) \geq T$ (at least one instance per tier) and $PAO(t) \in \{0, 1\}$.

The multi-tier service is then modeled following a queuing network approach, where each queue represents a server replica and is modeled as an M/M/c/K queue, and the network of queues represents the series of tiers in a multi-tier service. The model predicts the client request latency

(a) Service performance

(b) Service availability

(c) Online control of cloud instances

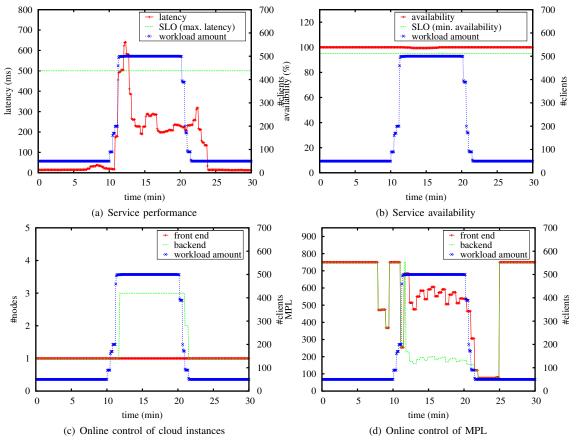(d) Online control of MPL

Figure 8.   Self-elastic multi-tier bookstore service

and service availability, based on the service workload, the multi-tier service size and the admission control level (MPL: Multi-Programming Level) usually applied on each tier of a multi-tier service. The workload consists of the number of concurrent clients, the average request response time and the visiting ratio (i.e. #requests in the back-end per request in the front-end). Then, the capacity of the multi-tier service that provides the highest utility (Eq. (4)) is calculated, and applied to the online service. Due to space limitations, the model and capacity planning algorithms are not detailed here, more information can be found in [16]. As for the previous case study, each run of the capacity planning finds the needed service size and it is ensured that the system stabilizes between two consecutive executions of the capacity planing.

The experiments on the online multi-tier bookstore service were run in G5K I (see Table I), with a read-only version of browsing-mix, a workload specified by TPC-W. Figure 8 depicts the results considering the SLA given in Figure 7. The number of concurrent clients was varied from 50 to 500 and then to 50 again. The service state (#clients, average response time and ratio of rejected requests) is monitored every 5 seconds and the capacity planning runs every minute using the average of the service states in the last 2 minutes.

Initially, the online service is composed of one instance

for the web tier, and one instance for the database tier. There is also another node that runs the SLA controller. The SLA is violated when the number of concurrent clients increases to 500 (see Figure 8(a)), that triggers the reconfiguration of the cloud service creating two new instances in the database tier and adjusting the MPL as shown in Figures 8(c) and 8(d). Once the reconfiguration of the cloud service has been applied, the service is able to cope with the SLA requirements again (see Figure 8(a) and 8(b)). Finally, when the load decreases, the system is over-provisioned and some instances are released and the MPL adjusted accordingly. Therefore, the SLAaaS-oriented multi-tier service is able to successfully guarantee SLA despite workload variations.

D. SLAaaS-Oriented Locking PaaS

Locking allows to ensure exclusive access to shared resources by concurrent processes, and is usually provided as a Platform-as-a-Service in the cloud. For instance, Google provides the Chubby distributed locking mechanism that is used by other cloud services such as Google Filesystem service and Bigtable data storage service [18]. Such a mechanism provides a functional interface with operations to acquire or release locks, among others. However, locking procedures remain costly. Locking was identified as an important and poorly resolved problem [19]; these protocols

have to be scalable and take into account QoS objectives.

We apply the SLAaaS model to a locking PaaS to illustrate our proposed approach. Thus, a SLA is contracted between the locking service and its customer. Figure 9 gives an example of SLA that combines performance and availability objectives. The SLA specifies that the response time of a request to the lock service should not exceed 400 ms. It also specifies that the usage of the locked shared resource is kept as high as possible. This is translated ad hoc into a utility function:

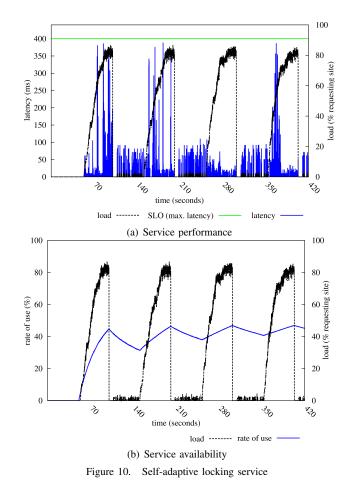$$\theta(t) = \frac{PO(t)}{\rho(t)} \qquad (5)$$

where $PO(t)$ is given in Eq. (1), and $\rho(t)$ is the use rate of locked resource. Intuitively, the locking service with the highest utility is the one that guarantees the performance SLO (if possible) with a high resource use rate, and therefore, the SLA. Then, we combine admission control techniques with a distributed locking algorithm in order to guarantee the SLA [20]. Thus, before accepting a request, the locking service controller first verifies that, taking into account current system state, the performance SLO can be satisfied. If so, the request for lock acquisition is accepted and will be served; otherwise, the request is rejected. Due to space limitations, algorithm details are not provided but can be found in [20]. In the present paper, we show how the locking algorithm is integrated with the SLaaS model.

```
<Guarantees>
    <Guarantee guaranteeID="G1" serviceID= "LockingService" >
        <SLO sloID="Rt" Metric="ResponseTime" unit="millisecond"
            comparator="le" threshold= "400" fuzziness="0" />
        <SLO sloID="Ru" Metric= "ResourceUsage" unit="% of time"
            comparator= "max" />
        <SLO sloID="CompSlo" A="Rt" Operator="and" B="Ru"/>
    </Guarantee>
</Guarantees>
```

Figure 9.   SLA for locking PaaS in CSLA language

We conducted experiments with our SLAaaS-oriented locking service, running in a 40 node cluster in the G5K II infrastructure (see Table I). To emulate long distance, we injected network latency between nodes. Each node runs a process that may request to acquire the lock on a shared resource. The load varies over time, and is characterized by the ratio of processes requesting lock acquisition to the total number of processes, as shown in Figure 10. Figure 10(a) presents lock request response time over time. When the load is low, the response time remains low compared to the SLO. When the load increases, there is more contention on the shared resource, with an increase of lock request latency. However, the locking service is able to automatically adapt to keep request latency below the threshold as specified by the SLA. This is obtained thanks to admission control.

Figure 10(b) illustrates the use rate of the shared resource, i.e. how often the resource is actually locked and used by one of the processes. It shows the time ratio during which the resource is used by processes to the total time. In our network configuration this ratio cannot exceed 50% since

(a) Service performance

(b) Service availability

Figure 10.   Self-adaptive locking service

half of time is spent in message transmission. Interestingly, when the load increases the locking service adapts to the load, with an increasing use rate until the maximum value, which corresponds to the availability objective of the underlying SLA. In summary, SLAaaS successfully applies to associate SLA with a locking PaaS.

IV. RELATED WORK

Existing public clouds provide very few guarantees in terms of performance and dependability [2]. Amazon EC2 compute service offers a service availability of at least 99.95% [3], and Amazon S3 storage service guarantees a service reliability of 99.9% [3]. However, in case of an outage, Amazon requires the customer to send them a claim within thirty business days for Amazon EC2 and ten days for Amazon S3. Amazon cloud services do not provide performance guarantees or other QoS guarantees. Rackspace and Azure cloud services provide similar behaviors [4], [5].

Several recent research works consider SLA in cloud environments [21], [22], [23], [24]. Chhetri *et al.* propose the automation of SLA establishment based on a classification of cloud resources in different categories with different costs, e.g. on-demand instances, reserved instances and spot instances in Amazon EC2 cloud [21]. However, this approach does not provide guarantees in terms of performance, nor

dependability. Macias and Guitart follow a similar approach for SLA enforcement, based on classes of clients with different priorities, e.g. Gold, Silver, and Bronze clients [22]. Here again, a relative best-effort behavior is provided for clients with different priorities, but neither performance nor dependability SLOs are guaranteed. Other works propose heuristics for SLA management [23], or target specific environments such SaaS [24]. The former work provides best-effort without strict guarantees on SLA, and the latter does not tackle the many types of clouds.

Regarding the specification of SLA, some initiatives contributed to this effort, such as WSLA [7], and WS-Agreement [25]. The proposed CSLA language shares motivations with these projects and goes further by taking into account high cloud elasticity and QoS instability. Its general concepts were introduced in [6]; in the present paper we describe its integration with the SLAaaS model and its application to real cloud services.

V. Conclusion

This paper presents SLA-aware-Service (SLAaaS) cloud model, for a systematic and principled way to integrate quality-of-service (QoS) and service level agreement (SLA) into the cloud. The CSLA specific language is proposed to describe SLAs associated with cloud services in a convenient way. A control-theoretic approach is followed to provide performance, dependability and cost guarantees for online services. Our experiments on online cloud services through various case studies successfully demonstrate the usefulness of SLAaaS. While this paper illustrates SLA with QoS metrics such as client request response time, availability, resource usage and resource cost, we believe that the proposed model and control approach may apply to other metrics, such as service throughput, and energetic cost. This work opens interesting perspectives in terms of cooperative clouds and cooperative SLAs. We hope that such a model will lead to more principled, less ad-hoc solutions of cloud QoS and SLA management.

References

[1] L. M. Vaquero, L. Rodero-Merino, J. Caceres, and M. Lindner, "A Break in the Clouds: Towards a Cloud Definition," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 1, Jan. 2009.

[2] S. A. Baset, "Cloud SLAs: Present and Future," *ACM SIGOPS Operating Systems Review*, vol. 46, no. 2, Jul. 2012.

[3] "Amazon Web Services," aws.amazon.com/, 2012.

[4] "Rackspace," www.rackspace.com/cloud/, 2012.

[5] "Windows Azure," www.microsoft.com/windowsazure, 2012.

[6] Y. Kouki and T. Ledoux, "CSLA: a Language for Improving Cloud SLA Management," in *2nd Int. Conf. on Cloud Computing and Services Science (CLOSER)*, 2012.

[7] H. Ludwig, A. Keller, A. Dan, R. P. King, and R. Franck, "Web Service Level Agreement (WSLA) Language Specification," IBM, Tech. Rep., 2003.

[8] "Sla@soi," sla-at-soi.eu/, 2012.

[9] D. Irwin, L. Grit, and J. Chase, "Balancing Risk and Reward in a Market-based Task Service," in *13th IEEE Int. Symp. on High Performance Distributed Computing (HPDC)*, 2004.

[10] R. Bolze *et al.*, "Grid'5000: A Large Scale and Highly Reconfigurable Experimental Grid Testbed," *Int. J. High Performance Computing Applications (IJHPCA)*, vol. 20, no. 4, Nov. 2006.

[11] A. Sangroya, D. Serrano, and S. Bouchenak, "Benchmarking Dependability of MapReduce Systems," in *31st IEEE Int. Symp. on Reliable Distributed Systems (SRDS)*, 2012.

[12] Transaction Processing Performance Council, "TPC-W," www.tpc.org/tpcw.

[13] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," in *6th USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, 2004.

[14] Z. Fadika and M. Govindaraju, "DELMA: Dynamically ELastic MapReduce Framework for CPU-Intensive Applications," in *11th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing (CCGrid)*, 2011.

[15] A. W. Gordon and P. Lu, "Elastic Phoenix: Malleable MapReduce for Shared-Memory Systems," in *8th IFIP Int. Conf. on Network and Parallel Computing (NPC)*, 2011.

[16] J. Arnaud and S. Bouchenak, *Performance and Dependability in Service Computing*. IGI Global, 2011, ch. Performance, Availability and Cost of Self-Adaptive Internet Services.

[17] "MovieLens web site," http://movielens.umn.edu/.

[18] M. Burrows, "The Chubby Lock Service for Loosely-Coupled Distributed Systems," in *7th USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, 2006.

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

[20] J. Lejeune, L. Arantes, J. Sopena, and P. Sens, "Service Level Agreement for Distributed Mutual Exclusion in Cloud Computing," in *12th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing (CCGrid)*, 2012.

[21] M. B. Chhetri, Q. B. Vo, and R. Kowalczyk, "Policy-Based Automation of SLA Establishment for Cloud Computing Services," in *12th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing (CCGrid)*, 2012.

[22] M. Macias and J. Guitart, "Client Classification Policies for SLA Enforcement in Shared Cloud Datacenters," in *12th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing (CCGrid)*, 2012.

[23] H. Goudarzi, M. Ghasemazar, and M. Pedram, "SLA-based Optimization of Power and Migration Cost in Cloud Computing," in *12th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing (CCGrid)*, 2012.

[24] L. Wu, S. K. Garg, and R. Buyya, "SLA-Based Resource Allocation for Software as a Service Provider (SaaS) in Cloud Computing Environments," in *11th IEEE/ACM Int. Symp. on Cluster, Cloud and Grid Computing (CCGrid)*, 2011.

[25] A. Andrieux, "Web Services Agreement Specification (WS-Agreement)," OGF, Tech. Rep., 2007.