International Journal of Cooperative Information Systems
© World Scientific Publishing Company

EVALUATING CLOUD SERVICE ELASTICITY BEHAVIOR

GEORGIANA COPIL, HONG-LINH TRUONG, DANIEL MOLDOVAN, SCHAHRAM DUSTDAR

Distributed Systems Group, Vienna University of Technology
Argentinierstrasse 8, A-1040 Vienna, Austria

DEMETRIS TRIHINAS, GEORGE PALLIS, MARIOS D. DIKAIAKOS

Department of Computer Science, University of Cyprus
P.O. Box 20537, 1678 Nicosia, Cyprus

Received (15 Feb 2015)
Revised (25 Jun 2015)

To optimize the cost and performance of complex cloud services under dynamic requirements, workflows and diverse cloud offerings, we rely on different elasticity control processes. An elasticity control process, when being enforced, produces effects in different parts of the cloud service. These effects normally evolve in time and depend on workload characteristics, and on the actions within the elasticity control process enforced. Therefore, understanding the effects on the behavior of the cloud service is of utter importance for runtime decision-making process, when controlling cloud service elasticity.

In this article we present a novel methodology and a framework for estimating and evaluating cloud service elasticity behaviors. To estimate the elasticity behavior, we collect information concerning service structure, deployment, service runtime, control processes, and cloud infrastructure. Based on this information, we utilize clustering techniques to identify cloud service elasticity behavior, *in time*, and for different parts of the service. Knowledge about such behavior is utilized within a cloud service elasticity controller to substantially improve the selection and execution of elasticity control processes. These elasticity behavior estimations are successfully being used by our elasticity controller, in order to improve runtime decision quality. We evaluate our framework with three real-world cloud services in different application domains. Experiments show that we are able to estimate the behavior in 89.5% of the cases. Moreover, we have observed improvements in our elasticity controller, which takes better control decisions, and doesn't exhibit control oscillations.

Keywords: elasticity; elasticity behavior; clustering

1. Introduction

With the wide adoption of cloud computing across multiple business domains, stakeholders seek to improve the efficiency of their complex cloud services, while also if possible to reduce costs, by acquiring on-demand virtualized infrastructure and, at the same time, benefiting from a pay-as-you-go price model offered by cloud providers. The key technique to achieve these goals is *elasticity* [1, 7] – the ability of cloud services to acquire and release resources on-demand, in response to runtime fluctuating workloads. From the customer perspective, resource elasticity can minimize task execution time, without exceeding a given budget. From the cloud provider perspective, elasticity provisioning contributes to maximizing their financial gain while keeping their customers satisfied and reducing administrative costs. However, automatic elasticity provisioning is not a trivial task.

To date, the user utilizes elasticity controllers, offered as a service, by either cloud providers (e.g., Amazon Auto Scaling[a]) or third-party vendors (e.g., Rightscale[b]), to scale his/her distributed cloud services. A common approach, employed by many elasticity controllers [2] [21], is to monitor the cloud service and (de-)provision virtual instances for the service when metric thresholds are violated. This approach may be sufficient for simple cloud services, but for large-scale distributed cloud services with complex inter-dependencies among components, we need a deeper understanding of their elasticity behavior in order to select and enforce suitable elasticity control processes. For this reason, existing work [20] [21] has identified a number of elasticity control processes to improve the performance and quality of cloud services, while additionally attempting to minimize cost. However, a crucial question still remains unanswered: *which elasticity control processes are the most appropriate for a cloud service in a particular situation at runtime*? Moreover, can both cloud customers and providers benefit from insightful information such as *how the addition of a new instance to a cloud service will affect the throughput of the overall deployment and individually each part of the cloud service*? Thus, cloud service elasticity behavior knowledge under various controls and workloads is of paramount importance to elasticity controllers for improving their runtime decision making.

To this end, a wide range of approaches relying on service profiling or learning from historic information were proposed [17] [22]. However, these approaches limit their decisions to evaluating only low-level VM metrics (e.g., CPU and memory usage) and do not support elasticity decisions considering complex cloud service behavior at multiple levels (e.g., a specific part of the service or the entire service). Additionally, current approaches only evaluate resource utilization, without considering elasticity as a multi-dimensional property composed of cost, quality, and resource elasticity. Finally, existing approaches do not consider the outcome of a control process on the overall service behavior, where often enforcing a control process on the wrong part of the cloud service can lead to side effects, such as increasing the cost or decreasing performance of the overall service.

In this article, we focus on addressing the previous limitations by introducing a methodology for estimating cloud service elasticity behavior, and a corresponding framework named ADVISE (*evAluating clouD serVIce elaSticity bEhavior*). Our behavior estimation technique introduces a clustering-based process which considers heterogeneous information for computing expected elasticity behavior, in time,

[a] http://aws.amazon.com/autoscaling/
[b] http://www.rightscale.com/

for various service parts. To estimate cloud service elasticity behavior, ADVISE utilizes different types of information, such as service structure, deployment strategies, and underlying infrastructure dynamics, when applying different workload and elasticity control processes. ADVISE analyses historical cloud service behavior, at various levels of abstraction, and produces estimations for elasticity control processes evaluated by the elasticity controller, *in time*, and for all cloud service parts, not only for the one targeted by the elasticity control process.

For validating our techniques, we integrate ADVISE in rSYBL [3] elasticity controller. rSYBL is based on SYBL elasticity requirements specification language [4], which allows service providers to describe invariants and expected service behavior. rSYBL interprets requirements specified in SYBL, and based on these requirements it provides multi-grain elasticity control for complex cloud services. To evaluate ADVISE effectiveness, experiments were conducted on two cloud platforms with a testbed comprised of three cloud services originating from different service domains. Results show that ADVISE is able to determine the expected elasticity behavior, in time, with a low error rate (i.e., average standard deviation 0.46 over all considered elasticity control processes). Therefore, ADVISE can be integrated by cloud providers alongside their elasticity controllers to improve the decision quality, or used by service providers to evaluate and understand how various elasticity control processes impact their offered services. ADVISE and all the other tools used in this article (i.e., rSYBL, JCatascopia [19], and MELA [14]) are available as open source tools, thus enabling various stakeholders to apply them, or if needed to extend them, for obtaining elasticity in their respective domains.

This article substantially extends and details our previous work [5], as follows: (i) we detail the methodology used to provide elasticity behavior estimation; (ii) we integrate ADVISE with the rSYBL elasticity controller, and detail the new architecture; (iii) we design new decision mechanisms for controlling a service based on the elasticity behavior evolution, in time, as opposed to discrete values; (iv) we present further experiments on ADVISE elasticity behavior estimation; and finally, (v) we present new experiments showing ADVISE-driven control decisions.

The rest of this paper is structured as follows: in section 2 we model relevant information regarding cloud services. In section 3, we present the elasticity behavior evaluation process. In section 4, we present how ADVISE is integrated into rSYBL elasticity controller. In Section 5, we evaluate ADVISE framework effectiveness. In section 6 we discuss related work and section 7 concludes this paper.

2. Cloud Service Structural and Runtime Information

2.1. *Cloud Service Information*

Following existing common service descriptions [16] [15], in our study we refer to cloud applications, platforms, and systems as *cloud services*. A cloud service can be decomposed into different parts such as individual service units and topologies of units; in this paper we use the term *Service Parts* (*SP*) to refer them. For understanding the behavior of cloud services, we must gather multiple types of

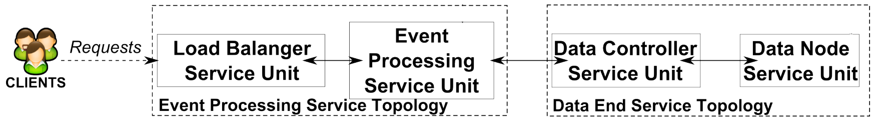

CLIENTS
Requests
Load Balanger Service Unit
Event Processing Service Unit
Event Processing Service Topology
Data Controller Service Unit
Data Node Service Unit
Data End Service Topology

Fig. 1: An M2M DaaS topology structure

information, including application-specific behavior for different service parts and the various virtual resources used, and their characteristics. Taking for instance a M2M DaaS cloud service, as depicted in Fig. 1 and used in our experimental evaluation (see section 5), the M2M DaaS is composed of multiple service units: an event processing unit, a load balancer unit, a data node unit and a data controller unit. The first two units are grouped into a service topology, which processes incoming events (i.e., *EventProcessingServiceTopology*), while events are stored into the *DataEndServiceTopology*, composed of the latter two units.

To represent such above-mentioned complex cloud services, we extend the conceptual cloud service representation model, as proposed in [3], with a rich set of information types for determining cloud elasticity behavior. Fig. 2 depicts the extensions made (white background) to include *elasticity control processes*, *service part behaviors* and *service parts*. Overall, our information model contains: (i) *Structural Information*, describing the architectural structure of the cloud service; (ii) *Infrastructure System Information*, describing runtime information regarding resources allocated for the cloud service by the underlying cloud platform; and (iii) *Elasticity Information*, capturing elasticity *metrics*, *requirements*, and *capabilities*.

Elasticity information is composed of elasticity metrics (e.g., average response time, cost, active connections), elasticity requirements (e.g., minimize response time when cost is small enough), and elasticity capabilities (e.g., add new resources), each of them being associated to different SPs or infrastructure resources. *Elasticity Capabilities* are grouped together as *Elasticity Control Processes* ($ECPs$), as described

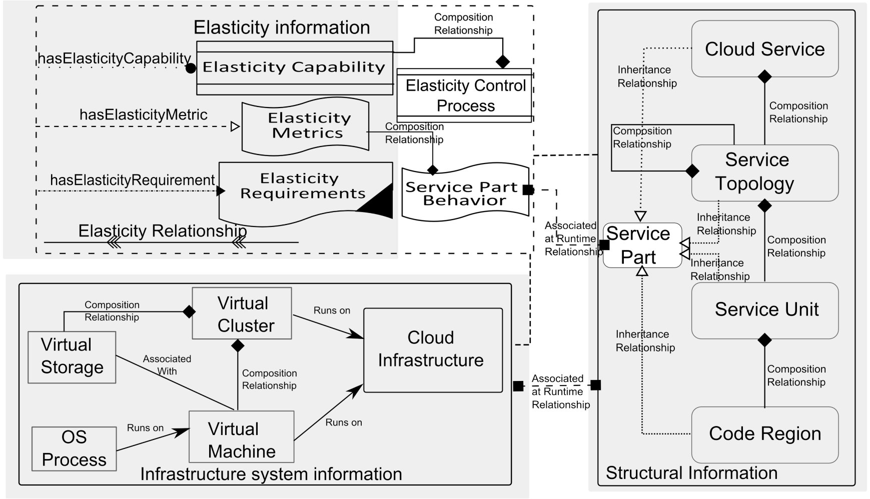


Fig. 2: Cloud service information for estimating elasticity behavior

in the next subsection, and inflict specific elasticity behaviors upon enforcement on different SPs, which we model as *Service Part Behaviors.* We model SP behaviors, since controllers must determine the effect of enforcing an ECP at different levels. For instance, in the service previously described, before allocating a new *DataNodeServiceUnit* instance, the effect, in time, at the *DataEndServiceTopology* (e.g., latency evolution for the entire cluster), and at the entire cloud service level (e.g., number of violated requirements while enforcing the ECP) should be determined.

Conceptually, a *Service Part Behavior*, denoted as $Behavior_{SP_i}$, in a defined period of time $[start, end]$, contains all the metrics, $M_a^{SP_i}$, being monitored for SP_i. Therefore, the behavior of a cloud service, denoted as $Behavior_{CloudService}$, over a period of time is defined as the set of all cloud service SP behaviors:

$$M_a^{SP_i}[start, end] = \{M_a(t_j) | SP_i \in ServiceParts, j = \overline{start, end}\} \tag{1}$$

$$Behavior_{SP_i}[start, end] = \{M_a^{SP_i}[start, end] | M^a \in Metrics(SP_i)\} \tag{2}$$

$$\begin{aligned} Behavior_{CloudService}[start, end] = \{&Behavior_{SP_i}[start, end] | SP_i \in \\ &ServiceParts(CloudService)\} \end{aligned} \tag{3}$$

The above information is captured and managed at runtime through an *Elasticity Dependency Graph*, $EDG = \{(V, E) | V \in SP \cup InfrastructureInfo, E \in Relationships\}$, which has as nodes instances of concepts from the model in Fig. 2 (e.g., `Virtual Machine, Elasticity Metric`), and relationships (e.g., `Elasticity Relationship, Inheritance, Composition`) as edges. The elasticity dependency graph is continuously updated with: (i) *pre-deployment* information, such as service topology descriptions (e.g., TOSCA [16]) or profiling information; and (ii) *runtime* information, such as metric values from monitoring tools or allocated resources from provider APIs.

2.2. *Elasticity Capabilities and Control Processes*

Elasticity capabilities (ECs) are the set of actions associated with a cloud service, whose invocation affect the cloud service behavior. Such capabilities can be exposed by: (i) different SPs (e.g., change refresh rate for a SP); (ii) cloud providers (e.g., create new virtual resources); and (iii) resources which are supplied by cloud providers (e.g., change virtual resource characteristics). An EC can be considered as the abstract representation of API calls, which differ amongst providers and cloud services. Fig. 3 depicts the different subsets of ECs provided for the *Event Processing Service Unit* (i.e., a web service hosted in a web container) when deployed on two different cloud platforms (e.g., Flexiant[c] and Openstack[d]), as well as the ECs exposed by the cloud service and its containers (e.g., Apache Tomcat). In each of the two cloud platforms, the cloud service must run on a specific container, and all these capabilities, when enforced by an elasticity controller, will affect various

[c] http://www.flexiant.com/

[d] https://www.openstack.org/

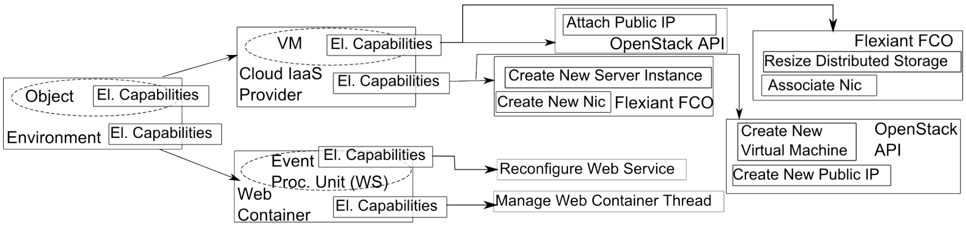


Fig. 3: Elasticity capabilities exposed by different elastic objects

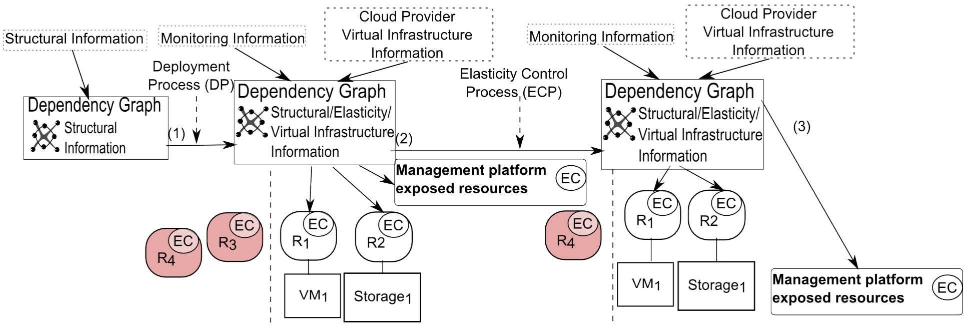

Structural Information
Monitoring Information
Cloud Provider Virtual Infrastructure Information
Monitoring Information
Cloud Provider Virtual Infrastructure Information
Deployment Process (DP)
Elasticity Control Process (ECP)
Dependency Graph
Structural Information
(1)
Dependency Graph
Structural/Elasticity/ Virtual Infrastructure Information
(2)
Dependency Graph
Structural/Elasticity/ Virtual Infrastructure Information
(3)
Management platform exposed resources
EC
EC
R4
EC
R3
EC
R1
EC
R2
VM1
Storage1
EC
R4
EC
R1
EC
R2
VM1
Storage1
Management platform exposed resources
EC

Fig. 4: Elastic cloud service evolution

cloud service parts (e.g., in the M2M DaaS, elasticity capabilities of *Event Processing Service Unit* might affect the performance of the *Data End Service Topology*).

Elasticity Control Processes (ECP) are processes composed of elasticity capabilities, which can be abstracted into higher level capabilities having predictable effects on the cloud service. $ECPs$ can be in their simplest forms, sequential elasticity capabilities, while the more complex $ECPs$ are similar to business processes (e.g., enforcement plans from TOSCA described in BPMN). We model these $ECPs$ as graphs, $ECP = (V = \{EC\}, E = \{CF, DF\})$, where the vertices are elasticity capabilities, while edges are flow dependencies among elasticity capabilities. There are two types of flow dependencies: (i) Control Flow dependencies CF, which direct the execution of the process considering the initial state, and (ii) Data Flow dependencies DF, which carry data to be used by the next EC.

An ECP causes a change in the elasticity dependency graph and in the virtual infrastructures (e.g., a change in the properties of a single VM or tier). For example, in the case of a distributed database backend which is composed of multiple nodes, a scale out ECP, with certain parameters, can be applied for both a Cassandra and an HBase database, with the following ECs: (i) add a new node; (ii) configure node properties; and (iii) subscribe node to the cluster.

2.3. *Cloud Service Elasticity*

To estimate the effects of $ECPs$ on SPs, we rely on the elasticity dependency graph which captures all the variables that contribute to cloud service elasticity behavior evolution. Fig. 4 depicts on the left-hand side the cloud service at pre-deployment

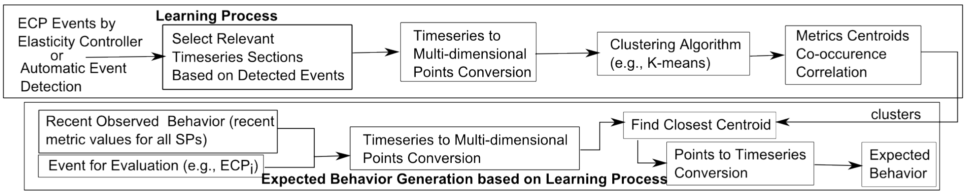


Fig. 5: Modeling cloud service behavior process

time, where automatic elasticity controllers know about it only from structural information provided by different sources (e.g., TOSCA description). After enforcing a Deployment Process (e.g., create VM, create network interface and connect to VPN), the elasticity dependency graph will be updated with infrastructure-related information obtained from the cloud provider, and elasticity information, obtained from monitoring services showing metric evolution for different SPs.

Infrastructure resources, as mentioned previously, have associated elasticity capabilities (EC in Fig. 4), that describe the change(s) to be enforced and the mechanisms for triggering them (e.g., API call assigned to the EC). In addition, a cloud platform exposes ECs in order to create new resources or instantiate new services (e.g., increase memory is an EC exposed by a VM, while create new VM is an EC exposed by the cloud platform). In this context, for being able to discover the effects that an ECP produces in time, for each SP, taking into account correlations between metrics, we use the elasticity dependency graph. We analyze this information to determine the effect of an ECP for all SPs, regardless on whether the ECP is application specific, or it does not have any apparent direct link to other SPs.

3. Evaluating Cloud Service Elasticity Behavior

Existing behavior learning solutions [22] [17] learn discrete metric models, without correlating metrics with the multiple variables affecting cloud service behavior. In contrast to these solutions, we provide behavior learning based on different SPs and their relation to multiple $ECPs$, which may or may not be directly linked, estimating the effect of an ECP, *in time*, considering correlations among several metrics and SPs. Fig. 5 depicts the SP behavior *Learning Process* which is continuously executed, refining the previously gathered knowledge base.

3.1. *Obtaining Necessary Information*

For evaluating cloud service elasticity behavior, we populate the dependency graph, described in Section 2, with all the necessary information (i.e., service parts, elasticity relationships, infrastructure system information). First, we acquire pre-deployment information to understand the cloud service and its execution environment, such as: (i) *structural information*, regarding the topology of the cloud service, and (ii) *cloud infrastructure information*. The first, described in Section 2, is generally known by the service provider, and contains the SPs of the service and relationships which appear among them. The latter describes virtual resources available in the current cloud infrastructure and their capabilities (e.g., a virtual machine of `type x` exposes the capability of memory ballooning). Afterwards, run-

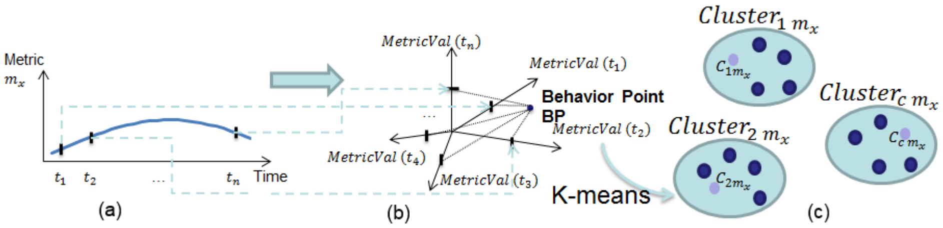


Fig. 6: Clustering process

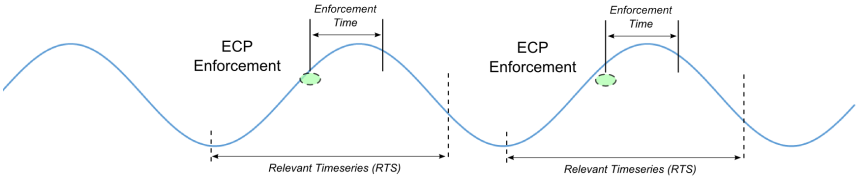


Fig. 7: Relevant timeseries selection

time information regarding the service behavior, is collected via monitoring tools (e.g., using MELA [14]), and associated with structural service units or topologies.

Monitoring data is either collected at runtime while a controller is enforcing different $ECPs$, or through a profiling step, where both rational and incorrect $ECPs$ are enforced. In both cases, issues may arise, such as application failure or virtual resource failure, leading to incomplete monitoring data. Therefore, we acknowledge two issue types: (i) *recurring issues*, which characterise a control step (e.g., a capability) and must be considered; and (ii) *random issues* which do not affect the behavior of a service. The first issue type must be reflected in the estimations, since they characterize the behavior of the service (e.g., while enforcing ECP_i SP_j cannot be monitored for X seconds). However, the second issue type can be ignored. For this, the following clustering methodology considers both, characterizing recurring behaviors (e.g., missing measurements each time a $type_x$ reconfiguration is enforced) and filtering outliers (e.g. random issues).

3.2. *Learning Process*

Fig. 6 depicts the overall elasticity behavior clustering process via selected metrics observations, with the three main steps: (a) input data processing, (b) clustering process, and (c) behavioral clusters update.

3.2.1. *Processing input data*

The learning process receives as input each metric's evolution, in time, $M_a^{SP_i}[start, current]$ (see Equation 3) starting from the beginning of a service's lifecycle. To evaluate the expected metric evolution in response to enforcing a specific ECP, we select for each monitored metric, of each service part, a *Relevant Timeseries Section* (RTS) (see Fig. 7), in order to compare it with previously encountered $M_a^{SP_i}[start, current]$. The RTS size strongly depends on the average time required to enforce an ECP (see Section 5.2.1). Consequently, a metric RTS is a

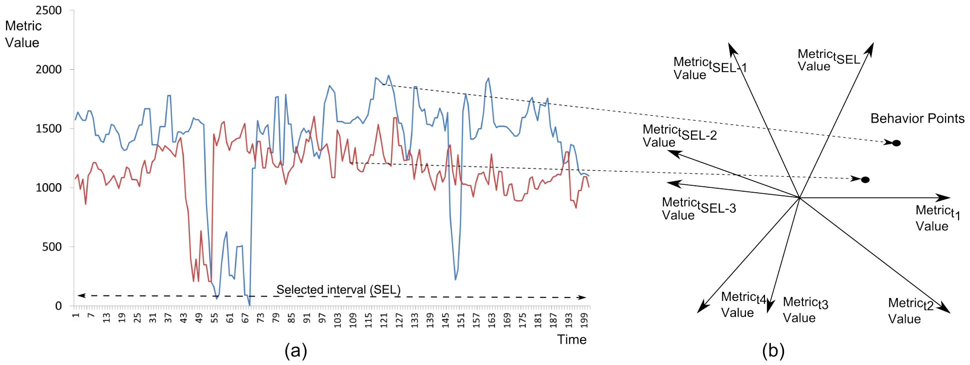


Fig. 8: Relevant timeseries sections to points

sub-sequence of $M_a^{SP_i}$, ranging from an interval before and after ECP enforcement:

$$RTS_{M_a}^{SP_i} = M_a^{SP_i}[x - \frac{\delta + ECP_{time}}{2}, x + \frac{\delta + ECP_{time}}{2}], \quad (4)$$

$$[ECP_{startTime}, ECP_{endTime}] \subset [x - \frac{\delta + ECP_{time}}{2}, x + \frac{\delta + ECP_{time}}{2}],$$

where x is the ECP index and δ is the length of the period we aim to evaluate.

As part of the input pre-processing phase, we represent $\delta + ECP_{time}$ (Fig. 5a) as multi-dimensional points (Fig. 5b and Equation 5) in the n-dimensional Euclidian space, where the value for dimension $t(j)$ is the timestamp j of the current RTS.

$$BP_a^{SP_i}[j] = RTS_{M_a}^{SP_i}[t(j)], j = 0, ..., n, BP : M^{SP} \mapsto R^n, n = \delta + ECP_{time} \quad (5)$$

3.2.2. *Clustering process*

To detect the expected behavior of an ECP enforcement result, we construct behavioral point clusters $Cluster_{SP_i}$, for all SPs and each ECP as defined in Equation 6. We do not limit our approach to only considering $ECPs$ available for the current SP_i since, as previously mentioned, enforcing an ECP to a specific SP may affect the behavior of another SP or the overall cloud service. Our objective function is to find the multi-dimensional behavior point $C(\Theta^*)$, which minimizes the distance among points belonging to the same cluster $Cluster_k$ (Equation 7). Since our focus is not to evaluate the quality of different clustering algorithms, we use the K-means algorithm, which is inexpensive, following the practice where the number of clusters is $K = \sqrt{N/2}$, N being the number of objects. However, as shown in Section 5, even with a simple K-means algorithm, our approach outputs the expected elasticity behavior with a low estimation error rate.

$$dist(BP_a^x, BP_a^y) = \sqrt{\sum_i (BP_a^x[i] - BP_a^y[i])^2} \quad (6)$$

$$\Theta^* = \arg\min \sum_{k=0}^{K} \sum_{i=0}^{N} \Theta_{i,k} dist(Cluster_k, BP_i), \quad \theta_{i,k} = \begin{cases} 1 & BP_i \in Cluster_k \\ 0 & BP_i \notin Cluster_k \end{cases} \quad (7)$$

3.2.3. *Behavioral clusters update*

For the update process, we start from the already existing clusters, and we search for new behavior points given by new ECP enforcements. We select relevant timeseries RTS for each SP in response to newly enforced $ECPs$, whenever new data is available, according to the process presented in Section 3.2.1. We represent these as behavior points BP, and add each of them to the closest clusters. The cluster update process then consists of moving BPs among the clusters until convergence, which is a lightweight process compared to running entire clustering algorithm. The overhead of updating the clusters is proportional with the number of selected $RTSs$ and the change in cloud service behavior. Even with $ECPs$ enforced very often, the cluster updating process is still insignificant due to the fact that the RTS dimensions are quite small compared to the overall monitoring data. However, it must be noted that repeated $ECPs$ in short intervals reflect a weakness in the controller, which doesn't manage to stabilize the system as we can see in section 5.

3.3. *Determining the Expected Elasticity Behavior*

3.3.1. *Elasticity Behavior Correlation*

After obtaining $\|\delta{+}ECP_{time}\|$-dimensional point clusters, we construct for each SP_i a co-occurence matrix $CM_{SP_i}[C_x^{m_i}, C_y^{mj}]$, where C_x is the centroid for $Cluster_x$, and the value of CM is the probability of clusters from metrics m_i and m_j to appear together (e.g., increase in data reliability is usually correlated with increase in cost). By taking this into consideration, when determining expected behavior points, we have a better estimation which is also based on correlation among metrics(e.g., when in M2M DaaS the *EventProcessingServiceUnit* `throughput` is high, a `Scale IN` ECP will increase `response time`, while when low, the impact might be negligible). An item in the CM represents a ratio between the number of times the behavior points C_x and C_y were encountered together and the total number of behavior points. This matrix is continuously updated when behavior points move from one cluster to another, or when new $ECPs$ are enforced, thus, increasing the knowledge base.

3.3.2. *Expected Behavior Point Determination*

In the *Expected Behavior Generation based on Learning Process* step in Fig. 5, we select the latest metric values for each SP_i, $M_a^{SP_i}[current - \delta, current]$, and the ECP_ξ which the controller is considering for enforcement, or for which the user would like to know the effects. We find the *ExpectedBehavior* (see Equation 8) which consists of a tuple of cluster centroids from the clusters constructed during the *Learning Process* that are the closest to the current metrics behavior for the part of the cloud service we are focusing on, and which have appeared together throughout the execution of the cloud service. The result of this step, for each metric of SP_i, is a list of expected values from the enforcement of ECP_ξ (e.g., all of the expected metric values for the case the elasticity controller would like to

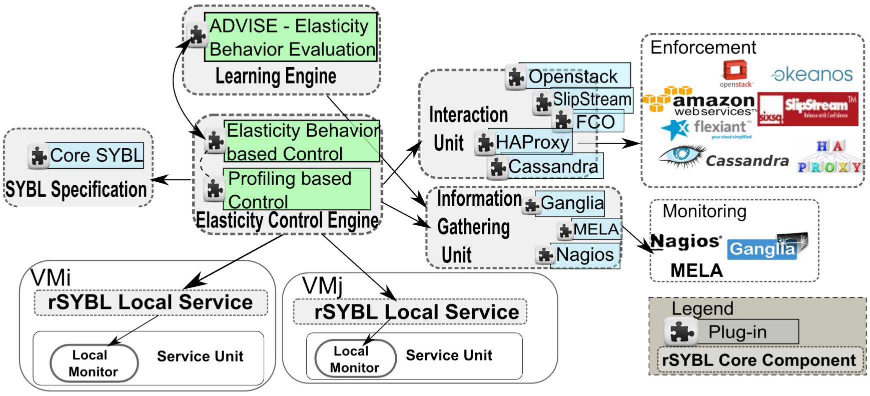


Fig. 9: ADVISE integration into rSYBL

perform a scale out ECP).

$$ExpectedBehavior[SP_i, Behavior_{SP_i}[current - \delta, current], ECP_\xi] = \\ \{C_{i_{a^1}}^{M_{a^1}}, ..., C_{i_{a^m}}^{M_{a^m}} | M_{a^m} \in Metrics(SP_i)\} \quad (8)$$

3.4. *Parameterizing and qualifying the learning process*

The quality of the expected elasticity behavior estimation depends on several, highly configurable, variables, which are either: (i) the output of a pre-profiling process; or (ii) empirically determined, based on good practices or on observing the behavior of the estimation process. Such variables include: (i) variable K, denoting the number of expected clusters; (ii) the variable $cutoff$, denoting the acceptable clustering convergence error; and (iii) the monitoring information completeness. For K, as defined above, we follow the rule of thumb proposed by Mardia et al. [13], stating that the number of clusters in a set of objects is $\sqrt{N/2}$, where N is the total number of objects. The offset is empirically determined, considering the quality of the results and the time needed for computing an estimation, as shown in Section 5.2.6.

4. Controlling Elasticity with Elasticity Behavior Estimation

We have implemented our elasticity estimation techniques in a framework named ADVISE. As described in Section 2, ADVISE collects the following heterogeneous types of information, from plugins which we have developed, to populate the elasticity dependency graph: (i) cloud service *structural information*, from TOSCA service descriptions; (ii)*infrastructure* and *application performance information*, from both JCatascopia [19] and MELA [14] monitoring systems; and (iii) *elasticity information*, regarding $ECPs$ from the rSYBL [3] elasticity controller.

We extend rSYBL to integrate ADVISE to support both understanding the behavior of the different service parts and enforcing control processes in accordance. Fig. 9 depicts the rSYBL framework integrated with ADVISE for estimating elasticity behavior and considering it when generating ECP plans. The rSYBL *Elasticity Control Engine*, now features two planning mechanisms: (i) *profiling-based control*, enabled when ADVISE-based estimations are not yet accurate enough (e.g. average standard deviation is low), and (ii) *elasticity behavior-based control*, which considers runtime elasticity behavior estimations for all service parts in order to evaluate the needed elasticity control process(es). Moreover, ADVISE also exposes statistics on the time required for an ECP to be enforced, thus being able to refine the enforcement *cool-off* period on the controller side as presented in Section 5.2.1.

Profiling-based control requires as input, per ECP, the expected effects obtained through manual or automated profiling. The effects which are available for this case are forecasts for expected metric values after finishing enforcing the respective ECP, and do not consider relationships among various SPs. The $ECPs$ are enforced through the rSYBL Interaction Unit, which interacts with both cloud provider-specific APIs and with application specific control mechanisms.

For generating control plans, we interpret the effect, in time, of $ECPs$ in relation to expected behavior without enforcing the respective control process. In this sense, *elasticity behavior-based control* estimates current metric evolution using a polynomial fitting approach. We obtain the elasticity requirements to be violated for the case where we don't enforce ECP_ξ (Equation 9) by computing the integral of violated requirements over estimated polynomial metric evolutions. Equation 10 shows the computation method for violated elasticity requirements for the case of enforcing ECP_ξ, denoted $ViolatedReq(ECP_\xi)$, using ADVISE estimations. We compare the two estimations, and select the one with the least violated requirements.

$$ViolatedReq(ECP_\xi) = \int_{current}^{current+\lambda} ViolReq(P(SP_i, Metric_j(x)))dx \quad (9)$$

$$ViolatedReq(ECP_\xi) = \int_{current}^{current+\lambda} ViolReq(ADVISE(SP_i, Metric_j(x)))dx (10)$$

We design the controller in such a way, that when the current behavior cannot be accurately estimated, we are able to rollback to profiling-based control. In this way, the enforced $ECPs$ are not restricted solely to information obtained via the pre-deployment phase, instead the ADVISE behavior estimation is continuously refined improving the knowledge base and learning new behavior points representative for each ECP. In other words, when the closest estimated centroids are farther than a distance `dist`, empirically defined, the current decision making algorithm rolls back to the initial decision-making algorithm, as described in [3].

5. Experiments

In this section, we provide an evaluation of the ADVISE framework[e], focusing on the clustering-based behavior estimation process to determine the effectiveness of our approach as ADVISE can be used in both service profiling/pre-deployment or during runtime, for various service types, whenever monitoring information and enforced $ECPs$ are available. As described in Section 2, ADVISE collects the following

[e] code, detailed descriptions and more charts at `http://tuwiendsg.github.io/ADVISE`

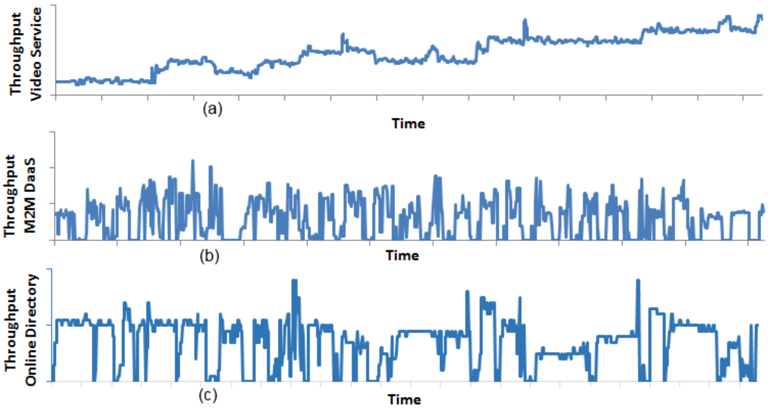


Fig. 10: Workload applied on the three services

heterogeneous types of information, from plugins which we have developed, to populate the elasticity dependency graph: (i) cloud service *structural information*, from TOSCA service descriptions; (ii)*infrastructure* and *application performance information*, from both JCatascopia [19] and MELA [14] monitoring systems; and (iii) *elasticity information*, regarding $ECPs$ from the rSYBL [3] elasticity controller.

Our evaluation is divided into two phases: (i) ADVISE framework evaluation; and (ii) ADVISE-enabled rSYBL evaluation. To evaluate the functionality of the ADVISE framework, we established a testbed on the Flexiant public cloud comprised of three cloud services originating from different service domains featuring distinct structural and behavior requirements. On the selected cloud services, we first, enforce $ECPs$ exposed by their respected SPs randomly, and then study, at runtime, their behavior at multiple levels of the cloud service. It must be noted, that we did not configure rSYBL as a rational controller, since we are interested in estimating the elasticity behavior for all SPs as a result of enforcing both good and bad elasticity control decisions. For the second phase, we established a testbed on an OpenStack private cloud, with rSYBL elasticity control for a cloud service. We evaluate how ADVISE affects rSYBL elasticity control on various workloads.

5.1. *Experimental Cloud Services*

The **first cloud service** is a **three-tier web application** providing **video streaming services** to online users, comprised of: (i) an *HAProxy Load Balancer* which distributes client requests across multiple application servers; (ii) An *Application Server Tier*, where each application server is an Apache Tomcat server exposing the video streaming web service; and (iii) A Cassandra *NoSQL Distributed Data Storage Backend* from where the necessary video content is retrieved. The database backend initially holds 2GB of data while at the end of the experiment the size approaches 6GB. To stress this cloud service we generate client requests under a fixed request rate, though the load is not stable and depends on the type of the requests (e.g. download video) and the size of the requested video, as shown

Cloud Service	**ECP ID**	**Action Sequence**
Video Service	ECP_1	*Scale In Application Server Tier*: (i) stop the video streaming service, (ii) remove instance from HAProxy, (iii) restart HAProxy, (iv) stop JCatascopia Monitoring Agent, (v) delete VM
	ECP_2	*Scale Out Application Server Tier*: (i) create new network interface, (ii) instantiate new VM, (ii) deploy and configure video streaming service, (iv) deploy and start JCatascopia Monitoring Agent, (v) add VM IP to HAProxy, (vi) restart HAProxy
	ECP_3	*Scale In Distributed Video Storage Backend*: (i) select VM to remove, (ii) decommission instance data to other nodes, (iii) stop JCatascopia Monitoring Agent, (iv) delete VM
	ECP_4	*Scale Out Distributed Video Storage Backend*: (i) create new network interface, (ii) instantiate new VM, (iii) deploy and configure Cassandra (e.g., assign token to node), (iv) deploy and start JCatascopia Monitoring Agent, (v) start Cassandra
M2M DaaS	ECP_5	*Scale In Event Processing Service Unit*: (i) remove service from HAProxy, (ii) restart HAProxy, (iii)remove recursively VM
	ECP_6	*Scale Out Event Processing Service Unit*: (i) create new network interface, (ii) create new VM, (iii) add service IP to HAProxy
	ECP_7	*Scale In Data Node Service Unit*: (i) decommission node (copy data from VM to be removed), (ii)remove recursively VM
	ECP_8	*Scale Out Data Node Service Unit*: (i) create new network interface, (ii) create VM, (iii) set ports, (iv) assign token to node, (v) set cluster controller, (vi) start Cassandra
Online Directory	ECP_9	*Scale In Distributed Document Store*: (i) select Couchbase-server to remove, (ii) decommission node from Couchbase cluster, (iii) rebalance cluster data, (iv) remove VM
	ECP_{10}	*Scale Out Distributed Document Store*: (i) create new network interface, (ii) instantiate VM, (ii) configure interfaces, ports, and Couchbase-server (iv) start Couchbase-server, (v) join Couchbase cluster, (vi) rebalance cluster

Table 1: Elasticity control processes available for the cloud services

in the workload pattern in Fig.10a.

The **second cloud service** in our evaluation is a **Machine-to-Machine (M2M) DaaS** which processes information originating from several different types of remote data sensors (e.g., temperature, atmospheric pressure, or pollution). Specifically, the M2M DaaS is comprised of an *Event Processing Service Topology* and a *Data End Service Topology*. Each service topology consists of two service units, one with a processing goal, and the other acting as the balancer/controller. To stress this cloud service we generate random sensor event information (see Fig. 10b) which

Cloud Service	SP Name	Metrics
Video Service	Application Server Tier	cost, busy thread number, request throughput
	Distributed Video Storage Backend	cost, CPU usage, memory usage, query latency
M2M DaaS	Cloud Service	cost per client per hour
	Event Processing Service Topology	cost, response time, throughput, number of clients
	Data End Service Topology	cost, latency, CPU usage
Online Directory	Document Store Controller	cost, request rate, active sessions, error rate, CPU usage, network I/O
	Distributed Store Node	cost, throughput, cache miss rate, disk I/O, memory usage, CPU usage, query response time

Table 2: Elasticity metrics per cloud service for different service parts

is processed by the *Event Processing Service Topology*, and stored/retrieved from the *Data End Service Topology*.

The **third cloud service** showcased is a **two-tier OLTP service** deployed as an **online business directory** hosting 7503 local business listings and their products[f]. The topology of this service is comprised of a *Document Store Controller*, under a public domain, distributing client requests (i.e., create new listing, get directions to *Restaurant X*) to a *Document Store Nodes*, forming a *Distributed Document Store* which is a Couchbase database backend. Specifically, the database backend is a distributed, shared-nothing NoSQL (JSON-like) document store, optimized for interactive web applications, incorporating in its core application logic allowing developers to prepare and expose to their users queries as light-weight map/reduce functions (i.e., top-k *breweries* in town *Nicosia* etc.). We stress this service by generating client requests under a variable read-heavy request rate, mimicking a real online directory's behavior, as depicted in Fig. 10c (writes occur only when adding a new listing or updating an existing one and constitute less than 10% of the load). Tables 1 and 2 list the *ECPs* associated to each *SP* and the monitoring metrics analyzed for the three cloud services respectively.

5.2. *Cloud Service Elasticity Behavior Evaluation*

5.2.1. *ECP Temporal Effect*

ADVISE computes, as shown in Table 3, the *average time* required for an *ECP* to be completed, and returns also a standard deviation which gives the degree of

[f] Our dataset is synthetic, created from real data and workload patterns from www.finditcyprus.com which we gratefully thank

	ECP	**Standard Deviation**	**Average ECP Time (s)**
Video Service	ECP1	0.06	90
	ECP2	0.12	25
	ECP3	0.13	160
	ECP4	0.11	30
M2M Service	ECP5	0.34	45
	ECP6	0.16	20
	ECP7	0.11	70
	ECP8	0.14	20
Online Directory	ECP9	0.29	110
	ECP10	0.12	25

Table 3: Elasticity control processes time statistics

confidence with regard to this estimation. This application-specific information is of high importance and affects the decision-making process of the elasticity controller since it is an indicator of the *grace period* which it should await until effects of the resizing actions are noticeable. Thus, it can define the time granularity of which resizing actions should be taken into consideration. For example, we observe that the process of reconfiguring and removing an instance from the video service's storage backend requires an average time interval of 160 seconds which is mainly due the time required to receive and store data from other nodes of the ring. If decisions are taken in smaller intervals, the effects of the previous action will not be part of the current decision process and may cause cascading *ping-pong* effects where a Scale In ECP followed by a slight increase in metric utilization (in the *grace* period) causes a false Scale Out ECP.

5.2.2. *Online Video Streaming Service - Elasticity Behavior Estimation*

Fig. 11 depicts both the *observed* and the *estimated* behavior for the Video Service Application Server Tier when $ECPs$ of type ECP_1 (*remove application server*) are enforced. At first, we observe that the average `request throughput` per application server is decreasing. This is due to two possible cases: (i) the video storage backend is under-provisioned and cannot satisfy the current number of requests which, in turn, results in requests being queued; (ii) there is a sudden drop in client requests which indicates that the application servers are not utilized efficiently. For an elasticity controller driven by simple "if-then-else" policies for application-specific metrics (e.g. request throughput) there is no apparent way in determining the case in hand and it will act upon metric violations without considering if an ECP will indeed improve QoS or cost. From Fig. 11, we observe that after the Scale In ECP occurs, the average `request throughput` and `busy thread number` rises which denotes that this behavior corresponds to the second case where resources are now efficiently utilized. ADVISE revealed an insightful correlation between two metrics to consider in the decision process.

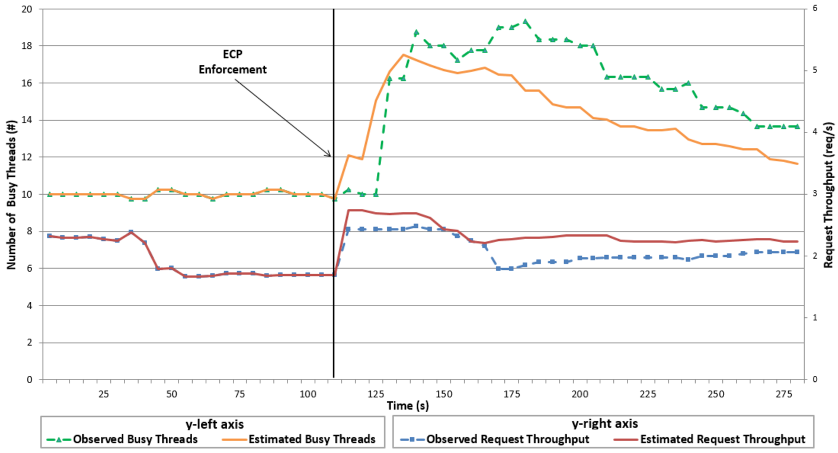


Fig. 11: Effect of ECP_1 on the application server tier

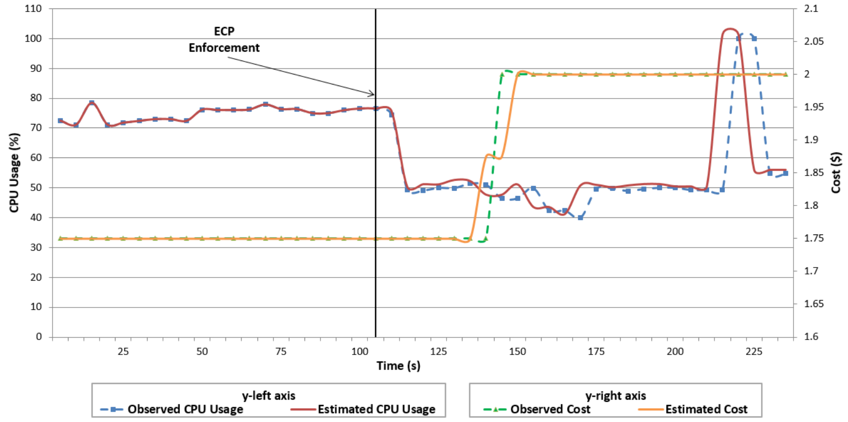


Fig. 12: Effect of ECP_4 on the entire video streaming service

Similarly, in Fig. 12 we depict both the *observed* and the *estimated* behavior for the Distributed Video Storage Backend when a Scale Out ECP is enforced (add Cassandra Node to ring) due to high `CPU utilization`. We observe that after the Scale Out *ECP* is enforced, the actual `CPU utilization` decreases to a normal value as also indicated by the estimation.

Analyzing the estimations made for this service (i.e., Fig. 11 - 12), we conclude that the estimations provided by ADVISE successfully follow the actual behavior pattern and that, as time intervenes, the curves tend to converge.

5.2.3. *M2M DaaS - Elasticity Behavior Estimation*

Fig. 13 showcases how an *ECP* targeting a service unit affects the entire cloud service. The `Cost/Client/h` is a complex metric (see Table 2) which depicts how profitable is the service deployment in comparison to the current number of users. Although `Cost/Client/h` is not accurately estimated, due to the high fluctuation in number of clients, our approach approximates how the cloud service would behave

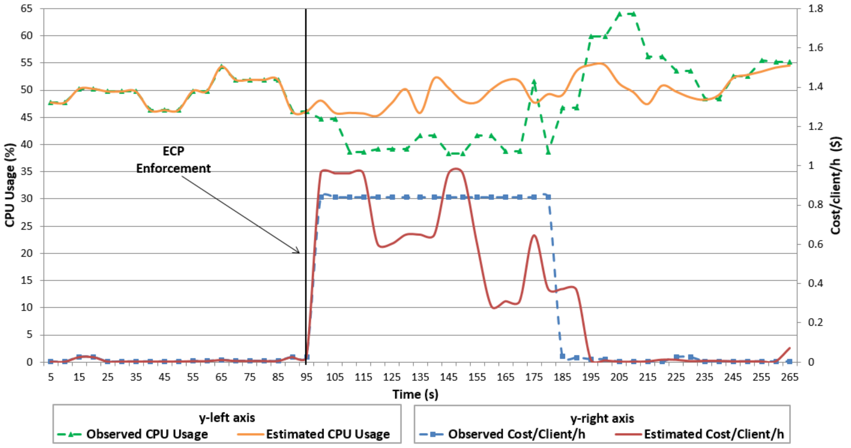


Fig. 13: Effect of ECP_7 on M2M DaaS

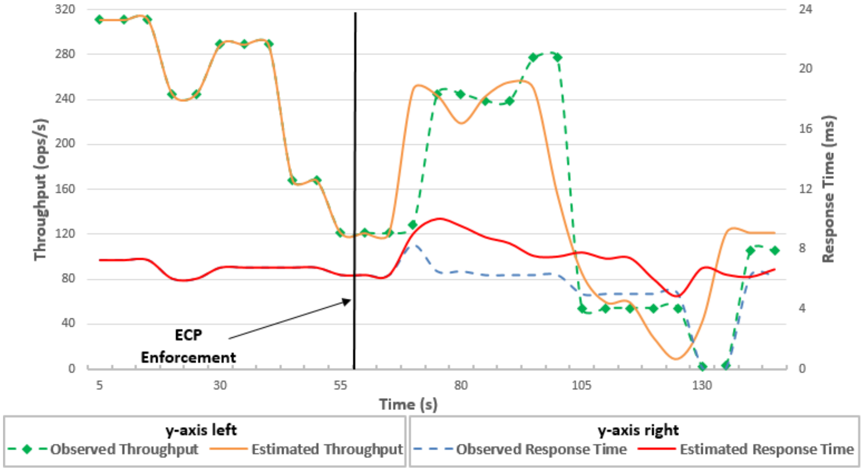


Fig. 14: Effect of ECP_6 on the event processing service topology

in terms of *expected metric fluctuations*. This information is important for elasticity controllers to improve their decisions when enforcing $ECPs$ by knowing how the `Cost/Client/h` for the entire cloud service would be affected. Although `CPU usage` is not estimated perfectly, since it is a highly oscillating metric, and it depends on the `CPU usage` at each service unit level, knowing the baseline of this metric can also help in deciding whether this ECP is appropriate (e.g., for some applications `CPU usage` above 90% for a period of time might be inadmissible). Fig. 14 shows estimations of behavior for the Event Processing Service Topology, when a Scale Out ECP occurs on the Event Processing Service Unit. Although the `throughput` is accurately estimated with a slight lag, `response time` is estimated with a slightly larger error due to the fact that a down peak is not estimated, as not being part of the usual behavior for the current SP.

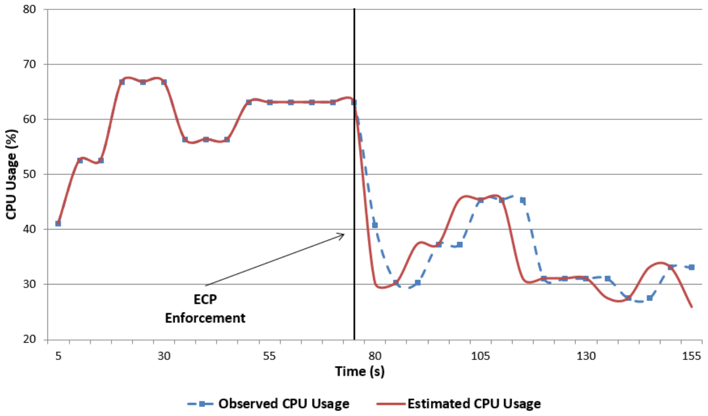


Fig. 15: Effect of ECP_8 on the data controller service unit

ADVISE can estimate the effect of an ECP of a SP, on a different SP, even if apparently unrelated and therefore provide, multi-grain elasticity behavior evaluation. Fig. 15 depicts an estimation on how the Data Controller Service Unit is impacted by the data transferred at the enforcement of ECP_8. In this case, the controller `CPU usage` initially, as one would aspect, decreases since the new node will offload other nodes, however, effort is required in transferring data to the new node which rises utilization due to the fact that reconfigurations are also necessary on the controller, following a slight decrease and then stabilization. Therefore, even in circumstances of random workload, ADVISE can give useful insights on how different SPs behave when enforcing $ECPs$ exposed by other SPs which, again, elasticity controllers have no knowledge of.

5.2.4. *Online Directory - Elasticity Behavior Estimation*

Fig. 16 depicts the *observed* and *estimated* `throughput` and `CPU usage` measured at the Document Store Controller after a Scale Out ECP is enforced. The Document Store Controller is a document store node itself, however, it features additional functionality: it *supervises (meta-)data migration* when the cluster is re-balanced as in the case of a Scale In/Out ECP enforcement. While other nodes continue to accept client requests when an ECP is enforced, the Document Store Controller prioritizes the supervisioning of data movement and thus, ceases to process client requests. This is evident in Fig. 16, where we observe that when the cluster is rebalanced, `throughput` drops to zero while `CPU usage` does not decrease, as one would expect, until after rebalancing is complete. In turn, Fig. 17 depicts the effects of a Scale In ECP on one of the document store nodes. ADVISE identifies the increase in `memory utilization` as the node receives part of the load from the decommissioned node, while the estimation for `CPU utilization` follows the observed oscillating trend.

On the other hand, Fig. 18 depicts both `CPU usage` and `cost` of a document store node before and after a Scale Out ECP. We observe that before the ECP

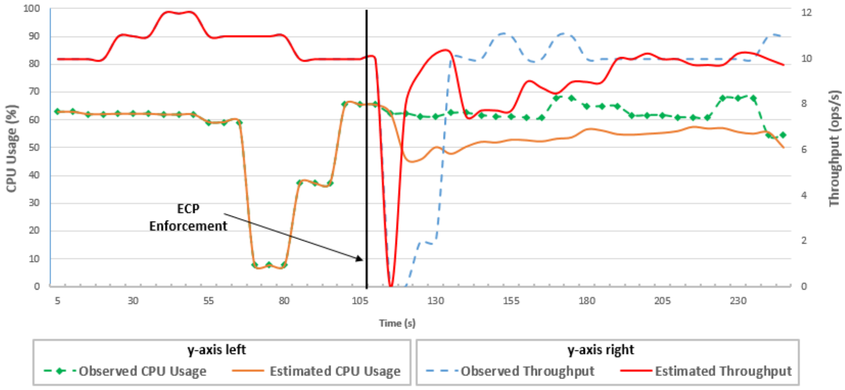


Fig. 16: Effect of ECP_{10} on the Document Store Controller

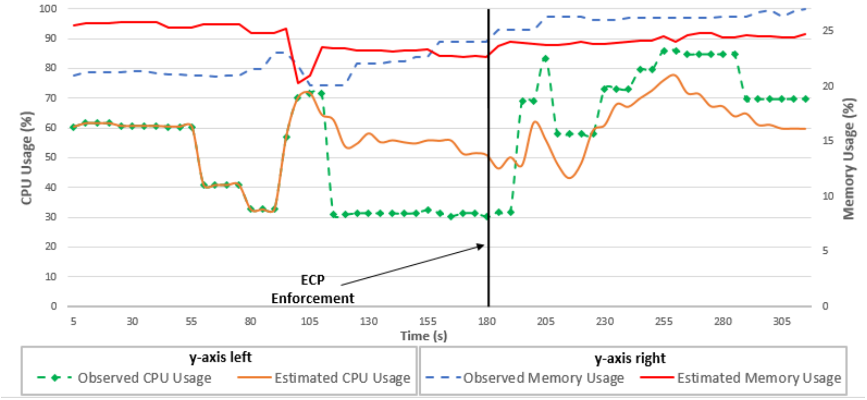


Fig. 17: Effect of ECP_9 on the Document Store Node

enforcement the ADVISE estimation follows the observed values, however, after the ECP enforcement, the provided estimation slightly deviates from the observed `CPU utilization` before converging again. The reason behind this slight deviation is due to, an out of the ordinary, large data movement, not evident in most Scale Out *ECPs*, where the whole dataset must be replicated since the Scale Out *ECP* occurs immediately after a series of multiple Scale In *ECP*s which left the cluster with only two instances.

5.2.5. *Quality of Results*

ADVISE is able to estimate, in time, the elasticity behavior of different *SPs* by considering the correlations amongst metrics and the *ECPs* which are enforced. To evaluate the quality of our results, we have considered the fact that existing tools do not produce continuous-time estimations. Thus, we evaluate ADVISE by computing the variance Var and standard deviation *StdDev* (Equation 11), over

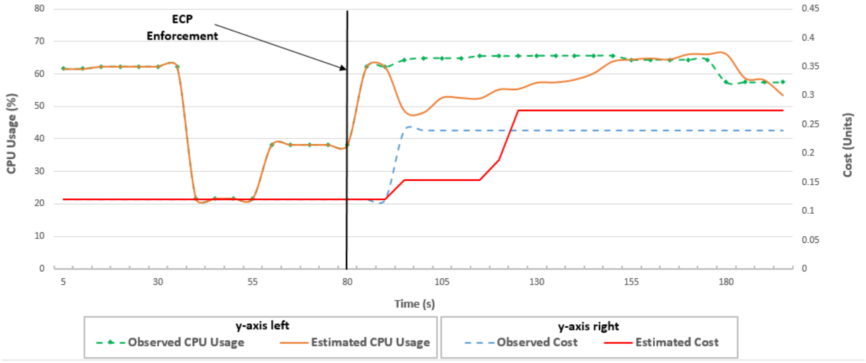


Fig. 18: Effect of ECP_{10} on the data node

Cloud Service	Observed Cloud Service Part	Elasticity Control Process	Average Standard Deviation	Maximum Variance	Minimum Variance
Video Service	Video Service	ECP_3	0.23	0.09	0.03
		ECP_4	0.61	0.99	0.23
	Distributed Video Storage Backend	ECP_3	0.28	0.14	0.034
		ECP_4	0.2	0.042	0.04
	Application Server	ECP_1	0.43	0.4	0.06
		ECP_2	0.31	0.47	0.01
M2M Service	Cloud Service	ECP_5	0.9	6.65	0.24
	Data End Service Topology	ECP_5	0.23	0.35	7.44E-05
	Event Processing Service Topology	ECP_7	1.1	4.9	0.046
		ECP_8	0.76	2.46	0.027
	Data Controller Service Unit	ECP_6	0.12	0.25	0
		ECP_8	0.22	0.41	0
	Data Node Service Unit	ECP_5	0.572	0.68	0.32
		ECP_6	0.573	1.4	0.07
	Event Processing Service Unit	ECP_7	1.08	3.59	0.11
		ECP_8	0.77	1.9	0.14
Online Directory	Document Store Node	ECP_9	0.19	0.05	0.29
	Document Store Node	ECP_{10}	0.14	0.005	0.18
	Document Store Controller	ECP_{10}	0.13	0.023	0.38

Table 4: ECPs effect estimation quality statistics

100 estimations as the result differs little afterwise.

$$Var_{metric_i} = \frac{\sum \frac{\sum_{i=[0,rts_{size}]} (estMetric_i - obsMetric_i)^2}{rts_{size}}}{nbEstimations - 1}$$
$$StdDev_{metric_i} = \sqrt{Var_{metric_i}} \qquad (11)$$

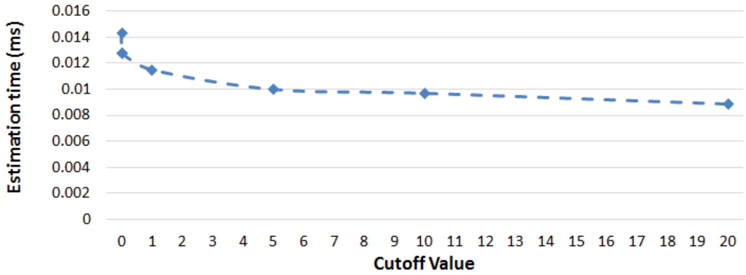


Fig. 19: ECP_5 *estimation time* under different Cutoff values

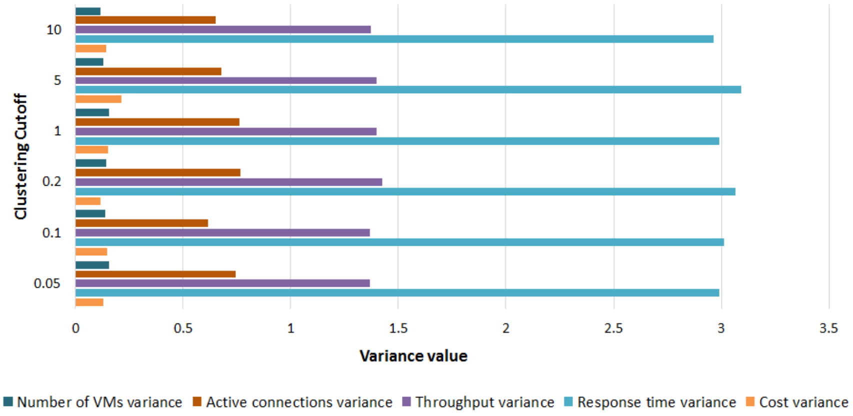


Fig. 20: Estimation **variance** for ECP_5 under different Cutoff values

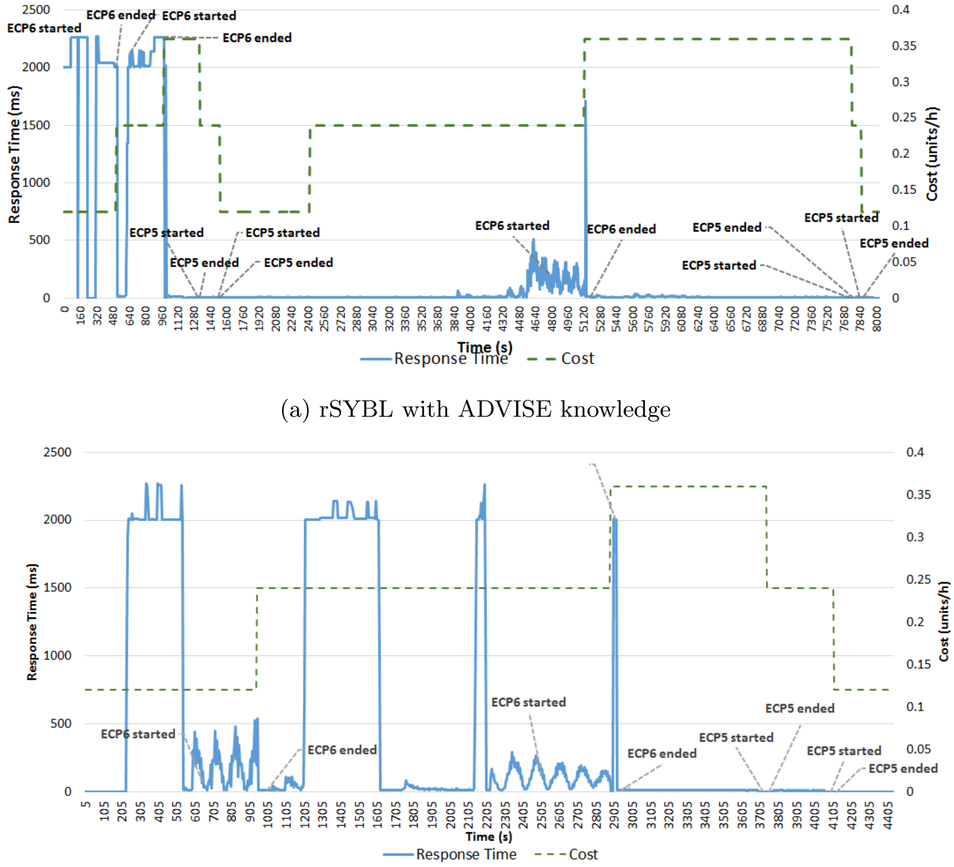
Table 4 presents the accuracy of our results. When comparing the **three services**, the Video Service achieves a higher accuracy (smaller standard deviation), since the imposed workload is considerably stable. Focusing on the M2M DaaS estimation accuracy, we observe that it depends on the granularity at which the estimation is calculated, and on the ECP. Moreover, the standard deviation depends on the metrics monitored for the different parts of the cloud service. For instance, in the case of the M2M Service, the `number of clients` metric can be hardly predicted, since we have sensors sending error or alarm-related information. This is evident for the Event Processing Service Topology, where the maximum variance for the `number of clients` is 4.9.

Overall, even in random cloud service load situations, the ADVISE framework analyses and provides estimations for elasticity controllers, allowing them to improve the quality of control decisions, with regard to the evolution of monitored metrics at the different cloud service levels. Moreover, these estimations are delivered together with the confidence of the estimation, given by the distance from current behavior point to the estimated behavior point. Without this kind of estimation, elasticity controllers would need to use VM-level profiling information, while having to control complex cloud services. This information, for each SP, is valuable for controlling elasticity of complex cloud services, which expose complex control mechanisms.

5.2.6. *Sensitivity of results*

For analyzing the sensitivity of our results, we evaluated how our empirically determined parameters (e.g. clustering offset) affect the variance of the estimation in regards to the observed behavior. Fig.20 concludes that our results are very little affected by the choice of the offset.

When analyzing the impact that the choice of the offset has over time (Fig. 22b), we can see that very small offset values reflect in a considerable increase in the estimation time. This is why, the offset was chosen at `0.2`, as a tradeoff between estimation time and estimation quality.



(a) rSYBL with ADVISE knowledge

(b) rSYBL without ADVISE

Fig. 21: Event Processing Topology control

5.3. *rSYBL elasticity control enhanced with ADVISE estimations*

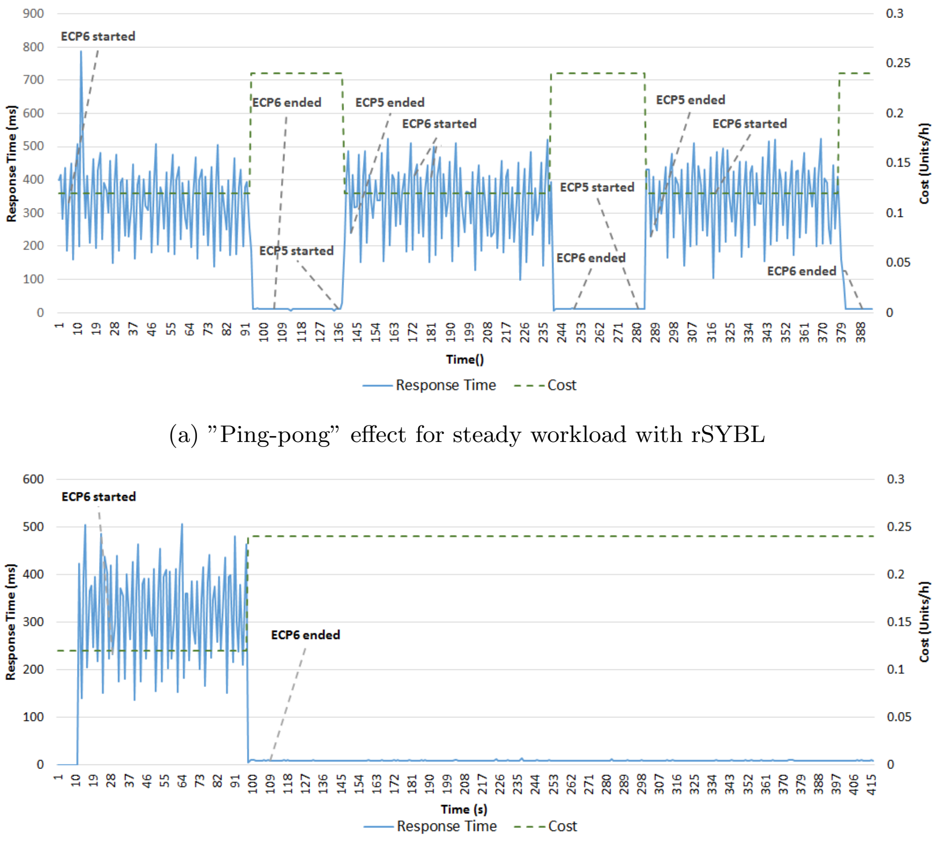
As described in Section 4, we integrate the ADVISE behavior estimation into the rSYBL elasticity controller, which controls elasticity at multiple levels of abstraction [3].

We use the M2M DaaS service with fixed *Data End Topology*, while controlling *Event Processing Topology* using ECP_5 (i.e., Scale In) and ECP_6 (i.e., Scale Out) elasticity control processes on an OpenStack private cloud, with the following SYBL [4] elasticity requirements:

- *EventProcessingTopology* – Co1:CONSTRAINT `responseTime` < 100 ms;
- *EventProcessingTopology* – St1:STRATEGY CASE `responseTime` < 12 AND `throughput`<100 : minimize(`cost`)

We compare the elasticity control performed by ADVISE-enabled rSYBL decision making and respectively the profiling-based decision making. We apply a stepwise workload in order to observe controller's behavior under different circumstances. Fig. 21 shows at (a) the outcome of controlling the service with ADVISE-enabled rSYBL and at (b) the outcome of controlling the service considering profiling information, consisting of how much the metrics would be affected by enforcing an ECP. At first, we observe that ADVISE-based control provides the elasticity controller with a better elasticity behavior understanding, even in cases where metric values exceed their expected values (i.e. in this case due to a memory bottleneck).However, it accomplishes this with a bigger cost, as one would expect, since it is using more resources to fulfill the requirements (e.g., in this case ADVISE-enabled controller achieves a cost 27.5% higher). In contrast with this, the profiling based elasticity controller is not able to find appropriate $ECPs$ in unexpected situations, when the value observed exceeds its possibilities of controlling the metric, as known from profiling information (e.g., if the response time is expected to decrease with 200 ms whenever $ECP6$ is enforced, while current metric values are 2000 ms). Moreover, the ADVISE-based control considers the following interval when analyzing the expected behavior, not only the final metric values. Thus, whenever enforcing an ECP results in comparable requirements violation over the whole period as for the case of no ECP enforcement, the controller chooses to not take any action. When applying a steady/fix workload, depending on the formulated requirements, some controllers can reach continuous control oscillations. This was also the case for rSYBL with profiling information, as we can see from Fig. 22 (a). Due to the continuous effects which are being used by rSYBL with ADVISE, on the same workload, and the same elasticity requirements, this "ping-pong" effect is avoided (see Fig. 22 (b)), since the controller knows that the enforcement of $ECP5$ will imply the overall increase of the `response time`, after a time period.

Using ADVISE in elasticity control can also avoid situations where various control processes are enforced without understanding their outcome. For instance, enforcing Scale Out when response time satisfies a condition does not always result in response time decrease. Moreover, in some cases, where the service is not truly elastic, enforcing $ECPs$ considering expected discrete effects would only cause increase in costs. With ADVISE and SYBL, the control processes are enforced only when, considering current and previous behavior, the ECP would help fulfilling SYBL requirements (e.g., increase throughput, minimize response time).



(a) "Ping-pong" effect for steady workload with rSYBL

(b) No "ping-pong" when using ADVISE

Fig. 22: Ping-pong effect

6. Related Work

In our previous work, we focused on modeling current and previous behavior with the concepts of *elasticity space* and *pathway* [14], where we utilize different algorithms to determine enforcement times in observed service behavior (e.g., with change-point detection), but without modeling expected behavior of different service parts, in time. Verma et al. [20] study the impact of reconfiguration actions on system performance. They observe infrastructure level reconfiguration actions, with actions on live migration, and observe that the VM live migration is affected by the CPU usage of the source virtual machine, both in terms of the migration duration and application performance. The authors conclude with a list of recommendations on dynamic resource allocation. Kaviani et al. [9] propose profiling as a service, to be offered to other cloud customers, trying to find tradeoffs between profiling accuracy, performance overhead, and costs incurred. Zhang et al. [22] propose algorithms for performance tracking of dynamic cloud applications, predicting metrics values like throughput or response time.

Dean et al. [6] propose an approach for predicting running application performance anomalies, self-organizing maps for capturing emergent behavior and predicting unknown anomalies. Using unsupervised learning, this approach also identifies previously unknown anomalies/faults which may appear in the system (e.g., memory leaks, cpu leaks). For cloud service SLA violation prediction several solutions have been proposed, such as [11] [18], which use statistical models (e.g., decision trees, artificial neural networks) or naive bayes classifiers, predicting when the SLA would be violated without focusing on the violation cause. LaCurts et al. [10] propose Cicada, a framework which predicts network traffic for cloud applications, without making assumptions about application structure. The authors argue that cloud providers should offer predictive guarantees as a service, instead of bandwidth guarantees, which would also encapsulate application runtime changes. Similarly, ADVISE-enabled rSYBL can be used to guarantee or to sell as a service cloud service elasticity, with little specifications coming from the user.

Juve et al. [8] propose a system which helps automating the provisioning process for cloud-based applications. They consider two application models, one workflow application and one data storage case, and show how for these cases the applications can be deployed and configured automatically. Li et al. [12] propose Cloud-Prophet framework, which uses resource events and dependencies among them for predicting web application performance on the cloud. Shen et al. [17] propose the CloudScale framework which uses resource prediction for automating resource allocation according to service level objectives (SLOs) with minimum cost. Based on resource allocation prediction, CloudScale uses predictive migration for solving scaling conflicts (i.e. there are not enough resources for accommodating scale-up requirements) and CPU voltage and frequency for saving energy with minimum SLOs impact. Compared with this research work, we construct our model considering multiple levels of metrics, depending on the application structure for which the behavior is learned. Moreover, the stress factors considered are also adapted to the application structure and the elasticity capabilities (i.e. action types) enabled for that application type.

Compared with presented research work, we focus not only on estimating the effect of an elasticity control process on the service part with which it is associated, but on different other parts of the cloud service. Moreover, we estimate and evaluate the elasticity behavior of different cloud service parts, in time, because we are not only interested in the effect after a predetermined period, but also with the pattern of the effect that the respective ECP introduces.

7. Conclusions and Future Work

It is important to understand elasticity behavior of complex cloud services due to different control processes, in order to support better elasticity provisioning. In this paper, we have presented a methodology for estimating cloud service elasticity behavior, and implemented it in our ADVISE framework. ADVISE is able to estimate the behavior of cloud service parts, in time, when enforcing various $ECPs$, by considering different types of information represented through the elasticity dependency graph. Experiments from three different cloud services, show that ADVISE framework is indeed able to *advise* elasticity controllers about cloud service behavior, contributing towards improving cloud service elasticity. We show how we integrate ADVISE with the rSYBL elasticity controller [3], and the decision mechanisms we refined in order to consider continuous ECP effects. We have shown the improvement that ADVISE brings to the rSYBL elasticity controller, and discussed the various decision types that ADVISE influences.

As future work, we will focus on further understanding how heterogeneous resources used in the control process would affect the cloud service elasticity behavior. Moreover, we are considering various aspects which are dynamic in a complex cloud service. For instance, communication patterns might change at runtime as a reaction of the service to incoming load properties (e.g., service units which are re-grouping into other topologies when the users interests, and load characteristics, change). Moreover, enforcing elasticity capabilities can also involve changes in service structure (e.g., adding another layer of balancing involves creating a sub-topology in an already existing topology). Focusing on these aspects, we are investigating how communication and structural dynamism affect the cloud service behavior, how these can be estimated, and finally, how can they be integrated in the ADVISE framework.

Acknowledgement

This work was supported by the European Commission in terms of the CELAR FP7 project (FP7-ICT-2011-8 #317790).

References

1. 2014 State of the Cloud Survey. http://www.rightscale.com/blog/cloud-computing-trends-2014-state-cloud-survey.
2. Ahmad Al-Shishtawy and Vladimir Vlassov. Elastman: Autonomic elasticity manager for cloud-based key-value stores. In *Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing*, HPDC '13, pages 115–116, New York, NY, USA, 2013. ACM.
3. Georgiana Copil, Daniel Moldovan, Hong-Linh Truong, and Schahram Dustdar. Multi-level Elasticity Control of Cloud Services. In Samik Basu, Cesare Pautasso, Liang Zhang, and Xiang Fu, editors, *Service-Oriented Computing*, Lecture Notes in Computer Science. Springer Heidelberg.
4. Georgiana Copil, Daniel Moldovan, Hong-Linh Truong, and Schahram Dustdar. Sybl: An extensible language for controlling elasticity in cloud applications. In *Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on*, pages 112–119, May 2013.
5. Georgiana Copil, Demetris Trihinas, Hong-Linh Truong, Daniel Moldovan, George Pallis, Schahram Dustdar, and Marios Dikaiakos. Advise a framework for evaluating cloud service elasticity behavior. In Xavier Franch, AdityaK. Ghose, GraceA. Lewis, and Sami Bhiri, editors, *Service-Oriented Computing*, volume 8831 of *Lecture Notes in Computer Science*, pages 275–290. Springer Berlin Heidelberg, 2014.

6. Daniel Joseph Dean, Hiep Nguyen, and Xiaohui Gu. Ubl: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems. In *Proceedings of the 9th International Conference on Autonomic Computing*, ICAC '12, pages 191–200, New York, NY, USA, 2012. ACM.
7. S. Dustdar, Yike Guo, B. Satzger, and Hong-Linh Truong. Principles of elastic processes. *Internet Computing, IEEE*, 15(5):66–71, Sept 2011.
8. Gideon Juve and Ewa Deelman. Automating application deployment in infrastructure clouds. In *Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science*, CLOUDCOM '11, pages 658–665, Washington, DC, USA, 2011. IEEE Computer Society.
9. Nima Kaviani, Eric Wohlstadter, and Rodger Lea. Profiling-as-a-service: Adaptive scalable resource profiling for the cloud in the cloud. In Gerti Kappel, Zakaria Maamar, and HamidR. Motahari-Nezhad, editors, *Service-Oriented Computing*, volume 7084 of *Lecture Notes in Computer Science*, pages 157–171. Springer Berlin Heidelberg, 2011.
10. Katrina LaCurts, Jeffrey C. Mogul, Hari Balakrishnan, and Yoshio Turner. Cicada: Introducing predictive guarantees for cloud networks. In *6th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 14)*, Philadelphia, June 2014. USENIX.
11. Philipp Leitner, Johannes Ferner, Waldemar Hummer, and Schahram Dustdar. Data-driven and automated prediction of service level agreement violations in service compositions. *Distributed and Parallel Databases*, 31(3):447–470, 2013.
12. Ang Li, Xuanran Zong, Srikanth Kandula, Xiaowei Yang, and Ming Zhang. Cloudprophet: towards application performance prediction in cloud. In *Proceedings of the ACM SIGCOMM 2011 conference*, SIGCOMM '11, New York, NY, USA, 2011. ACM.
13. KV Mardia, JT Kent, and JM Bibby. Multivariate analysis. *Probability and Mathematical Statistics, London: Academic Press, 1979*, 1, 1979.
14. Daniel Moldovan, Georgiana Copil, Hong-Linh Truong, and Schahram Dustdar. Mela: Monitoring and analyzing elasticity of cloud services. In *2013 IEEE Fifth International Conference on Cloud Computing Technology and Science (CloudCom)*, 2013.
15. OASIS Cloud Application Management Platforms (CAMP). https://www.oasis-open.org/committees/camp.
16. OASIS Committee Specification Draft 01. Topology and Orchestration Specification for Cloud Applications Version 1.0. 2012.
17. Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, pages 5:1–5:14. ACM, 2011.
18. Bing Tang and Mingdong Tang. Bayesian model-based prediction of service level agreement violations for cloud services. In *Theoretical Aspects of Software Engineering Conference (TASE), 2014*, pages 170–176, Sept 2014.
19. Demetris Trihinas, George Pallis, and Marios D. Dikaiakos. JCatascopia: Monitoring Elastically Adaptive Applications in the Cloud. In *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 2014.
20. Akshat Verma, Gautam Kumar, and Ricardo Koller. The cost of reconfiguration in a cloud. In *Proceedings of the 11th International Middleware Conference Industrial Track*, pages 11–16, New York, NY, USA, 2010. ACM.
21. Wei Wang, Baochun Li, and Ben Liang. To reserve or not to reserve: Optimal online multi-instance acquisition in iaas clouds. In *Presented as part of the 10th International Conference on Autonomic Computing*, pages 13–22, Berkeley, CA, 2013. USENIX.
22. Li Zhang, Xiaoqiao Meng, Shicong Meng, and Jian Tan. K-scope: Online performance tracking for dynamic cloud applications. In *Presented as part of the 10th International Conference on Autonomic Computing*, pages 29–32, Berkeley, CA, 2013. USENIX.