# ADVISE – A Framework for Evaluating Cloud Service Elasticity Behavior*

Georgiana Copil[1], Demetris Trihinas[2], Hong-Linh Truong[1], Daniel Moldovan[1], George Pallis[2], Schahram Dustdar[1], and Marios Dikaiakos[2]

[1] Distributed Systems Group, Vienna University of Technology
{e.copil,d.moldovan,truong,dustdar}@dsg.tuwien.ac.at
[2] Computer Science Department, University of Cyprus
{trihinas,gpallis,mdd}@cs.ucy.ac.cy

**Abstract.** Complex cloud services rely on different elasticity control processes to deal with dynamic requirement changes and workloads. However, enforcing an elasticity control process to a cloud service does not always lead to an optimal gain in terms of quality or cost, due to the complexity of service structures, deployment strategies, and underlying infrastructure dynamics. Therefore, being able, a priori, to estimate and evaluate the relation between cloud service elasticity behavior and elasticity control processes is crucial for runtime choices of appropriate elasticity control processes. In this paper we present ADVISE, a framework for estimating and evaluating cloud service elasticity behavior. ADVISE gathers service structure, deployment, service runtime, control processes, and cloud infrastructure information. Based on this information, ADVISE utilizes clustering techniques to identify cloud elasticity behavior produced by elasticity control. Our experiments show that ADVISE can estimate the expected elasticity behavior, in time, for different cloud services thus being a useful tool to elasticity controllers for improving the quality of runtime elasticity control decisions.

## 1 Introduction

One of the key features driving the popularity of cloud computing is elasticity, that is, the ability of cloud services to acquire and release resources on-demand, in response to runtime fluctuating workloads. From customer perspective, resource auto-scaling could minimize task execution time, without exceeding a given budget. From 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.

A common approach, employed by many elasticity controllers [1, 2] is to monitor the cloud service and (de-)provision virtual instances when a metric threshold is violated. This approach may be sufficient for simple service models

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

X. Franch et al. (Eds.): ICSOC 2014, LNCS 8831, pp. 275–290, 2014.

but, when considering large-scale distributed cloud services with various interdependencies, a much deeper understanding of its elasticity behavior is required. For this reason, existing work [2, 3] 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*? Both cloud customers and providers can 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 of 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 runtime decision making.

To this end, a wide range of approaches relying on service profiling or learning from historic information [3–5] have been proposed. 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 based on cloud service behavior at multiple levels (e.g., per node, tier, entire service). Additionally, current approaches only evaluate resource utilization, without considering elasticity as a multi-dimensional property composed of three dimensions (cost, quality, and resource elasticity). Finally, existing approaches do not consider the outcome of a control process on the overall service, where often enforcing a control process to 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 our previous work, we focused on modeling current and previous behavior with the concepts of elasticity space and pathway [6], or using different algorithms to determine enforcement times in observed behavior (e.g., with change-point detection), but without modeling expected behavior of different service parts, in time.

In this paper, we focus on addressing the limitations above by introducing the ADVISE (*evAluating clouD serVIce elaSticity bEhavior*) framework, which estimates cloud service elasticity behavior by utilizing different types of information, such as service structure, deployment strategies, and underlying infrastructure dynamics, when applying different external stimuli (e.g., elasticity control processes). At the core of ADVISE is a clustering-based evaluation process which uses these types of information for computing expected elasticity behavior, in time, for various service parts. To evaluate ADVISE effectiveness, experiments were conducted on a public cloud platform with a testbed comprised of two different cloud services. Results show that ADVISE outputs the expected elasticity behavior, in time, for different services with a low estimation error rate. ADVISE can be integrated by cloud providers alongside their elasticity controllers to improve their decision quality, or used by cloud service providers to evaluate and understand how different elasticity control processes impact their services.

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 evaluate ADVISE framework effectiveness. In section 5 we discuss related work. Section 6 concludes this paper.

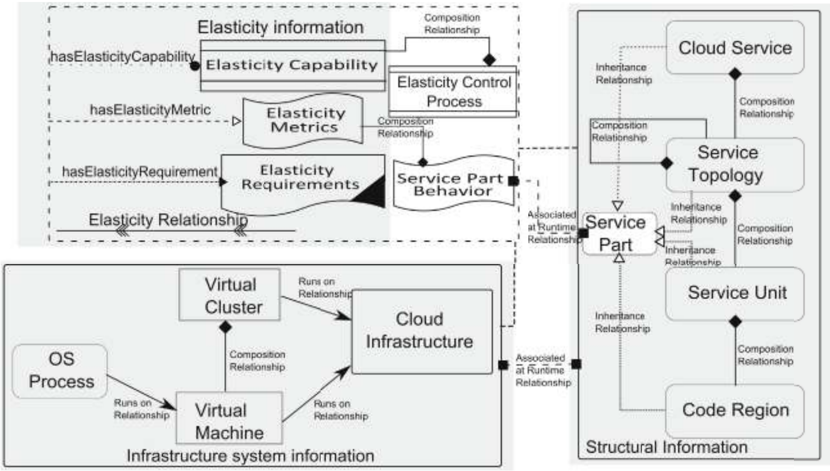


**Fig. 1.** Elasticity capabilities exposed by different elastic objects

## 2 Cloud Service Structural and Runtime Information

### 2.1 Cloud Service Information

To follow existing common service descriptions [7], we refer to a cloud application in our study as a *cloud service*. A cloud service can be decomposed into *service topologies* (e.g., a business tier, or a part of a workflow) which represent a group of semantically connected service units. A *service unit* (e.g., a web service) represents a module offering computation or data capabilities. In order to refer to these cloud service structures globally, we use the term *Service Parts* ($SP$).

We extend the conceptual cloud service representation model proposed in [8] with a rich set of information types for determining cloud elasticity behavior. Fig. 1 depicts the extensions we made (white background) to include *elasticity control processes*, *service part behaviors* and *service parts*. Overall, this representation contains: (i) *Structural Information*, describing the architectural structure of the application to be deployed on the cloud, (ii) *Infrastructure System Information*, describing runtime information regarding resources allocated by the cloud service from the underlying cloud platform, and (iii) *Elasticity Information*, which is associated with both structural and infrastructure system information for describing elasticity *metrics*, *requirements*, and *capabilities*.

Elasticity information is composed of elasticity metrics, elasticity requirements, and elasticity capabilities, each of them being associated to different $SPs$ or infrastructure resources. *Elasticity Capabilities* are grouped together as *Elasticity Control Processes* ($ECPs$), as described 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 (e.g., before allocating a new database node, the effect at the database service topology and at the entire cloud service level should also be determined). Conceptually, a *Service Part*

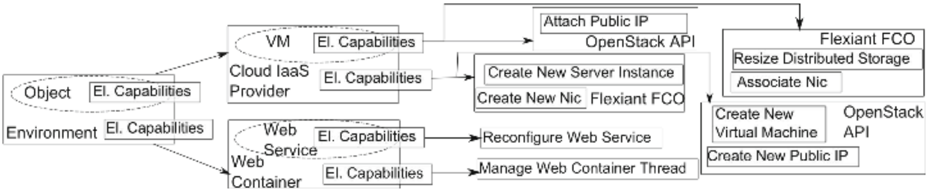


**Fig. 2.** Elasticity capabilities exposed by different elastic objects

*Behavior*, denoted as $Behavior_{SP_i}$, for a specific $SP_i$ 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:

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

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

$$Behavior_{CloudService}[start, end] = \{Behavior_{SP_i}[start, end] | SP_i \in ServiceParts(CloudService)\} \quad (3)$$

The above information is captured and managed at runtime through an *Elasticity Dependency Graph*, which has as nodes instances of concepts from the model presented in Fig. 1 (e.g., `Virtual Machine`), and relationships (e.g., `Elasticity Relationship`) as edges. The elasticity dependency graph is populated and continuously updated with (i) *pre-deployment* information, such as service topology descriptions (e.g., TOSCA [7]) or profiling information; and (ii) *runtime* information such as metric values from monitoring tools or allocated resources information from cloud provider APIs.

### 2.2 Elasticity Control Processes

*Elasticity capabilities* ($ECs$) are the set of actions associated with a cloud service, which a cloud service stakeholder (e.g., an elasticity controller) may invoke, and which affect the behavior of a cloud service. Such capabilities can be exposed by: (i) different $SPs$, (ii) cloud providers, or (iii) resources which are supplied by cloud providers. An EC can be considered as the abstract representation of API calls, which differ amongst providers and cloud services. Fig. 2 depicts the different subsets of $ECs$ provided for an exemplary web application when deployed on two different cloud platforms (e.g., Flexiant, and Openstack private cloud), as well as the $ECs$ exposed by the cloud service and the installed software. In each of the two aforementioned cloud platforms, the cloud service needs to run on a specific environment (e.g., Apache Tomcat web server), and all these capabilities, when enforced by an elasticity controller, will have an effect on various parts of the cloud service. For instance, even if not evident at first sight, elasticity capabilities of a web server topology of the cloud service could also affect the performance of its database backend.

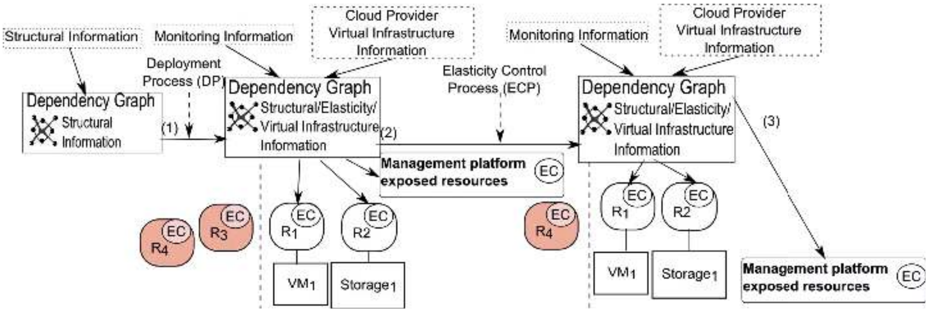


**Fig. 3.** Elastic cloud service evolution

*Elasticity Control Processes* ($ECP$) are sequences of elasticity capabilities $ECP_i = [EC_{i_1} \rightarrow EC_{i_2} \rightarrow ... \rightarrow EC_{i_n}]$, which can be abstracted into higher level capabilities having predictable effects on the cloud service. An $ECP$ causes a change in the elasticity dependency graph and in the virtual infrastructure related information (e.g., change in $ECP$ properties or in the properties of the VM). For example, in the case of a distributed database backend which is composed of multiple nodes, a scale out $ECP$, with certain parameters, can apply 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 during Runtime

To be able to estimate the effects of $ECPs$ upon $SPs$, we rely on the elasticity dependency graph which captures all the variables that contribute to cloud service elasticity behavior evolution. Fig. 3 depicts on the left-hand side the cloud service at a pre-deployment time, where automatic elasticity controllers know about it only from structural information provided by different sources (e.g., TOSCA service description). After enforcing a Deployment Process (e.g., create machine x, and configure software z), the elasticity dependency graph will additionally contain infrastructure-related information obtained from the cloud provider, and elasticity information, obtained from monitoring services showing the metrics evolution for different $SPs$. This information is continually updated during runtime (step 3 in Fig. 3), while for estimating the behavior we make the assumption that we have complete information (i.e., no information missing).

Infrastructure resources, as mentioned previously, have associated elasticity capabilities ($EC$ in Fig. 3), 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 $SPs$,

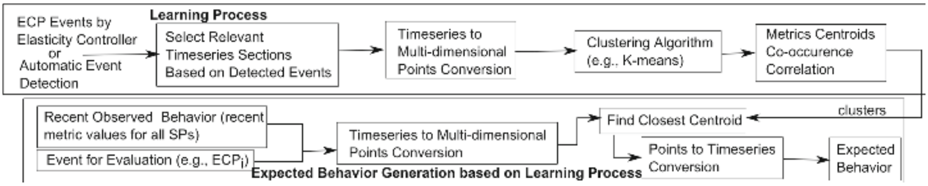


**Fig. 4.** Modeling cloud service behavior process

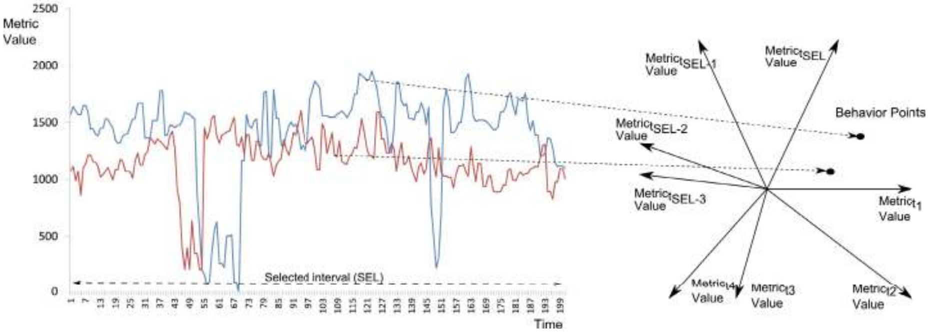


**Fig. 5.** Relevant timeseries sections to points

regardless on whether the $ECP$ is application specific, or it does not have any apparent link to other $SPs$. In fact, as we show in Section 4, the impact of various $ECPs$ over different $SPs$ and over the entire cloud service is quite interesting.

## 3 Evaluating Cloud Service Elasticity Behavior

Existing behavior learning solutions [4, 5] learn discrete metric models, without correlating them with the multiple variables which affect cloud service behavior. As opposed to them, we are learning the behavior of different cloud service parts, and their relation to different $ECPs$, not only with directly linked ones, and estimating the effect of an $ECP$, *in time*, considering the correlations among several metrics and among several service parts. The *Learning Process* used to determine cloud service part behavior is depicted in Fig. 4, and is executed continuously, refining the previously gathered knowledge base.

### 3.1 Learning Process

**Processing input data.** Our learning process takes as input each metric's evolution, in time, $M_a^{SP_i}[start, current]$ (see Equation 3) from the beginning of the service execution on the current cloud platform. To evaluate the expected evolution of metrics in response to enforcing a specific $ECP$, we select for each monitored metric, of each service part, a *Relevant Timeseries Section* ($RTS$), in order to compare it with previously encountered $M_a^{SP_i}[start, current]$. The $RTS$ size strongly depends on the average time needed to enforce an $ECP$

(see Section 4.3). Consequently, a metric $RTS$ is a sub-sequence of the $M_a^{SP_i}$, from before enforcing an $ECP$ until after the enforcement is over:

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

, where $x$ is the $ECP$ index and $\delta$ is the length of the period we aim to evaluate.

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

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

**Clustering process.** To detect the expected behavior, as a possible result of enforcing an $ECP$, we construct clusters of behavioral points $Cluster_{SP_i}$ for all $SPs$ and each $ECP$ based on the distance between behavior points 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. The objective function of this process is finding the multi-dimensional behavior point $C(\Theta^*)$, which minimizes the distance among points belonging to the same cluster $Cluster_k$ (see Equation 7). Since the focus of this paper is not to evaluate the quality of different clustering algorithms, we choose to use the K-means algorithm, following the practice where the number of clusters is $K = \sqrt{N/2}$, $N$ being the number of objects. However, as shown in Section 4, even with a simple K-means algorithm, our approach outputs the expected elasticity behavior with a low estimation error rate.

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

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

After obtaining $\delta + ECP_{time}$-dimensional point clusters, we construct for each $SP_i$ a correlation matrix, $CM_{SP_i}[C_x, C_y]$, where $C_x$ is the centroid of $Cluster_x$, giving the probability, for all metrics, of clusters from different metrics to appear together (e.g., increase in data reliability is usually correlated with increase in cost). An item in the $CM$ represents a ratio between the number of times the behavior points $C_x$ and $C_y$ were encountered together towards 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.

### 3.2 Determining the Expected Elasticity Behavior

In the *Expected Behavior Generation based on Learning Process* step in Fig. 4, we select latest metrics values for each $SP_i$, $M_a^{SP_i}[current - \delta, current]$, and the $ECP_\xi$ 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 is, for each metric of $SP_i$, a list of expected values from the enforcement of $ECP_\xi$ (e.g., expected values for each metrics for the case the user would like to deploy one new web service of type x in the same web application container).

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

The above process is executed continuously, as shown in Fig. 4, by refining clusters, re-computing cluster centroids with the time and with the enforcement of new $ECPs$. This process is highly flexible and configurable, as we can use different manners of detecting $ECPs$ (e.g., sent by the elasticity controller), or other clustering algorithms which lead to different solutions.

## 4 Experiments

To evaluate the effectiveness of the proposed approach, we have developed the ADVISE framework[1] which incorporates the previously described concepts. Current ADVISE version gathers various types of information to populate the elasticity dependency graph, such as: (i) *Structural information*, from TOSCA service descriptions; (ii) *Infrastructure and application performance information* from JCatascopia [9] and MELA [6] monitoring systems; (iii) *Elasticity information* regarding ECPs from the rSYBL [8] elasticity controller where we developed an enforcement plugin to randomly enforce ECPs on cloud services. To evaluate the functionality of the ADVISE framework, we established a testbed comprised of two services deployed on the Flexiant public cloud. On both cloud services, we enforce random $ECPs$ exposed by different $SPs$. We do not use 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.

ADVISE currently receives monitoring information in two formats: (i) as simple `*.csv` files, or (ii) automatically pulling monitoring information from MELA. ADVISE can be used both in service profiling/pre-deployment phase or during runtime, for various service types, whenever monitoring information and enforced $ECPs$ are available for generating estimations for various metrics of service parts.

[1] Code & documents: `http://tuwiendsg.github.io/ADVISE`

**Table 1.** Elasticity control processes available for the two cloud services

| 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 instance |
| | $ECP_2$ | *Scale Out Application Server Tier*: (i) create new network interface, (ii) instantiate new virtual machine, (ii) deploy and configure video streaming service, (iv) deploy and start JCatascopia Monitoring Agent, (v) add instance IP to HAProxy, (vi) restart HAProxy |
| | $ECP_3$ | *Scale In Distributed Video Storage Backend*: (i) select instance to remove, (ii) decommission instance data to other nodes (using Cassandra nodetool API), (iii) stop JCatascopia Monitoring Agent, (iv) delete instance |
| | $ECP_4$ | *Scale Out Distributed Video Storage Backend*: (i) create new network interface, (ii) instantiate new instance, (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 virtual machine |
| | $ECP_6$ | *Scale Out Event Processing Service Unit*: (i) create new network interface, (ii) create new virtual machine, (iii) add service IP to HAProxy configuration file |
| | $ECP_7$ | *Scale In Data Node Service Unit*: (i) decommision node (copy data from virtual machine to be removed), (ii) remove recursively virtual machine |
| | $ECP_8$ | *Scale Out Data Node Service Unit*: (i) create new network interface, (ii) create virtual machine, (iii) set ports, (iv) assign token to node, (v) set cluster controller, (vi) start Cassandra |

### 4.1 Experimental 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 (i.e., download, or upload video) across application servers; (ii) An *Application Server Tier*, where each application server is an Apache Tomcat server containing the video streaming web service; (iii) A Cassandra *NoSQL Distributed Data Storage Backend* from where the necessary video content is retrieved. We have evaluated the ADVISE framework by generating client requests under a stable rate, where the load depends on the type of the requests and the size of the requested video, as shown in the workload pattern in Fig.6.

The second service in our evaluation is a Machine-to-Machine (M2M) DaaS which processes information originating from several different types of 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. 6) which is processed by the *Event Processing Service Topology*, and stored/retrieved from

**Table 2.** Elasticity metrics for different service parts

| Cloud Service | SP Name | Metrics |
|---|---|---|
| Video Service | Application Server Tier | cost, busy thread number, memory utilization, request throughput |
| | Distributed Video Storage Backend | cost, CPU usage, memory usage, query latency |
| M2M DaaS | Cloud Service | cost per client per hour (Cost/Client/h) |
| | Event Processing Service Topology | cost, response time, throughput, number of clients |
| | Data End Service Topology | cost, latency, CPU usage |

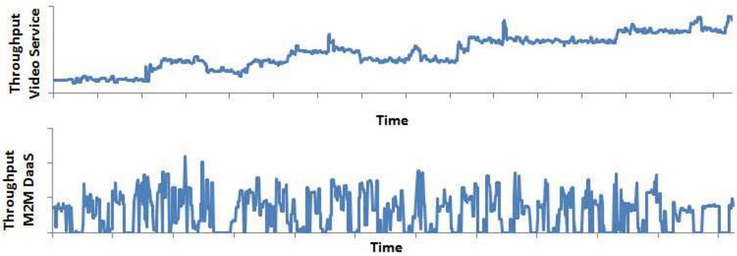


**Fig. 6.** Workload applied on the two services

the *Data End Service Topology*. Tables 1 and 2 list the $ECPs$ associated to each $SP$ and the monitoring metrics analyzed for the two cloud services respectively.

### 4.2 Elasticity Behavior Estimation

**Online Video Streaming Service.** Fig. 7 depicts both the *observed* and the *estimated behavior* for the Application Server Tier of the cloud service when a *remove application server from tier* ECP occurs ($ECP_1$). 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. We observe that after the `scale in` action 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 when deciding which $ECP$ to enforce for this behavior.

Similarly, in Fig. 8 we depict both the *observed* and the *estimated behavior* for the Distributed Video Storage Backend when a `scale out` action occurs (add Cassandra node to ring) due to high `CPU utilization`. We observe that after the `scale out` action occurs, the actual `CPU utilization` decreases to a normal value as also indicated by the estimation. Finally, from Fig. 7 and 8,

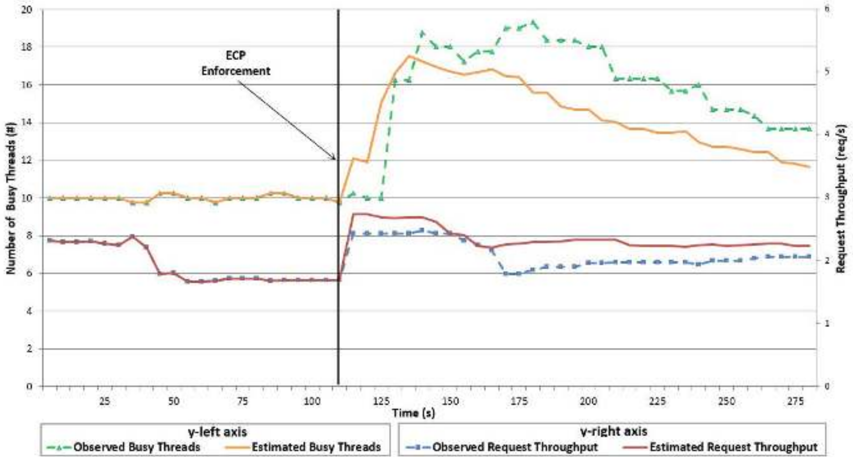


**Fig. 7.** Effect of $ECP_1$ on the application server tier

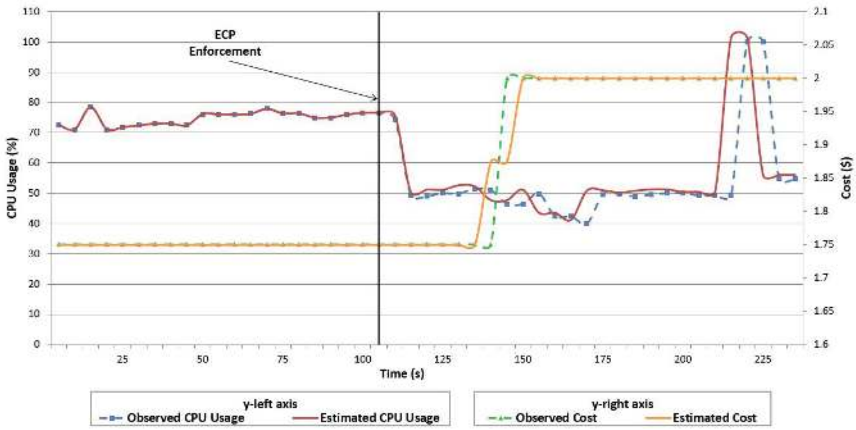


**Fig. 8.** Effect of $ECP_4$ on the entire video streaming service

we conclude that the ADVISE estimation successfully follows the actual behavior pattern and that in both cases, as time passes, the curves tend to converge.

**M2M DaaS.** Fig. 9 shows 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 in terms of *expected time* and *expected metric fluctuations*. This information is important for elasticity controllers to improve their decisions when enforcing this $ECP$ by knowing how the `Cost/Client/h` for the entire cloud service would be affected. Although the `CPU usage` is not estimated perfectly, since it is a highly oscillating metric, and it depends on the

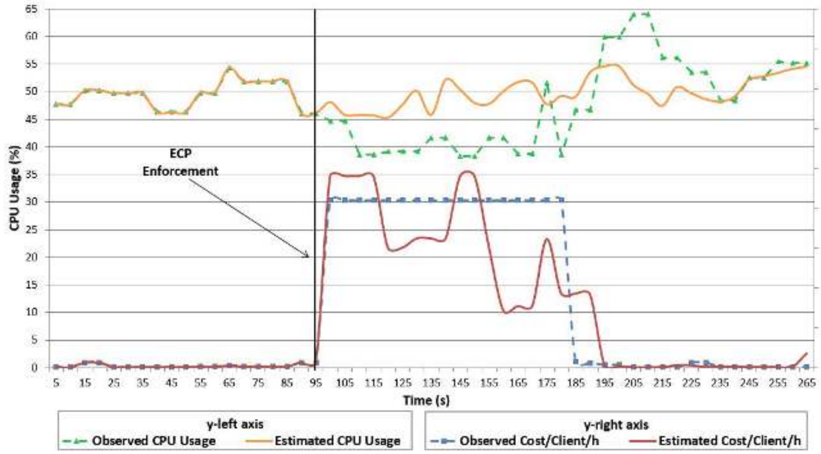


**Fig. 9.** Effect of $ECP_7$ on M2M DaaS

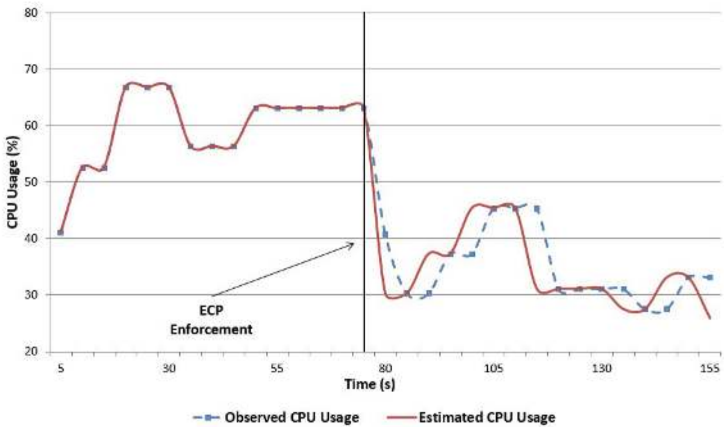


**Fig. 10.** Effect of $ECP_8$ on the data controller service unit

`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).

ADVISE can estimate the effect of an $ECP$ of a $SP$, on a different $SP$, even if apparently unrelated. Fig. 10 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` drops, since the new node is added to the ring, and a lot of effort goes for transferring data to the new node, then it raises due to the fact that reconfigurations are also necessary on the controller, following a slight decrease and stabilization. Therefore, even in circumstances of random workload, ADVISE can give useful insights on how different $SPs$ behave when enforcing $ECPs$ exposed by other $SPs$.

**Table 3.** Elasticity control processes time statistics

| | ECP | Standard Deviation | Average ECP Time (s) |
|---|---|---|---|
| Video Service | ECP1 | 0 | 65 |
| | ECP2 | 0 | 15 |
| | ECP3 | 0 | 25 |
| | ECP4 | 1.414 | 150 |
| M2M Service | ECP1 | 4.5 | 45 |
| | ECP2 | 1.4 | 20 |
| | ECP3 | 0 | 20 |
| | ECP4 | 1 | 75 |

### 4.3 ECP Temporal Effect

Table 3 presents the *average time* required for an ECP to be completed. 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 defines the time granularity of which resizing actions should be taken into consideration. For example, we observe that the process of adding and configuring a new instance to the video service's storage backend requires an average time interval of 150 seconds which is mainly 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.

### 4.4 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 choose to evaluate ADVISE by computing the variance $Var$ and standard deviation $StdDev$ (Equation 9), over 100 estimations as the result differs little afterwise.

$$Var_{metric_i} = \frac{\sum (estMetric_i - obsMetric_i)^2}{nbEstimations - 1}, StdDev_{metric_i} = \sqrt{Var_{metric_i}} \tag{9}$$

Table 4 presents the accuracy of our results. When comparing the two 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.

**Table 4.** ECPs effect estimation quality statistics

| 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 |

Overall, even in random cloud service load situations, the ADVISE framework analyses and provides accurate information 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. Without this kind of estimation, elasticity controllers would need to use VM-level profiling information, while they have 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 Related Work

Verma et al. [3] 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. [10] 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. [4] propose algorithms for performance tracking of dynamic cloud applications, predicting metrics values like throughput or response time. Shen et al. [5] 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. Juve et al. [11] propose a system which helps at 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 CloudProphet framework, which uses resource events and dependencies among them for predicting web application performance on the cloud.

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.

## 6 Conclusions and Future Work

We have presented ADVISE framework, which is able to estimate the behavior of cloud service parts, in time, when enforcing various $ECPs$, by taking into consideration different types of information represented through the elasticity dependency graph. Based on results from two different cloud services, we show that ADVISE framework is indeed able to *advise* elasticity controllers about cloud service behavior, contributing towards improving cloud service elasticity.

As future work, we intend to integrate ADVISE with the rSYBL elasticity controller [8] and develop new decision mechanisms that take continuous $ECP$ effects as inputs, taking decisions based on the expected behavior of each $SP$.

## References

1. Al-Shishtawy, A., Vlassov, V.: 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 2013, pp. 115–116. ACM, New York (2013)
2. Wang, W., Li, B., Liang, B.: To reserve or not to reserve: Optimal online multi-instance acquisition in IaaS clouds. Presented as part of the 10th International Conference on Autonomic Computing, Berkeley, CA, USENIX, pp. 13–22 (2013)
3. Verma, A., Kumar, G., Koller, R.: The cost of reconfiguration in a cloud. In: Proceedings of the 11th International Middleware Conference Industrial Track. Middleware Industrial Track 2010, pp. 11–16. ACM, New York (2010)
4. Zhang, L., Meng, X., Meng, S., Tan, J.: K-scope: Online performance tracking for dynamic cloud applications. Presented as part of the 10th International Conference on Autonomic Computing, Berkeley, CA, USENIX, pp. 29–32 (2013)

5. Shen, Z., Subbiah, S., Gu, X., Wilkes, J.: Cloudscale: elastic resource scaling for multi-tenant cloud systems. In: Proceedings of the 2nd ACM Symposium on Cloud Computing, SOCC 2011, pp. 5:1–5:14. ACM, New York (2011)
6. Moldovan, D., Copil, G., Truong, H.L., Dustdar, S.: Mela: Monitoring and analyzing elasticity of cloud services. In: 2013 IEEE Fifth International Conference on Cloud Computing Technology and Science, CloudCom (2013)
7. OASIS Committee Specification Draft 01: Topology and Orchestration Specification for Cloud Applications Version 1.0 (2012)
8. Copil, G., Moldovan, D., Truong, H.-L., Dustdar, S.: Multi-level Elasticity Control of Cloud Services. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) ICSOC 2013. LNCS, vol. 8274, pp. 429–436. Springer, Heidelberg (2013)
9. Trihinas, D., Pallis, G., Dikaiakos, M.D.: JCatascopia: Monitoring Elastically Adaptive Applications in the Cloud. In: 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (2014)
10. Kaviani, N., Wohlstadter, E., Lea, R.: Profiling-as-a-service: Adaptive scalable resource profiling for the cloud in the cloud. In: Kappel, G., Maamar, Z., Motahari-Nezhad, H.R. (eds.) Service Oriented Computing. LNCS, vol. 7084, pp. 157–171. Springer, Heidelberg (2011)
11. Juve, G., Deelman, E.: Automating application deployment in infrastructure clouds. In: Proceedings of the 2011 IEEE Third International Conference on Cloud Computing Technology and Science, CLOUDCOM 2011, pp. 658–665. IEEE Computer Society, Washington, DC (2011)
12. Li, A., Zong, X., Kandula, S., Yang, X., Zhang, M.: Cloudprophet: towards application performance prediction in cloud. In: Proceedings of the ACM SIGCOMM 2011 Conference, SIGCOMM 2011. ACM, New York (2011)