An Elasticity-aware Governance Platform for Cloud Service Delivery

Carlos Müller*, Hong-Linh Truong◇, Pablo Fernandez*, Georgiana Copil◇, Antonio Ruiz–Cortés*, Schahram Dustdar◇
*ISA research group, University of Seville, {cmuller, pablofm, aruiz}@us.es.
◇Distributed Systems Group, TU Wien, {truong, e.copil, dustdar}@dsg.tuwien.ac.at.

***Abstract*—In cloud service provisioning scenarios with a changing demand from consumers, it is appealing for cloud providers to leverage only a limited amount of the virtualized resources required to provide the service. However, it is not easy to determine how much resources are required to satisfy consumers expectations in terms of Quality of Service (QoS). Some existing frameworks provide mechanisms to adapt the required cloud resources in the service delivery, also called an elastic service, but only for consumers with the same QoS expectations. The problem arises when the service provider must deal with several consumers, each demanding a different QoS for the service. In such an scenario, cloud resources provisioning must deal with trade-offs between different QoS, while fulfilling these QoS, within the same service deployment. In this paper we propose an elasticity-aware governance platform for cloud service delivery that reacts to the dynamic service load introduced by consumers demand. Such a reaction consists of provisioning the required amount of cloud resources to satisfy the different QoS that is offered to the consumers by means of several service level agreements. The proposed platform aims to keep under control the QoS experienced by multiple service consumers while maintaining a controlled cost.**

I. Introduction

The delivery of cloud services is a nontrivial task for service providers that usually requires to provision an estimated amount of the required cloud resources (both infrastructural and software resources) to satisfy consumers demand. A precise estimation of such resources would benefit cloud providers with a lower resources provisioning cost. However, such an estimation is a challenging task that depends on: (i) the number of consumers, (ii) their variable demand, and (iii) their expectations in terms of Quality of Service (QoS).

In such a context, service providers can benefit from on-demand capabilities of cloud environments, using their elasticity capabilities (e.g., add/remove storage, or change the load distribution mechanism) to scale the resources and to configure the software as needed by their services. In this sense, we can find some approaches [1], [2], [3] providing mechanisms to adapt the provisioned resources depending on the system behaviour and specified QoS requirements. Specifically, in [2] authors propose to consider such QoS within elasticity strategies in scalable cloud service topologies.

However, the aforementioned proposals provide elastic services considering the service behaviour for one or more consumers with the same QoS expectation. They do not address the difficulties of dealing with multiple consumers, each with a different QoS expectation. In this paper, first we analyze challenges that need to be addressed. Second, we develop a cloud governance platform to tackle the challenges. Finally, we evaluate the benefits of our work through several test cases using our prototype.

The challenges we face in the paper are aligned with those exposed in [4], [5], [6], where the authors claim that a dynamic and self-managed service provisioning would require service-oriented features such as: service-level monitoring capabilities to control how service constraints, Service Level Agreements (SLAs), and elasticity strategies are fulfilled; SLA management functionality to guarantee a service's performance and reliability; and service elasticity management capabilities to guarantee that the service can scale automatically. Addressing them will enable capabilities to automate, optimize, govern operational business decisions, as pointed out by DiMarzio, in real-world operational systems[1].

The governance platform we develop in the paper tackles the exposed challenges, coordinates the cloud service delivery while limiting the cloud resource provisioning cost. Our developed platform supports two of the QoS properties that are most commonly used within commercial SLAs: the service availability and throughput (i.e., supported operations per seconds) (c.f. SLAs and pricing description of services provided by companies such as Amazon[2], Google[3], or Microsoft[4]). Finally, in the paper we show that by using our developed governance platform a cloud provider achieves the following benefits altogether: (i) the service to a number of consumers with different QoS expectations; and (ii) the resources provisioning to the consumers demand, which means to avoid under and over provisioning situations that take place when not enough or too much resources are leveraged to satisfy consumers demand, respectively.

This paper is structured as follows: Section II introduces the motivating scenario and challenges. Section III exposes the conceptual architecture of our platform. Section IV presents our elasticity-aware governance tecnniques. Section V validates our platform through various experiments. In Section VI we present related work. Finally, in Section VII we conclude the paper and outline future work.

This work was partially supported by the Spanish and the Andalusian R&D&I programmes and networks (grants P12-TIC-1867, TIN2012-32273, TIN2014-53986-REDT).

[1] https://goo.gl/Rxb8Wt
[2] SLA: http://goo.gl/v6AWzG; Pricing: http://goo.gl/ZT7Z5a
[3] SLA: http://goo.gl/DS5N4J; Pricing: http://goo.gl/fJQHpo
[4] SLA: http://goo.gl/e9aza4; Pricing: http://goo.gl/9l4yrp

978-1-5090-2628-9/16 $31.00 © 2016 IEEE
DOI 10.1109/SCC.2016.17

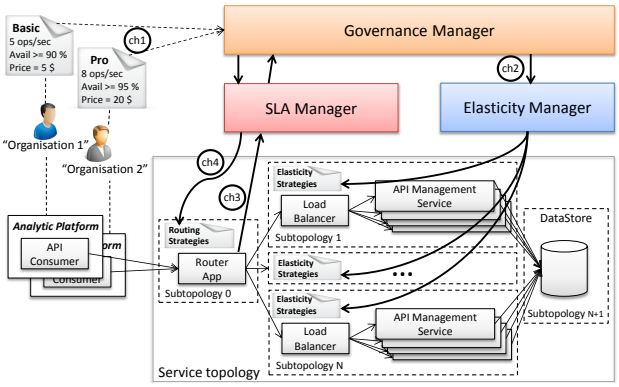

Fig. 1. Scenario with multiple consumer and research challenges.

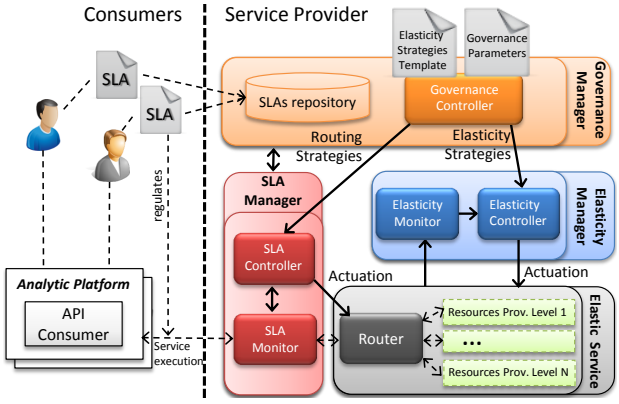

Fig. 2. Design of the Elasticity-aware SLA Governance platform.

II. Motivating Scenario

As motivating scenario we consider the provisioning of a cloud service for gathering sensitive data from a *data store*, through an *API management service* (c.f. Figure 1). In order to provide an SLA to the consumer in which several QoS properties such as a minimum availability and throughput are guaranteed, the cloud service provider must provision appropriate cloud software resources structured in a service topology [2]. Specifically, cloud objects such as virtual machines accessible through the API management service, or nodes in the data store are leveraged in the service topology of Figure 1, if needed.

Aiming to control resources provisioning, we can benefit from the work in [2]: (i) to monitor the service behaviour and (ii) to specify some elasticity strategies to scale in or out the service topology under certain conditions (e.g. scale out if the monitored throughput $<$ 200 operations per seconds). In our motivating scenario we have several service *subtopologies* with *load balancers* that makes the decision on scaling in or out the instances of API management service to fulfill the consumer SLA. Moreover, in the scenario we consider a new way to enable elasticity by routing the consumer requests to a corresponding subtopology 1 to N, when more resources are needed. Although, elasticity is a powerful mechanism in scenarios fulfilling just an SLA, these elasticity techniques are not focused on satisfying multiple consumers when using a single service with a different SLA each. For instance, as depicted in Figure 1, several organisations can be interested in gathering the information with a different SLA (c.f. basic and pro SLAs in Figure 1).

The following challenges arise when multiple consumers are considered in the previous scenario. Firstly, the service provider must consider in the cloud service offering that each consumer may have a different SLA (*ch1* in Figure 1). Secondly, the service provider must configure the subtopologies with appropriate elasticity strategies considering the different SLAs guaranteed to consumers (*ch2* in Figure 1). Thirdly, the service behaviour must be monitored to control if the service level provided to each and every consumer is as they agreed (*ch3* in Figure 1). Finally, the cloud resources provisioning should be optimized according to the previously mentioned monitoring information and the consumers demand. In this sense, several subtopologies with an appropriate cloud service provisioning should be created. In addition, some routing strategies must be established to redirect consumers requests to one of such subtopologies (*ch4* in Figure 1).

III. Design of Elasticity-aware SLA Governance

Our proposal to tackle the exposed challenges is a novel elasticity-aware governance architecture depicted in Figure 2. The underlying idea is an orchestration between a Governance Manager (GM), an Elasticity Manager (EM), and an SLA Manager (SM) to analyse the actual service behaviour and to react against a potential under or over provisioning situation without violating the consumers SLAs. Such a reaction consists in applying some routing and elasticity strategies when the monitored values goes through specific thresholds. As consequence, it is leveraged an optimised amount of resources, while the actual offered QoS of the service is under control.

The GM relies on a governance configuration information comprising: (i) the *SLAs* with the QoS offerings to the consumers; (ii) an *elasticity strategies template* including required cloud objects (e.g. load balancers or data store in the motivating scenario), and (iii) the *governance parameters* that set up some important aspects to govern the service delivery and monitoring. Section IV details how we deal with such configuration information.

Two main components of the GM are an *SLA repository* and a *governance controller*. The SLA repository is responsible for storing and retrieving the SLAs that the service must fulfill for each consumer. The governance controller is responsible for setting up both: the cloud service elasticity; and the routing of consumers requests between the resources provisioning levels. In order to set up the elasticity, the governance controller, as described in Section IV-A, firstly considers information from the SLAs to create the elasticity strategies that specify when to scale in or out (i.e. ch2 is tackled); and secondly, it establishes some routing strategies to redirect consumers requests to the appropriate resources provisioning level. Note that each provisioning level represents a service subtopology (c.f. Figure 1) which leverages a specific amount of resources.

The main components of the SM are an *SLA controller* and

```
Agreement ProSLA version 1.0
 Provider CloudServiceProvider as Responder;
 Consumer "Organisation 2";
 Metrics [iAgree.generalMetrics]

AgreementTerms
 Service CloudServiceDescription
  Global description:
   Price = 20 usd/m;
   Throughput <= 8 ops/sec

 Monitorable Properties
  global:
   Availability;
   Throughput;

 Guarantee Terms
  G1: Provider guarantees Availability >= 95;
```

Fig. 3. Pro SLA specified with iAgree.

an *SLA monitor*. The SLA controller is responsible for: managing the SLAs with the QoS offerings to the consumers (i.e. ch1 is tackled); routing consumers requests to the appropriate resources provisioning level established in routing strategies (c.f. actuation message from SLA controller to the router in Figure 2); but it also receives from the SLA monitor the current behaviour provided to consumers in order to detect a potential under or over provision situation. Moreover, aiming to solve an under-provision, the SLA controller must check if there exist risk of violating the SLA terms of an specific consumer (i.e. ch3 is tackled). If this is the case, the SLA controller must increase the resource provisioning to restore an appropriate service behaviour (i.e. ch4 is tackled).

In turn, the main components of the EM are an *elasticity controller*, and an *elasticity monitor*. Elasticity controller responsibilities are: (i) to adapt the deployed provisioning levels with the elasticity strategies provided by GM; and (ii) to analyse if the elasticity strategies are being fulfilling or not at service consumption time in order to scale in or out the resources within the provisioning levels. In turn, the elasticity monitor is responsible for gathering the actual service behaviour in each provisioning level. Note that the elasticity controller component is based on the previous work [2] and it does not constitute one of the main contributions of this paper.

IV. Governance by Leveraging Elasticity

In this section we explain our proposal to govern both: the elasticity within each provisioning level; and the routing of consumers requests to the appropriate provisioning level. Specifically, in Section IV-A we explain how to manage the elasticity within each provisioning level; and in Section IV-B we propose an SLA management to satisfy the consumers demand and SLAs by routing their requests to the suitable provisioning levels. It is worth to highlight that we consider in our approach SLAs defining a guarantee on the service availability and a maximum throughput. For instance, Figure 3 includes the pro SLA in the motivating scenario, written in iAgree, a WS–Agreement-based language [7], [8]. In further research we will analyse how to generalise our proposal to consider more kinds of SLAs.

The governance controller must be provided of some governance parameters that set up the aforementioned elasticity and SLA management. These governance parameters comprise: (i) the time intervals in which the service behaviour is monitored (e.g. 1 hour to compute current availability; and 30 seconds to compute current throughput) and the routing actuation is updated (e.g. 2 seconds to route consumers requests to the corresponding provisioning level); (ii) two or more provisioning levels with an elasticity strategy each, are needed to optimise the resource provisioning according to the consumers demand; and (iii) the routing strategies that guide when a consumer request must be routed to a specific provisioning level in order to avoid violating its SLA, or wasting resources.

A. Elasticity Management

Aiming to govern how to scale in or out the resources within each resources provisioning level, the elasticity controller must establish specific elasticity strategies that considers the SLA terms. In our approach, we use the models depicted in Figure 4, i.e. a simplified version of WS–Agreement [9] for SLAs, and SYBL [2] for the elastic service topology, that includes the elasticity strategies.

As Figure 4 denotes, the elasticity strategies of the SYBL service topology must be defined on monitorable properties of the SLA. The reason is the value of a monitorable property (e.g. the throughput) changes meanwhile the service is being consumed and therefore, it can be monitored if in an specific instant its value is acceptable for an specific resources provisioning level or it must be scaled out or in.

In this sense, for the kind of SLAs we use in our approach, we propose to define the elasticity strategies as follows depending on the throughput provided by the platform.

Definition 1 (Elasticity Strategies)**.** *An elasticity strategy E can be denoted as a pair of values for the service throughput $minTh$ and $maxTh$ under and over which a provisioning level must be scaled in and out, respectively. Thus, it must be denoted as $E = (minTh, maxTh)$.*

The formula we use for the *minTh* and *maxTh* depends on: (i) the variable number of virtual machines (VMs) used in the provisioning level; (ii) the maximum throughput supported by each VM (Th_VM), which does not vary over the time; and (iii) a pre-established elasticity speed (elastSpeed) parameter that ranges between 0 and 1 and denotes the speed of performing an scaling action (i.e. values near to 1 implies a slower scaling). The elasticity speed influences the agility of the service to adapt to drastic workload changes. Thus, the lower elasticity speed, the lower agility of the service to adapt to drastic workload changes is, and viceversa. In addition, when such drastic workload changes take place, lower elasticity speeds represent to increase the risk of violating an SLA term. Choosing an optimal elasticity speed depending on certain circumstances (e.g. the workload and scaling action to perform) requires a thorough study that is out of the scope of current paper. In current work we have empirically gathered an elasticity speed value of 20% that optimse the cost of developed infrastructure (c.f. Section V-B).

$$maxTh = Th\_VM * (1 + elastSpeed) * VMs$$
$$minTh = Th\_VM * (1 + elastSpeed) * (VMs - 1)$$

Figure 5 depicts an excerpt of the SYBL document used to specify the topology in the motivating scenario. Note that the excerpt includes the elasticity strategies for a resources

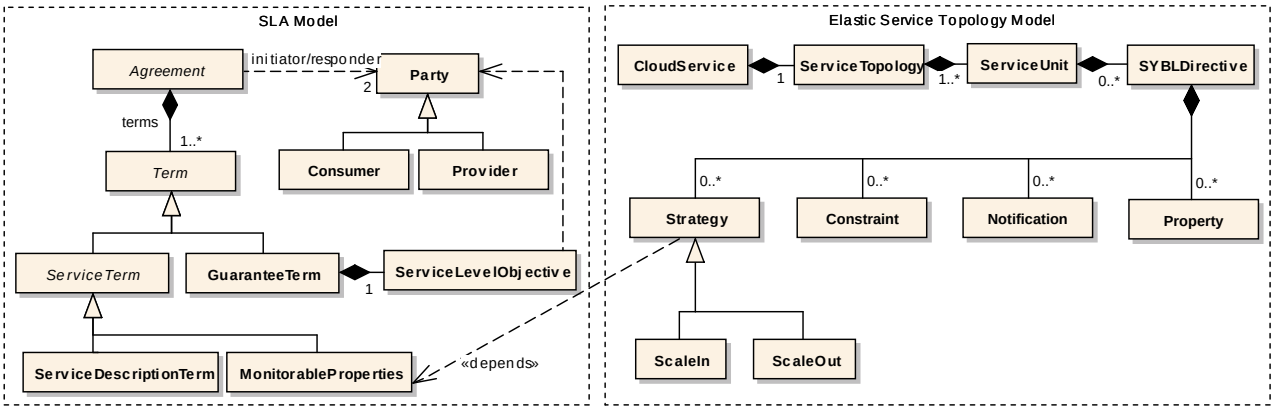

Fig. 4. SLA model and Elastic Service Topology model.

```
<CloudService id="APIManagementService">
 <ServiceTopology id="scenarioTopology">
  <ServiceUnit id="DataStoreUnit"/>
  <ServiceUnit id="ProvLevel1APIUnit">  <!-- Strategies -->
   <SYBLDirective Strategies="
    DN_ST2:STRATEGY CASE Th_in > Th_VM * (1 + elastSpeed)
                                      * VMs #:scaleOut;
    DN_ST2:STRATEGY CASE Th_in <= Th_VM * (1 + elastSpeed)
                                  * (VMs - 1) #:scaleIn"
    Constraints=""...>
  </...>
  <ServiceUnit id="ProvLevel1LoadBalancerUnit"/>
 </...>
</...>
```

Fig. 5. Excerpt of SYBL topology with elasticity strategy for a prov. level.

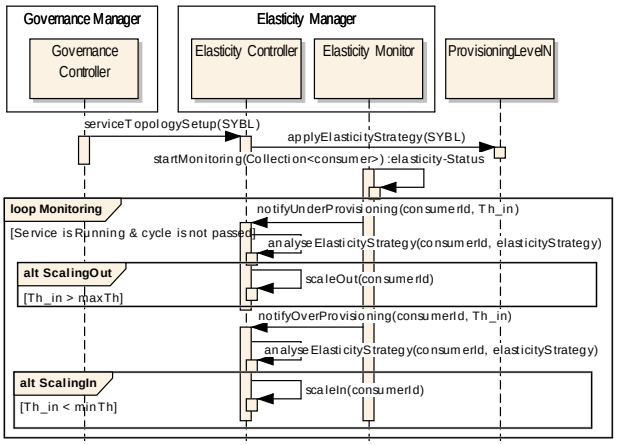

Fig. 6. Elasticity management within resources provisioning levels.

provisioning level depending on the incoming throughput that the platform receives (*Th_in*). As the SYBL notation, as the SYBL tooling support have been extended to support the formula specification included in the elasticity strategies. The other service units of the SYBL document such as the data store and the load balancer must be previously established by a topology expert (c.f. the initial setup message in Figure 6).

Once the elasticity strategies are established, the elasticity controller must actuate on each provisioning level accordingly. Thus, as depicted in sequence diagram of Figure 6, the elasticity monitor must detect if an under or over provision situation takes place while the service is being consumed. In other words, it must be controlled if the *maxTh* or *minTh* have been exceeded by the incoming throughput *Th_in* (c.f. conditions of *ScalingIn* and *ScalingOut* alt blocks of Figure 6). In case of under-provisioning situations the elasticity controller must perform an scale out to leverage more resources. To the contrary, in case of over-provisioning situations the elasticity controller must perform an scale in action to free resources.

B. SLA Management

Aiming to fulfill the SLA of consumers in the service delivery while optimising the resources provisioning, the governance platform performs an SLA management in parallel with the aforementioned elasticity management. The main goal of such an SLA management is to route each consumer request to an appropriate resources provisioning level. Such an appropriate routing is performed through the routing strategies that help the SLA controller to evaluate if the QoS experienced by each consumer satisfy or not its SLA. For the kind of SLAs we support in the current work, the routing strategies can be defined as follows.

Definition 2 (Routing Strategies)**.** *A routing strategy R can be denoted as a pair of values for the service availability $minAv$ and $maxAv$ under and over which a given consumer C_i must be routed to a lower or a higher provisioning level, respectively. Thus, it must be denoted as $R = (minAv, maxAv)$.*

The formula we use for the *minAv* and *maxAv* depends on: (i) the SLO specified in the SLA for the availability property; (ii) the maximum value for the availability property (e.g. 100% in general); and (iii) two pre-established routing speed parameters that ranges between 0 and 1 and denote the speed of performing a routing action for a consumer. That is, either increasing (*incRoutSpeed*) or decreasing (*decRoutSpeed*) a provisioning level for a consumer (i.e. values near to 1 implies a faster routing action).

$$maxAv = (decRoutSpeed * (Max_{Av} - SLO_{Av})) + SLO_{Av}$$
$$minAv = (incRoutSpeed * (Max_{Av} - SLO_{Av})) + SLO_{Av}$$

For instance, the routing strategy for the Pro SLA of motivating scenario that is included in Figure 3, would be $R_{pro} = (97, 97.5)$ if the increasing and decreasing routing speed parameters are *0.4* and *0.5*, respectively.

As denoted in Figure 7, the SLA monitor is responsible for detecting and notifying if the thresholds of the routing strategies (i.e. *minAv* or *maxAv*) have been exceeded or not. Thus, when current availability experienced by a consumer (c.f. *current_Av* in Figure 7) is less than the *minAv* threshold, the SLA controller increases the provisioning level for such a consumer (c.f. alt block called *IncreasingProvisioningLevel* in Figure 7). For instance, if the service must fulfill an availability of 95% to a consumer, and the routing strategy is $R_1 = (97\%, 99\%)$, when the SLA monitor detects that such a consumer has experienced an availability of 96.9% then, further service request from such a consumer must be routed to a higher provisoning level with more resources in order to maintain its experienced availability under control. To the contrary, when current availability experienced by the consumer is more than the *maxAv* threshold, the SLA controller decreases the provisioning level for the consumer in order to avoid wasting more resurces than required (c.f. alt block called *DecreasingProvisioningLevel* in Figure 7).

Note that establishing appropriate routing strategies is crucial because the less difference between its threshold values and the guarantees specified in the SLA, the more SLA violation *risk* for the consumers. For instance, in previous example a routing strategy $R_2 = (96\%, 99\%)$ would be more risky than R_1 because a high number of requests may violate the SLA before routing the requests to a higher provisioning level. An appropriate routing strategy requires to choose suitable routing speed parameters and this is a challenging task because of the following situations:

- While increasing a provisioning level:
  - faster routing strategies imply less SLA violation risk, but a higher cost for the provider.
  - slower routing strategies imply more SLA violation risk, but a lower cost for the provider.
- To the contrary, while decreasing a provisioning level:
  - faster routing strategies imply more SLA violation risk, but a lower cost for the provider.
  - slower routing strategies imply less SLA violation risk, but a higher cost for the provider.

In further research, we will study a general and formal methodology of choosing an appropriate routing strategy considering previous situations. In current work we have empirically gathered that aforementioned routing strategy of $R_{pro} = (97, 97.5)$ provides a good balance in terms of cost and violation risk (c.f. Section V-B).

V. Evaluation

A. Scenario

In this section we present the experiments developed to evaluate our proposal[5]. The evaluation is contextualized in a scenario of an API for data analytics of biodiversity information about birds where different consuming organizations perform requests. This scenario corresponds with a typical API service that should address a variable workload from their consumers by allocating the appropriate resources (i.e. a number of virtual API service instances); by doing this allocation, the infrastructure incurs in costs derived from the uptime of each instance.

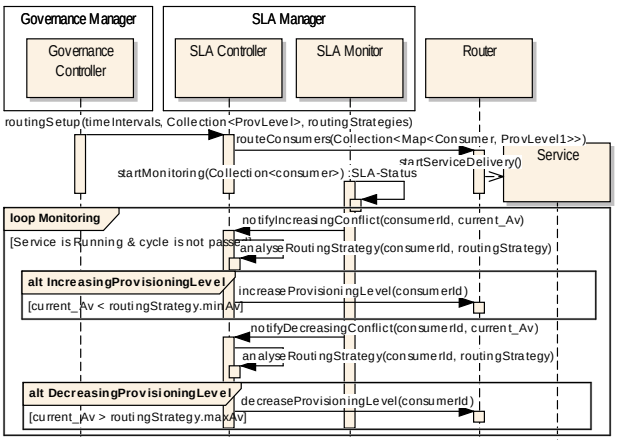

Fig. 7. SLA management between resources provisioning levels.

Specifically, in our experiments we assume two consuming organization with different SLAs: A *basic* plan for Organization 1 with a service level objective of 90 % average availability and a *pro* plan for Organization 2 with a service level objective of 95 % average availability. In this context, availability is measured as the rate of successful request compared with the total number requests. Both SLAs limit the throughput to 8 operations per seconds. We also refer throughput as workload in current section. In this experimental work we have chosen not to incorporate more concurrent organizations in order to better characterize the benefits of our approach; further works can lead to exploratory studies in more complex concurrent multi-tenant scenario.

In order to evaluate the benefits of our approach, we have designed a set of experiments to analyze the difference between our approach and two other alternatives:

- A *pre-provisioned* infrastructure with a number of resources already deployed (i.e. the required number of API service instances to fulfill the theoretical maximum number of concurrent requests). This alternative does not have any elasticity behavior and it is expected to be the most expensive option since a maximum number of instances is running during all the experimental run; as for the availability perspective, this alternative is expected to provide a 100% of availability since all required resources would be ready to respond at all the time.
- An *ungoverned* infrastructure with a pure elastic management of resources where instances are turned on and off depending on the number of concurrent requests in each point in time (a.k.a. *horizontal* elasticity). This alternative does not take into account the

[5]The evaluation prototype is available at https://github.com/isa-group/governify-research-elasticpapamoscas

different SLAs and it is expected to be the cheapest option since the instance number will be reduced to a minimum depending on the specific workload. From the availability perspective, this alternative should be highly sensitive to drastic workload changes and several response error can be generated from the elasticity adaptation (i.e. time elapsed between the elasticity controller triggers the need of a new instance and the actual instance is fully operational to accept new requests); as a consequence, this weakness could potentially cause SLA violations.

- A *governed* infrastructure (our approach) which provides a hybrid combination of the others where two provisioning levels are defined: one with a fully elasticity management of instances (such as the ungoverned approach) and other with a minimum number of instances allocated (such as the pre-provisioned approach). Specifically, in this alternative the consuming organizations are routed to the different levels depending on the service level provided in each moment so there is a fail-safe mechanism to address potential SLA violations (i.e. a kind of what is known as *vertical* elasticity as discussed in Section VII). Our approach is expected to have an appropriate trade-off in terms of costs and SLA violations.

B. Testbed

The experimental testbed consists of an 8-core i7-4770, 3.40GHz with 12GB RAM[6] running Ubuntu 14.04. The testbed is setup with an extended iCOMOT platform[7] featuring a docker-based VM elasticity for API service instances; each alternative compared correspond with a different topology described in TOSCA[8]. During the different experiment runs, each topology alternatively instantiates a different number of docker containers: (i) in the pre-provisioned topology, 8 docker containers (5 of them corresponding with API service instances) are instantiated during all the run; (ii) in the ungoverned topology, up to 7 docker containers are instantiated (4 of them corresponding with API service instances); (iii) in the topology of our governed approach, up to 9 docker containers are instantiated (6 of them corresponding with API service instances). Concerning the parametrization, elasticity speed for the ungoverned alternative is *0*, but for the governed alternative is *0.2*, resulting in the elasticity strategy *E=(1.4, 5.6)*, and *Th_VM = 7*. Th_VM was empirically calculated by stressing a single docker container (with API service instances) with different throughputs (operations per second) and assessing the maximum sustainable trouhgput that do not generates any error. In our governed infrastructure, the routing speed is established in *0.4*, and *0.5* to increase and decrease the provisioning levels, respectively, resulting in the routing strategy *R=(97, 97.5)*.

C. Experimental process

Our evaluation process is structured in a list of experiments with different workload patterns (i.e. a particular distribution of requests from the consuming organizations). Each run is repeated a number of times (typically 3) in order to calculate the average and standard deviation and assure a statistical significance of the results. Each workload pattern is defined as a sequence of N intervals of 90 or 120 seconds, during each interval each consumer organization develops a certain workload ranging form 1 to 8 (maximum allowed by SLA) operations per second. Specifically, we have developed three evolutive experiments (with a total duration of over 135 hours) to identify the benefits and applicability of our approach:

1) The first experiment consists of 11 different workload instances randomly generated from a workload pattern with N=3 (short patterns) and N=6 (long patterns).
2) The second experiment consists of 3 different workload instances manually designed to have extreme changes in the workload with N=6.
3) The third experiment consists of 6 different workload instances randomly designed with N=6 and a standard deviation of 30% to assure extreme changes in the workload.

D. Experimental results

Figures 8 and 9 show that evaluation results for the first experiment[9] validating our hypothesis: our governed alternative does not violate the SLA while the cost, that is provided by the extended iCOMOT platform, remains lower than with the pre-provisioned alternative[10]. In contrast, although the ungoverned alternative is cheaper than the governed one, it violates the SLA in two workload instances (c.f. *T08* and *T09*). In addition, it should be highlighted that in most workload instances (8 of 10), the availability provided by using our proposal is best than using the ungoverned alternative (in the other cases the difference is less than 1 % of availability as can be seen in *T06* and *T10*). As expected in our governed infrastructure, when the risk of SLA violation is detected (with the governance thresholds), the organization request is directly routed to the pre-provisioned level in order to keep the availability on desired values. Once the risk is dissipated, the organization requests would be re-routed to the elastic level again in order to decrease the cost. Thus, the governed alternative assures the SLA fulfillment. Moreover, although the cost of the governed alternative is higher than such provided by the ungoverned one (c.f. Figure 9), the SLA violations that take place with the ungoverned alternative (c.f. *T08* and *T09*) may have associated a penalty cost that is not considered in Figure 9. Note that we do not include the graph of the availability for test cases of the organization 1 because although our proposal always assured a better availability, the SLA were fulfilled by both, the governed and the ungoverned topology for all test cases.

The results of the second and third experiment analyze the situations where elasticity is stressed as much as possible (i.e. requesting in several iterations a minimum amount of throughput in a cycle and the maximum in the next cycle). The results of the workload instances of second experiment,

[6]A detailed specification can be found in the *evaluation* folder of the github repository

[7]http://tuwiendsg.github.io/iCOMOT/

[8]https://www.oasis-open.org/committees/tosca/

[9]Due to the space limitations, the full list of experiment results is available in the *evaluation/results* folder of the github repository

[10]The pre-provisioned alternative in all experiments incurs in a fixed total cost of 0.0162$ per minute and we normalize it to 100% in order to develop a comparison with the costs of the governed and ungoverned alternatives

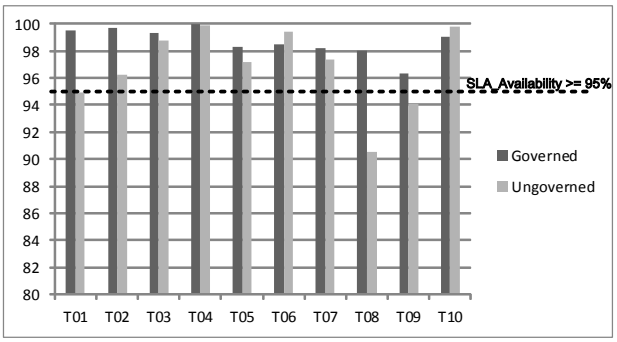

Fig. 8. Availability (in %) of Organisation 2 in the first experiment.

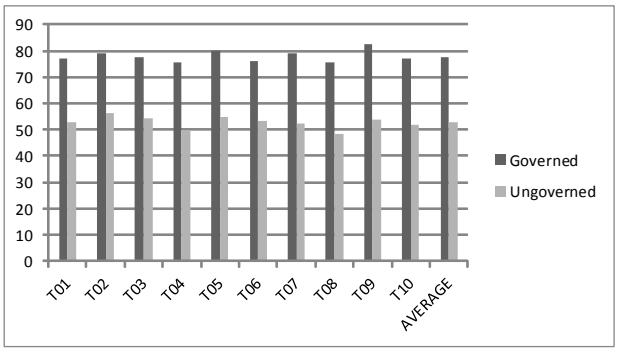

Fig. 9. Cost (in %) of both Organisations in the first experiment

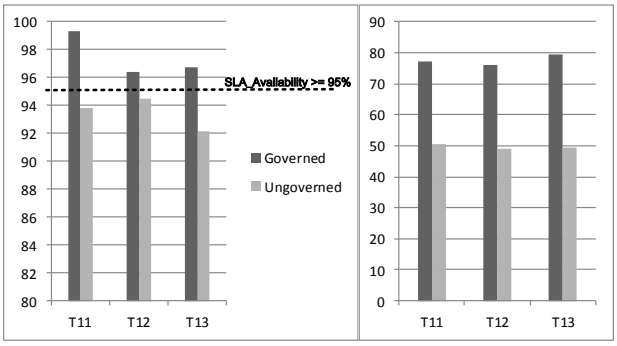

Fig. 10. Availability and Cost (in %) of Organisation 2 in the second experiment.

included in Figure 10, show that the ungoverned alternative violates the SLA of organization 2 for all test cases.

VI. Related Work

As depicted in Table I, several proposals that consider elasticity and SLAs in their works can be found in the state of the art. Most of studied proposals [10], [11], [12], [13], [14], [15] lack only horizontal elasticity. Specifically, Mansouri et al in [10] and Bonvin et al in [11] propose to use vertical elasticity by means of algorithms that improve user cost and selection of required storage services, respectively, while fulfilling the availability expected by users. Gohad et al propose in [12] an algorithm to form dynamic cloud collaborations

	Elasticity			SLAs	
	Vert.	*Horiz.*	*Multit.*	*Provider*	*Users*
Current Paper	✓	✓	✓	✓	✓
Mansouri et al [10]	✓		✓	✓	✓
Bonvin et al [11]	✓		✓	✓	✓
Gohad et al [12]	✓		✓	✓	✓
Simao et al [13]	✓		✓	✓	✓
Anastasi et al [14]	✓		✓	✓	
Cogo et al [15]	✓		✓		∼
Ali-Eldin et al [16]		✓		✓	✓
Copil et al [2]	✓	✓		✓	
Fitó et al [17]	∼		✓	✓	
Andrikopoulos et al [18]	✓			✓	

Legend: ✓ supported feature, ∼partially supported feature.

TABLE I. Support provided by the existing elasticity proposals

and thereby determine the most appropriate linkages within several providers. They take into account factors such as the changing consumers demand within the algorithm; and the resource health and provider capacities in SLAs. Simao et al in [13] propose to schedule execution units, i.e. virtual machines (VMs), driven by the partial utility of applying a certain amount of resources (CPU, memory or bandwidth) to a given VM. This partial utility metric, specified by the consumer, allows the provider to transfer resources between VMs. It is similar to the governance of topology elasticity we propose in Section IV-B, but we avoid users the responsibility of establishing such a partial utility. Anastasi et al provide in [14] a prototype of Cloud Federation that leverages the concept of usage control, by continuously monitoring and reassessing the users right on resources. They propose a usage control that permits to define policies containing conditions that must be satisfied all the time during the access (a.k.a. continuous control). Finally, Cogo et al analyse in [15] both, the benefits a replicated service may obtain from dynamic adaptations in the cloud and the requirements on the replication system. Some adaptations they analyse are: to increase/decrease the capacity of a service (as the subtopology elasticity proposed in Section IV-A), to recover compromised replicas, or rejuvenate ageing replicas (as we propose by moving from one resources provisioning level to another in Section IV-B).

From a vertical scalability perspective, our work does not present a classical approach but it provides a re-routing mechanism that represent an equivalent performance. Alternatively, two relevant works address this problem in a explicit way: on the one hand, Andrikopoulos et al propose in [18] a formal framework which allows exploring the possibility space of optimally distributing application components across cloud offerings in an efficient and flexible manner. On the other hand, Fitó et al [17] propose a comprehensive risk management approach to maximize profits and consumer satisfaction but in contrast with our approach, this work does not take into consideration elasticity strategies into the service level management.

Finally, we highlight the approach in [19] where authors provide a method to detect workload patterns that can be used to analyse and predict QoS outcomes in a given infrastructure; this model could be a valuable foundation to complement our work and develop a systematic characterization of a topology in order to design the most appropriate scalability thresholds.

VII. Conclusions and Future Work

Although existing elasticity approaches provide mechanism to leverage a limited amount of the virtualized resources required to provide the service, they are focused to consumers with the same QoS expectations. The elasticity benefit has not been properly analysed when dealing with several consumers each demanding a different QoS for the service. Thus, in this paper we develop an elasticity-aware governance platform combining elasticity management with SLA management to support multiple QoS expectations from multiple consumers.

The main contributions of the paper tackle the challenges exposed in the motivation scenario:

- An SLA management that support to provide the same service to consumers with different QoS expectations.
- An elasticity management that support elastic provisioning levels that scales in or out when needed, depending on the workload.
- An SLA monitoring that control if the service level provided to the consumers is as they agreed.
- An proper routing of consumer requests to leverage an adjusted amount of virtualized resources depending on the workload and the agreed SLAs.

In addition, a prototype has been developed and putting available online to evaluate our approach. Specifically, our proposals proof to provide a right balance between the cost of deploying the architecture and the risk of violating the SLA terms. In contrast, a compared pure elastic alternative provides a cheaper solution but with more SLA violation risk; and a compared pre-provisioned alternative results in higher cost for the provider. Specifically, our proposals proof to be better than both: (i) a cheaper elasticity approach that violates the SLAs, and (ii) a more expensive approach that leverages the 100% of resources.

Several issues that are out of the scope of current paper will be analysed in further research, such as optimising: (i) the number of resources provisioning levels, (ii) the elasticity speed, and (iii) the routing speed. In addition, a study of the most appropriate strategies as for the elasticity as for routing must be performed considering the provisioning cost for the provider. Moreover, since we just consider in our approach two of the most commonly used QoS properties (i.e. availability and throughput), we have to analyse in future how elasticity strategies may affect to the values of other QoS properties (e.g. response time, mean time between failures, etc.). Furthermore, although we have detected a promising scenario that validates our approach (i.e. with extreme changes in the workload), we will develop in further research a benchmark to a better characterisation of the different scenarios.

Acknowledgment: The authors would like to thank for his helpful comments and technical support: Duc-Hung Le, Daniel Moldovan (Distributed Systems Group, TU Wien); and Daniel Arteaga (ISA research group, University of Seville).

References

[1] A. Almeida et al., "A branch-and-bound algorithm for autonomic adaptation of multi-cloud applications," in *Proceedings of the 2014 IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2014.

[2] Georgiana Copil et al., "Multi-level elasticity control of cloud services," in *Service-Oriented Computing*, ser. Lecture Notes in Computer Science, S. Basu, C. Pautasso, L. Zhang, and X. Fu, Eds., 2013, vol. 8274.

[3] T. Kirkham et al., "Richer requirements for better clouds," in *2013 IEEE 5th International Conference on Cloud Computing Technology and Science (CloudCom)*, vol. 2, Dec 2013, pp. 7–12.

[4] Rajkumar Buyya et al., "Sla-oriented resource provisioning for cloud computing: Challenges, architecture, and solutions," in *Proc. of International Conference on Cloud and Service Computing*, 2011.

[5] Ana Juan Ferrer et al., "Optimis: A holistic approach to cloud service provisioning," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 66 – 77, 2012.

[6] R. Moreno-Vozmediano, R. Montero, and I. Llorente, "Key challenges in cloud computing: Enabling the future internet of services," *Internet Computing, IEEE*, vol. 17, no. 4, pp. 18–25, July 2013.

[7] C. Müller, M. Resinas, and A. Ruiz-Cortés, "Automated analysis of conflicts in ws–agreement," *IEEE Transactions on Services Computing*, 2014.

[8] C. Müller, "On the Automated Analysis of WS-Agreement Documents. Applications to the Processes of Creating and Monitoring Agreements," International dissertation, Universidad de Sevilla, 2013. [Online]. Available: http://www.isa.us.es/sites/default/files/muller-Phd-PTB.pdf

[9] Andrieux et al., "Web Services Agreement Specification (WS-Agreement) (v. gfd-r.192)," 2011, OGF - Grid Resource Allocation Agreement Protocol WG.

[10] Y. Mansouri, A. Toosi, and R. Buyya, "Brokering algorithms for optimizing the availability and cost of cloud storage services," in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE 5th International Conference on*, vol. 1, 2013, pp. 581–589.

[11] N. Bonvin, T. G. Papaioannou, and K. Aberer, "A self-organized, fault-tolerant and scalable replication scheme for cloud storage," in *Proceedings of the 1st ACM Symposium on Cloud Computing*, ser. SoCC '10. New York, NY, USA: ACM, 2010, pp. 205–216.

[12] A. Gohad et al., "Towards self-adaptive cloud collaborations," in *Proceedings of the 2013 IEEE International Conference on Cloud Engineering*, 2013, pp. 54–61.

[13] J. Simão and L. Veiga, "Flexible slas in the cloud with a partial utility-driven scheduling architecture," in *Proc. of IEEE International Conference on Cloud Computing Technology and Science - Volume 01*, ser. CLOUDCOM '13. Washington, DC, USA: IEEE Computer Society, 2013, pp. 274–281.

[14] Gaetano F Anastasi et al., "Usage control in cloud federations," in *Proc. of 2014 IEEE International Conference on Cloud Engineering*. Washington, DC, USA: IEEE Computer Society, 2014.

[15] Vinicius Cogo et al., "Fitch: Supporting adaptive replicated services in the cloud," in *Distributed Applications and Interoperable Systems*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, vol. 7891, pp. 15–28.

[16] A. Ali-Eldin, J. Tordsson, and E. Elmroth, "An adaptive hybrid elasticity controller for cloud infrastructures," in *Network Operations and Management Symposium (NOMS), 2012 IEEE*, April 2012, pp. 204–212.

[17] J. O. Fitó and J. Guitart, "Business-driven management of infrastructure-level risks in cloud providers," *Future Generation Computer Systems*, vol. 32, no. 0, pp. 41 – 53, 2014.

[18] Vasilios Andrikopoulos et al., "Optimal distribution of applications in the cloud," in *Advanced Information Systems Engineering*, ser. Lecture Notes in Computer Science, vol. 8484. Springer International Publishing, 2014, pp. 75–90.

[19] L. Zhang, Y. Zhang, P. Jamshidi, L. Xu, and C. Pahl, "Service workload patterns for qos-driven cloud resource management," *Journal of Cloud Computing*, vol. 4, no. 1, pp. 1–21, 2015.