2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing

On Controlling Cloud Services Elasticity in Heterogeneous Clouds

Georgiana Copil, Daniel Moldovan, Hong-Linh Truong, Schahram Dustdar

Distributed Systems Group, Vienna University of Technology
E-mail: {e.copil, d.moldovan, truong, dustdar}@dsg.tuwien.ac.at

***Abstract*—Various complex cloud services have to be deployed in multiple heterogeneous clouds, due to the service requirements for particular functionalities from specific clouds. In order to control these cloud services, we need to monitor and control the various units deployed across multiple clouds, dealing with cloud-specific protocols to support an end-to-end cloud service perspective. In this paper we present an approach for multi-cloud control, which evaluates relationships among different units deployed across heterogeneous clouds, and generates action plans necessary for controlling service elasticity. We show experiments of the end-to-end control and sensitivity analysis for a service deployed across two different types of clouds.**

***Keywords*—multi-cloud, end-to-end, elasticity control**

I. Introduction

A considerable amount of work has been put into cloud service control, focusing on controlling cloud services of various types, or meeting various types of stakeholder requirements [1]–[3]. Next to these challenges, some services might need to be distributed in several types of clouds, each with different characteristics (e.g., mini-clouds for managing smart buildings in combination with a public cloud for data analysis). Within this multi-cloud configuration, the cloud service elasticity should be controlled as a whole, for fulfilling stakeholder (e.g., service developer or service provider) requirements, possibly at multiple abstraction levels of the service.

In this case, several challenges need to be tackled when designing the control for services executed across multiple clouds. First, the *programming interfaces* offered by various cloud providers can differ both from the point of view of the services offered, and from the perspective of the protocols used (e.g., Flexiant's FCO[1] REST API, or OpenStack[2] python or java-based libraries). Second, the *run-time control capabilities* offered by different clouds have different enforcement time and results (e.g., in mini-clouds a VM spawning action usually takes much longer than in a public cloud). In this context, the user needs an elasticity control with an *end-to-end view of the multi-cloud service*, understanding different levels of service abstraction, as opposed to current control flows where the user needs to deploy separate controllers on each cloud, and treating the distributed service parts as different services. A controller offering this end-to-end perspective has to analyze various types of information characterizing cloud heterogeneity, and evaluate the impact upon the rest of the service when enforcing an action. For instance, in the context of smart city services, when releasing virtual resources in a smart building cloud, e.g., in sensors' low-activity period, a controller should consider the impact that this would have on the rest of the service possibly deployed on a public cloud.

To address challenges above, we propose mechanisms of controlling the service from an end-to-end perspective, transparent to users from the point of view of the clouds heterogeneity. To this end, we model details specific to multi-cloud deployments in order to base our mechanisms on a common representation of the highly heterogeneous services offered by cloud providers. For supporting controlling services from an end-to-end perspective, we focus on modeling relationships among multi-cloud distributed service units, to be used as a basis for elasticity control of the cloud service.

The contributions of our paper are the following: (i) a model for multi-cloud deployed services, with focus on relationships which occur among service parts, (ii) new mechanisms for relationship-driven control of multi-cloud services. We extend our rSYBL [4] controller with proposed multi-cloud mechanisms, and show that by specifying simple, high level requirements, stakeholders can have elastic services deployed on multiple, heterogeneous clouds controlled at runtime with rSYBL elasticity controller. Moreover, we emphasize the extensibility of our framework which can be easily customized to support a variety of clouds and enforcement APIs.

The rest of this paper is organized as follows: Section II motivates our work. Section III presents our approach of multi-cloud elasticity control. Section IV presents the multi-cloud controller prototype and experiments. Section V concludes the paper.

II. Motivation, Background and Related Work

A. Motivation

Let us consider a Machine-to-Machine Data as a Service (M2M DaaS), for a smart city. The smart city has millions of sensors in each building, and a group of buildings sending their data to a local mini-cloud (e.g., NuvlaBox[3], or Ubuntu Orange Box[4]) in which the M2M local processing units are deployed and to which the sensors send data (left side of Fig. 1). The data analyzed locally is sent to a public cloud

This work was supported by the European Commission in terms of the CELAR FP7 project (FP7-ICT-2011-8 #317790). The authors would like to thank Stefan Nastic and Hung Duc Le for their fruitful insights.

[1] http://www.flexiant.com/flexiant-cloud-orchestrator/
[2] http://developer.openstack.org/
[3] http://sixsq.com/products/nuvlabox.html
[4] http://www.ubuntu.com/cloud/tools/jumpstart-training

978-1-4799-7881-6/14 $31.00 © 2014 IEEE
DOI

IEEE computer society

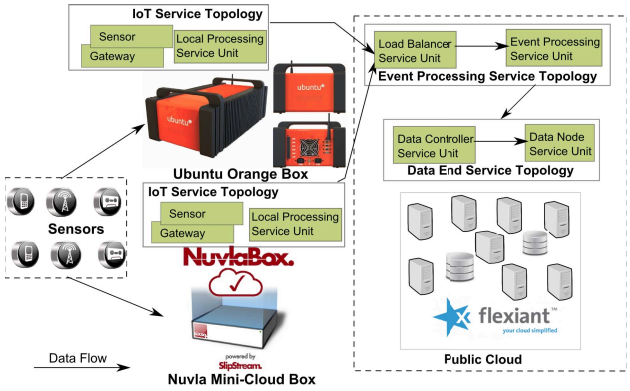

Fig. 1: Motivating Scenario

containing the rest of the M2M service (right side of Fig. 1) for the higher availability of public clouds and due to the fact that smart building mini-clouds can store a limited amount of data. The M2M service needs to be controlled both in the local mini-clouds and in the public one, in order to obtain the best possible performance at the best possible cost. To this end, user's (i.e., any service stakeholder) requirements for an elasticity controller play a major role, stating the trade-off with regard to cost, quality and performance. We envision that the users would describe requirements in a cloud agnostic manner, at a high level, considering application level elasticity metrics, and all the rest is being handled by the controller. Given that the M2M service is deployed on multiple clouds, controllers would need to use various programming interfaces and service elasticity capabilities offered by different types of cloud providers. Moreover, because users need an end-to-end control of the service, the controller has to understand the relationships among service parts deployed on different clouds, and control them accordingly.

To address above-mentioned challenges, we propose a model representing multi-cloud service information for controllers to understand elasticity capabilities specific to different cloud providers. Based on this model, we design control mechanisms using common methods of enforcing control capabilities, and focusing on relationships existent in multi-cloud services. With a stakeholders centric approach, we are not interested in providing standardized access to federated clouds, but quite the opposite: we want to help them profit from the current heterogeneity for creating multi-cloud elastic services within nowadays dynamic cloud context.

B. Background: Single-cloud service control

The first step in controlling the elasticity of a cloud-deployed service is the requirements specification, which enables the service stakeholders to describe how and towards what goal should the controller manage their service. For the description of elasticity requirements we use SYBL, a language for multi-level elasticity requirements specification. We consider the complex cloud service structure and enable the user to specify requirements at (i) cloud service, (ii) service topology, (iii) service unit, and (iv) code region level. For each level, the user can specify the following requirement types: (i) `MONITORING` directives for specifying what needs to be monitored and under which conditions, (ii) `CONSTRAINT` directives define limits for the elasticity behavior, and (iii) `STRATEGY` directives, specifying the mechanisms of achieving a desired elasticity behavior.

Our rSYBL elasticity controller analyzes the service runtime, and generates action plans for fulfilling user's requirements.The service is modeled during runtime as a dependency graph, containing structural, elasticity and infrastructure information, having as nodes the concepts describing the services and as edges the relationships among them.

C. Related Work

Cloud control [5] aims at formulating and solving a range of cloud management problems through control theoretic approaches. Al-Shishtawy et al. [3] propose ElastMan, a framework which combines feedforward and feedback control for the elasticity control of Cloud-based elastic key-value stores. Xiong et al. [6] describe vPerfGuard, a framework for application performance diagnosis in consolidated cloud environments, automatically discovering metrics which are most descriptive of application performance, and adaptively detecting changes in performance, thus supporting cloud service stakeholders in managing their applications. Miglierina et al. [7] propose a control theoretic approach for multi-cloud services, targeting resource level control and modeling formally specific types of service units which are supported (e.g., load balancer). Aragna et al. [8] propose a model-driven approach for the multi-cloud deployment and monitoring of cloud services over multiple cloud infrastructures, describing a framework to support full transparency from the user. For multi-cloud service control, Wu et al. [9] study how web services latency impact of multi-cloud environments, while Hu et al. [10] study the cost impact of having multi-cloud backup servers, proposing a schema for computing the optimal number of backup servers. Elmroth et al. [11] propose control mechanisms to facilitate multi-cloud architectures from a cloud federation perspective, focusing on predictive elasticity, admission control, and virtual machines placement. Almeida et al. [12] propose a branch and bound approach for optimally selecting services from multiple clouds during runtime. As opposed to this, we aim to support various types of services, using customization mechanisms for providing elasticity control, and we are focusing on cloud services which are deployed in heterogeneous multi-cloud environments due to their inherent requirements (e.g., privacy, availability, or functionality requirements).

Differently from other approaches, our work leverages the heterogeneity of clouds and their services, thus enabling better runtime elasticity control for service developers. We enable an end-to-end control of services during runtime, maintaining a simple requirements structure, thus shifting only the users' focus towards the elasticity metrics they are interested in, specific for multi-cloud services. This way, users can profit from differences among cloud providers, and improve their service performance, or achieve complex services with the combination of domain-specific private and public clouds.

III. Multi-cloud elasticity control

For enabling the multi-cloud control described in our motivation scenario, the controller needs to analyze multi-cloud service information, from capabilities of each service part, each

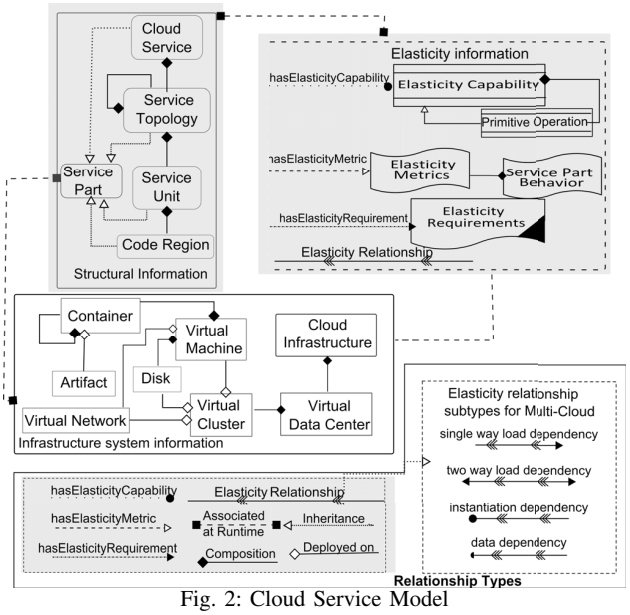

Fig. 2: Cloud Service Model

infrastructure element, to each measured metric. For instance, for the case of the M2M DaaS service in Fig. 1, gateways' elasticity capabilities and deployment structures from mini-clouds would differ from deployment stacks and elasticity capabilities of web services deployed in the public cloud. Therefore, clouds heterogeneity results in a diversity of the cloud services, not only of the supporting infrastructures.

This reveals a two-fold challenge: (i) from a conceptual perspective, we need to enable the user to have an end-to-end view of the service, (ii) from a technical point of view, the controller has to understand a variety of primitives and protocols from a variety of providers. For addressing the first challenge, we model the multi-cloud specific information by extending our model for single-cloud services [4]. We address the second challenge in Section III-B, showing how we use the modeled information to enable dependency-aware multi-cloud service control and how we abstract elasticity capabilities from primitive operations which are available for the different virtual resources offered by cloud providers.

A. Multi-cloud service model for elasticity control

In our control approach described in Section II, the cloud service is modeled together with the infrastructure in which it runs, which is single-cloud. For *multi-cloud service control*, we extend the model used in the single cloud control (Fig. 2, gray background) for being able to better represent the infrastructure, from virtual machine to software artifacts and their placement and configuration (Fig. 2, white background).

For understanding the cloud infrastructure used by the service during runtime, and the different resources from different clouds which are used to host the service's software stack, we introduce the following virtual infrastructure concepts:

- *Virtual Data Center* – the data center where the cloud service is deployed (e.g., Amazon EU location 1)
- *Virtual Cluster* – a group of resources physically isolated from the rest of the resources (e.g., the resources for a specific company)
- *Disk* – a disk instance associated to a VM, or shared among multiple VMs
- *Virtual Network* – network associated to a virtual cluster
- *Container* – any type of software having the property of being used by others as a container (e.g., Docker[5], web server, gateway, or data store).
- *Artifact* – any type of atomic software (e.g., web service, sensor, or data set).

Each of the service parts described in the structural information (upper left side of Fig. 2) has associated elasticity information (upper right side of Fig. 2), such as *Elasticity Metrics*, *Elasticity Requirements*, or *Elasticity Capabilities*. The latter represents complex actions exposed by service parts, composed of *Elasticity Capabilities* exposed by infrastructure elements described above, to which we refer as *Primitive Operations*. We distinguish among them since primitive operations can be linked to an actual operation made by the elasticity controller, e.g., an API call, while elasticity capabilities are more abstract actions which are composed of several primitive operations (e.g., a scale in elasticity capability for the service unit level can be a decommissioning primary operation achieved by the software/artifact level and a remove VM primary operation achieved at the virtual machine level). Enforcing elasticity capabilities is equivalent to the enforcement of the associated primitive operations, considering their dependencies.

For understanding how different service parts interact, we introduce the following elasticity relationship types (bottom of Fig. 2), as subtypes of *Elasticity Relationships*:

- *Single way load dependency* – a change in the antecedent load causes a similar change in consequent load
- *Two way load dependency* – a change in the antecedent or consequent causes a similar change in the other
- *Instantiation dependency* – for the instantiation of the consequent the antecedent should to exist. This relationship can also contain other properties, like the data needed to be transferred among the two.
- *Data dependency* – the specified data should to be transferred among the antecedent and the consequent.

Relationship concepts above are used in order to connect two parts of the service, regardless on whether or not they are in the same cloud. All the above concepts are represented at runtime through a runtime dependency graph, which has as nodes the concepts and as edges the relationships presented in the model (bottom of Fig. 2), and which is used by the elasticity controller to take control decisions. We obtain relationships above from service profilers that have analyzed the execution of respective services [13], or from other various service stakeholders. The model contains sufficient information for the controller to understand the complex implications of an enforcement of an elasticity capability on one end, on the rest of the cloud service hosted in other cloud infrastructures.

B. Multi-cloud service elasticity control

Based on the model above, we propose mechanisms to control the multi-cloud service, based on two major issues that rise out of the multi-cloud setting: (i) mechanisms to control service parts relationships/dependencies in multi-cloud

[5]http://www.docker.com/

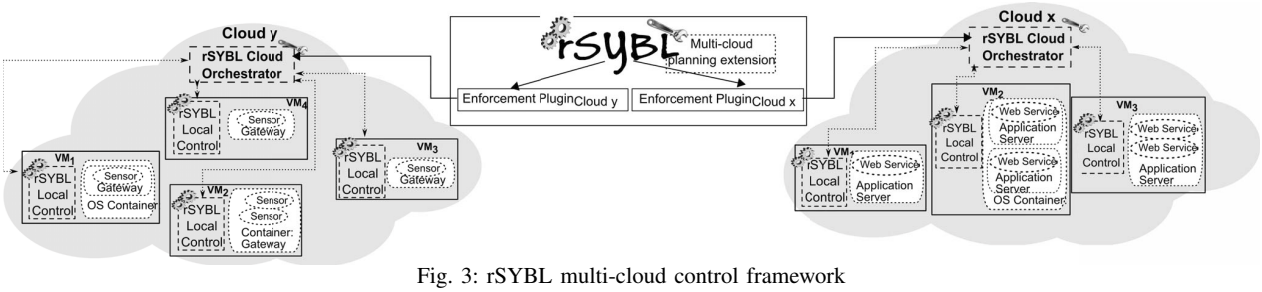

Fig. 3: rSYBL multi-cloud control framework

Algorithm 1 Multi-cloud elasticity capability analysis

Input: $evalEC$ - evaluated elasticity capability, sp - targeted service part, $graph$ - current dependency graph
Output: $resultedECs$ - Elasticity capabilities to be enforced with $evalEC$

1: $initialReq$=evaluateRequirements($graph$)
2: $initECs$ = evaluateInstantiations($evalEC$,sp,$graph$)
3: simulateEnforcementOfECs($initECs$)
4: $spsToEval$.add(simulateImpact($graph$.getDataRel()))
5: $spsToEval$.add(simulateImpact($graph$.getLoadRel()))
6: ecs=[]
7: **for each** $service\_part$ in $spsToEval$ **do**
8: ecs.add($node$.evaluateECs())
9: **end for**
10: simulateEnforcementOfECs(ecs)
11: **if** evaluateEnforcement($dependencyGraph$,$initialReq$) **then**
12: $resultedECs$.add($initECs$)
13: $resultedECs$.add(ecs)
14: **end if**
15: **return** $resultedECs$

(Section III-B1), and (ii) mechanisms to manage primary operations' heterogeneity (Section III-B2).

1) Control mechanisms based on service part dependencies in multi-clouds: For the elasticity control of a multiple heterogeneous clouds deployed service, we evaluate the runtime dependency graph modeling the various types of information (e.g., structural, elasticity, or infrastructure) with regard to the cloud service. The major difference in the control mechanism rests in the way in which we evaluate the actions to be enforced. For the multi-cloud scenario, we evaluate the consequences of the enforcement of one elasticity capability on the rest of the parts of the service, possibly deployed in other cloud infrastructures.

Algorithm 1 shows the evaluation procedure for an elasticity capability considered by our control mechanism. When choosing an elasticity capability to be enforced, as part of the control mechanism, we evaluate the elasticity relationships associated with the target service part. We determine the instantiations necessary with the enforcement of $evalEC$ (Line 2-3 of Algorithm 1) and simulate their enforcement on $graph$ for preparing it for the other relationship evaluations. We use $simulateImpact$ function to simulate on the dependency graph $graph$ the impact reflected by elasticity relationships (Lines 4-6 of Algorithm 1), by modifying metric values or used resources according to what is defined in the relationships (e.g., for data relationship we compute expected data-related metrics like data transfer, or storage size). We analyze the dependency graph and evaluate whether compensation elasticity capabilities should be enforced for overcoming the effect produced by our initial capability. At the end of this process we have a list of elasticity capabilities $resultedECs$ to be enforced for fulfilling user's elasticity requirements, which results in an end-to-end control of the multi-cloud service.

2) Heterogeneous services control: Considering we have a service deployed across a heterogeneous multi-cloud environment, services offered by providers are highly diverse, both in their structure and in the elasticity capabilities offered.

Virtual infrastructure element	Elasticity capabilities
Bash Process	change priority, kill process
Gateway	create, delete, add data repository
RabbitMQ	create, delete, modify policies, change environment variables
Data Repository	create, delete, change access rights, create new sub-repository
Tomcat	create, delete, tomcat manager-based runtime deployment and config
Docker	run new image/artifact
Disk	resize, attach
Virtual Machine	create, delete, change number of cores, change RAM

TABLE I: Examples of elasticity primitive operations

Table I shows different elasticity primitive operations associated with various artifacts and virtual resources, which can be used simultaneously in a multi-cloud deployment. An elasticity primitive operation can be seen as an atomic operation to be executed on an infrastructure-related or software related element (e.g., virtual resource, or artifact). As we can see in this case, the nature of these primitives is quite diverse, from simple creation/deletion, to reconfigurations, or policy changing, different types of artifacts having different manner of enforcing each primitive.

IV. Prototype and Experiments

A. Prototype

We implement the above mechanisms as an rSYBL [4] extension, for managing the interaction simultaneously with different types of clouds, which host different types of artifacts. The multi-cloud rSYBL controller is able to control, based on the dependency graph which follows the new model described in Section III-A, cloud services composed of service parts distributed over multiple, heterogeneous clouds.

Fig. 3 shows the multi-cloud *rSYBL* elasticity controller, having components deployment over different clouds, and their communication. Whenever a direct communication is not possible (e.g., a private cloud API is not publicly accessible), we deploy an *rSYBL Cloud Orchestrator* to which the cloud API calls and communication with *Local rSYBL Controller* are delegated. Although the *rSYBL* controller receives all the

Infrastructure Element	Primitive	Parameters
OpenStack VM	create/remove	IP
Flexiant VM	create/remove	UUID
Flexiant Nic	create/remove/attach	UUID
HAProxy	leave/join load balancer	IP, Load Balancer Config
Cassandra	leave/join cluster	IP, Data Controller Config

TABLE II: Currently supported primitives

elasticity requirements, some are delegated to the *rSYBL Local Controller* in case the user defines local (e.g., code region) requirements, most of the times they would refer local metrics and local enforcement mechanisms (e.g., manage thread pool). Given the service description given by the service stakeholder, including its structure, the possibility of combining different artifacts and containers, and the capabilities for each artifact and container, and the monitoring information, rSYBL evaluates the dependency graph and generates a sequence of elasticity capabilities to be enforced.

B. Experimental application end-to-end view

For our experiments we use the M2M DaaS, presented in Fig. 1. We simulate the mini-clouds gateways (in the left part of Fig. 1) through virtual machines deployed on our private OpenStack cloud. We have simulated sensors as units sending real GPS data from open data stores. The sensors send data to an ActiveMQ[6] queue, which our `LocalProcessingUnit` evaluates and decides whether is urgent to send it. The data is either sent on an on-demand basis, or as bulk in periodic intervals, to the `EventProcessingTopology`, composed of a `LoadBalancerUnit` and an `EventProcessingUnit`, which further analyzes and stores the data in a `DataEndTopology`, composed of a `DataControllerUnit` and a `DataNodeUnit`. The `EventProcessingTopology` and `DataEndTopology` are deployed on Flexiant's public cloud infrastructure. The two clouds are heterogeneous, providing different services, different control primitives, and interaction protocols.

Within this setting, we use our multi-cloud rSYBL controller to manage the M2M DaaS deployed over the two cloud infrastructures. The elasticity capabilities available for the M2M DaaS service parts are *scale in* and *scale out*, each being composed of a different sequence of primitives from the ones specified in Table II (e.g., scale in for `EventProcessingUnit` is composed of a "leave load balancer" for HAProxy, followed by a removal of Nic and a removal VM Flexiant primitive). For control primitives enforcement in the two cloud infrastructures, rSYBL uses Salsa [14] for the OpenStack private cloud, and Flexiant Cloud Orchestrator (FCO) for the case of the Flexiant public cloud, which differ in the protocols used and information required for the control.

For controlling the service on multi-clouds, the M2M DaaS stakeholder describes SYBL elasticity requirements, which are interpreted and enforced by rSYBL:

- *M2MDaaS*–STRATEGY CASE `avgBufferSize` < 5: minimize (`cost`)
- *LocalProcessingUnit*–CONSTRAINT `avgBufferSize`<50
- *EventProcessingUnit*–STRATEGY CASE `responseTime` < 40ms AND `throughput` < 20ops/s: `scalein`, CONSTRAINT `responseTime`<50ms

[6]http://activemq.apache.org/

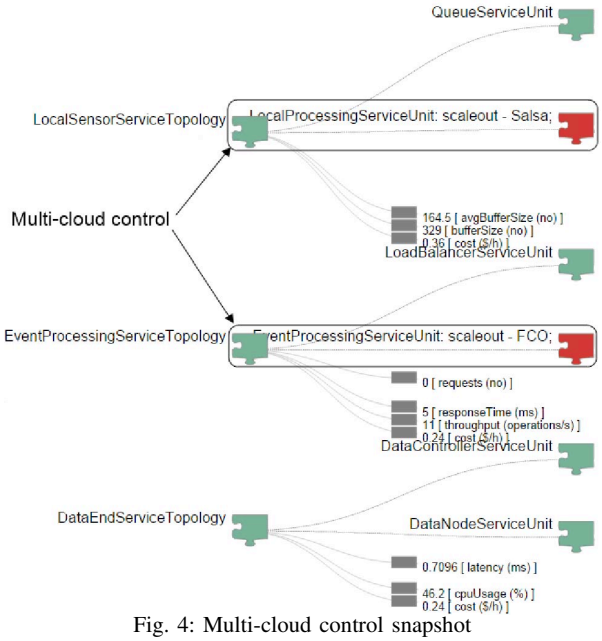

Fig. 4: Multi-cloud control snapshot

The requirements above specify constraints and strategies on monitored metrics (e.g., `bufferSize` is the total size of the currently processed data, `avgBufferSize` averages the `bufferSize` over all *LocalProcessingUnit* instances, `responseTime` is measured for the *EventProcessingUnit* web service). Moreover, the stakeholder adds a relationship that connects the two parts of the service deployed on multiple clouds, as shown in Listing 1, saying that we expect that whenever a huge amount of data is accumulated in the `LocalProcessingUnit` we would have huge amount of requests and vice-versa for the case of small amounts of data. This kind of information can be obtained from existing service analytics tools (e.g., Moldovan et al. [13]).

Listing 1: Relationship description

```
<Relationship type="Load" id="LoadRelationship">
    <source>LocalProcessingUnit</source>
    <target>LoadBalancerUnit</target>
    <metricSource>bufferSize</metricSource>
    <metricTarget>requests</metricTarget>
</Relationship>
```

Using the above information (e.g., M2M DaaS structure, associated primitives, requirements and relationships among service parts, and monitoring information from MELA[7]) which is represented in our runtime dependency graph, rSYBL generates and enforces when needed elasticity control plans containing sequences of elasticity capabilities for ensuring the fulfillment of elasticity requirements. The workload for our experiment consists of addition/removal of gateways with sensors, each gateway containing 3 to 15 sensors. Fig. 4 shows the monitored M2M DaaS, with the metrics monitored for each service topologies and service units which belong to the service. Using the mechanism described in Algorithm 1, rSYBL decides that even though the constraint targeting `responseTime` is currently fulfilled, a new instance of `LocalProcessingUnit`, would increase the load, thus

[7]http://github.com/tuwiendsg/MELA

decrease the response time. For this, a compensation elasticity capability is added to the control plan: scale out `EventProcessingUnit` on Flexiant using FCO. Fig. 5 shows the estimated cost associated to the private cloud, and cost associated to the public cloud, given that the buffer size has the shown peaks, due to the periodic sending of data to the public cloud. An increase of cost is associated with an increase of resources used by the M2MDaaS, and we can see that both on the public and private clouds the load intensity is followed. This way, the public cloud resources are allocated in advance to the load peak, for ensuring better elasticity and smaller cost on the M2M DaaS deployed on the public cloud, without having lags in provisioning sufficient resources.

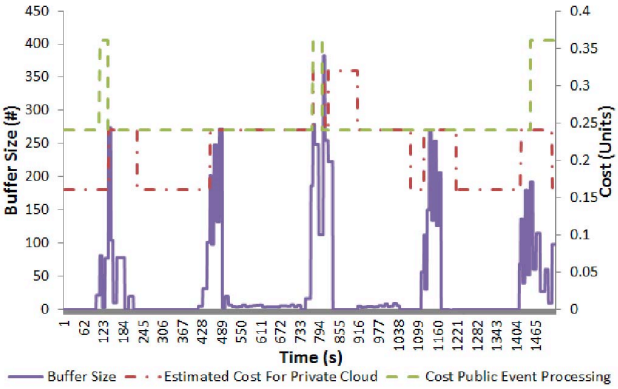

Fig. 5: Multi-cloud executed M2M DaaS cost in time

Fig. 6 shows the different amounts of time needed for enforcing elasticity capabilities on service units deployed on the two clouds. In this context, it is worth mentioning that Salsa deploys all artifacts needed on demand, while for Flexiant we use machines with needed software pre-installed. However, in both cases configurations are made on-demand, and the majority of the time is spent on creating and configuring the virtual machines. Therefore, we can affirm that in the case of Flexiant public cloud, the expected time for elasticity capabilities is much more reliable, with low standard deviation. In our experiment which lasted for 4 hours, in which each elasticity capabilities were enforced more than five times, the scaling out on Flexiant had a standard deviation of 0, while the scale in elasticity capability had a bigger deviation in average time, 2.82, due to the decomissioning (i.e., leaving the cluster) action which depends on the state of decomissioned unit.

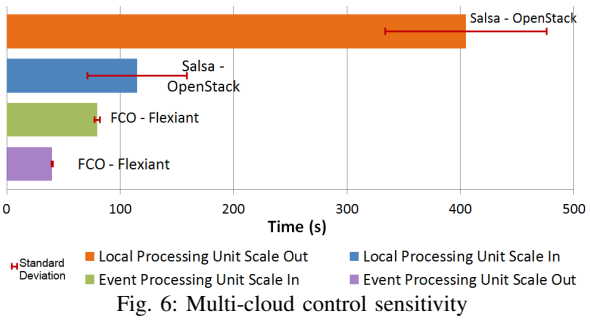

Fig. 6: Multi-cloud control sensitivity

We have shown that rSYBL is able to control an M2M service deployed on two different clouds, considering the end-to-end service perspective. For M2M service stakeholders, this is a step towards achieving better end-to-end service elasticity control, this way ensuring control for services possibly composed of multiple `LocalProcessingUnits` deployed over multiple mini-clouds.

V. Conclusions and Future Work

This paper emphasizes the need for elastic multi-cloud services, showing the main challenges, and presenting models and mechanisms for providing heterogeneous multi-cloud control for services. We have extended our framework, rSYBL, with these mechanisms, and presented some experiments with a service deployed on two different cloud infrastructures, with different time sensitivities. As future work, we will focus on discovering dependencies among services deployed on multiple clouds, which for this paper were manually described. Accurate elasticity dependencies will facilitate a better elasticity control of services executing on heterogeneous clouds.

References

[1] J. Rao, Y. Wei, J. Gong, and C.-Z. Xu, "QoS Guarantees and Service Differentiation for Dynamic Cloud Applications," *IEEE Transactions on Network and Service Management*, vol. 10, no. 1, pp. 43–55, 2013.

[2] A. Naskos, E. Stachtiari, A. Gounaris, P. Katsaros, D. Tsoumakos, I. Konstantinou, and S. Sioutas, "Cloud elasticity using probabilistic model checking," *CoRR*, vol. abs/1405.4699, 2014.

[3] A. Al-Shishtawy and V. Vlassov, "ElastMan: Elasticity Manager for Elastic Key-value Stores in the Cloud," in *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*, 2013, pp. 7:1–7:10.

[4] G. Copil, D. Moldovan, H.-L. Truong, and S. Dustdar, "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. Springer Berlin Heidelberg, 2013.

[5] M. Kihl, E. Elmroth, J. Tordsson, K.-E. Årzén, and A. Robertsson, "The Challenge of Cloud Control," in *8th International Workshop on Feedback Computing*, San Jose, CA, USA, Jun. 2013.

[6] P. Xiong, C. Pu, X. Zhu, and R. Griffith, "vPerfGuard: an automated model-driven framework for application performance diagnosis in consolidated cloud environments," in *Proceedings of the 4th ACM/SPEC International Conference on Performance Engineering*, 2013.

[7] M. Miglierina, G. Gibilisco, D. Ardagna, and E. Di Nitto, "Model based control for multi-cloud applications," in *5th International Workshop on Modeling in Software Engineering (MiSE)*, 2013, pp. 37–43.

[8] D. Ardagna, E. Di Nitto, P. Mohagheghi, S. Mosser, C. Ballagny, F. D'Andria, G. Casale, P. Matthews, C.-S. Nechifor, D. Petcu *et al.*, "Modaclouds: A model-driven approach for the design and execution of applications on multiple clouds," in *2012 ICSE Workshop on Modeling in Software Engineering (MISE)*. IEEE, 2012, pp. 50–56.

[9] Z. Wu and H. V. Madhyastha, "Understanding the Latency Benefits of Multi-cloud Webservice Deployments," *SIGCOMM Comput. Commun. Rev.*, vol. 43, no. 2, pp. 13–20, Apr. 2013.

[10] B. Hu, Y. Sudo, K. Hato, Y. Murata, and J. Murayama, "Cost reduction evaluation of sharing backup servers in inter-cloud," in *2013 19th Asia-Pacific Conference on Communications (APCC)*, 2013, pp. 256–261.

[11] in *Towards a Service-Based Internet*, ser. Lecture Notes in Computer Science, 2011, vol. 6994.

[12] A. Almeida, F. Dantas, E. Cavalcante, and T. Batista, "A Branch-and-Bound Algorithm for Autonomic Adaptation of Multi-cloud Applications," in *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, May 2014, pp. 315–323.

[13] D. Moldovan, G. Copil, H.-L. Truong, and S. Dustdar, "On Analyzing Elasticity Relationships of Cloud Services," in *2014 IEEE Sixth International Conference on Cloud Computing Technology and Science (CloudCom)*, 2014.

[14] D.-H. Le, H.-L. Truong, G. Copil, S. Nastic, and S. Dustdar, "On Analyzing Elasticity Relationships of Cloud Services," in *2014 IEEE Sixth International Conference on Cloud Computing Technology and Science (CloudCom)*, 2014.