ADVISE: Evaluating Cloud Service Elasticity Behavior*

Georgiana Copil[1], Demetris Trihinas[2], Daniel Moldovan[1], Hong-Linh Truong[1],

George Pallis[2], Schahram Dustdar[1], 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 changes of requirements and loads. 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 determining which elasticity control processes are the most appropriate. In this paper we present ADVISE, a framework for estimating and evaluating cloud service elasticity behavior which can be integrated by cloud providers alongside their elasticity controllers to improve the quality of their elasticity control decisions. Experiments show that ADVISE estimates the expected elasticity behavior, in time, for different cloud services.

1 Introduction

One of the key features driving the popularity of cloud computing is elasticity, the ability of cloud services to acquire and release resources on-demand in response to workloads whose requirements fluctuate over time. From the customers perspective, auto-scaling resources allows users to minimize the execution time of their tasks without exceeding a given budget. From the cloud providers 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, a number of elasticity controllers exist [1–3]. A common approach, followed by many elasticity controllers, is to monitor the cloud service and when a metric threshold is violated then (de-)provision virtual instances. This approach may be sufficient for simple service models but when considering large-scale distributed cloud services with various inter-dependencies, a much deeper understanding of its elasticity behavior is required. For this reason, existing

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

work [4] [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 is *which elasticity control processes are the most appropriate for a cloud service*? 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 on each part of the cloud service.* Therefore, knowing in advance the cloud service elasticity behavior towards an outside stress stimuli can be of great assistance to an elasticity controller, thus, improving the quality of the elasticity control decision.

To this end, a wide range of approaches have been proposed which rely on cloud service profiling or learning from historic information [4] [5] [6]. However, these approaches limit the decision process to evaluating only low-level VM metrics (i.e. CPU, memory utilization). Furthermore, current methods do not support multi-grain elasticity decisions where decisions are taken by studying the cloud service's behavior at multiple levels (i.e. elasticity behavior per node, tier, entire cloud service). Additionally, current approaches only evaluate resource utilization rather than considering elasticity as a multi-dimensional property [7] composed of three dimensions (cost, quality and resource elasticity) and therefore, evaluate the relations between the three dimensions. Finally, the existing approaches do not consider the outcome of a decision on the overall cloud service where in many cases, enforcing an elasticity control action to the wrong part of the cloud service, can lead to side effects, such as increasing the cost or decreasing the throughput of the overall cloud service.

In this paper, we focus on addressing the above limitations by introducing the ADVISE (*evAluating clouD serVIce elaSticity bEhavior*) framework, which can estimate the behavior of cloud services when different external stimuli (e.g., control processes) are applied. ADVISE can be integrated by cloud providers alongside their elasticity controllers to improve elasticity control decision quality. ADVISE is based on the elasticity control model defined in [8], which we represent at runtime as a runtime dependency graph, and continually update with information relevant for determining cloud service elasticity behavior. Our second main focus in this paper is to evaluate the effectiveness of the proposed framework. Experiments were conducted on a public cloud platform with a testbed comprised of two different domains of cloud services. Results show that our method outputs the expected elasticity behavior, in time, for different cloud services with a low estimation error rate.

The rest of this paper is structured as follows: in Section 2 we model relevant information regarding the cloud service, we describe the evaluation process in Section 3, present experiments in Section 4 and conclusions in Section 5.

2 Cloud Service Structural and Runtime Information

2.1 Cloud Service Information

Fig. 1 shows the conceptual cloud service representation proposed in [8], with few extensions (on white background) which help us to determine cloud service

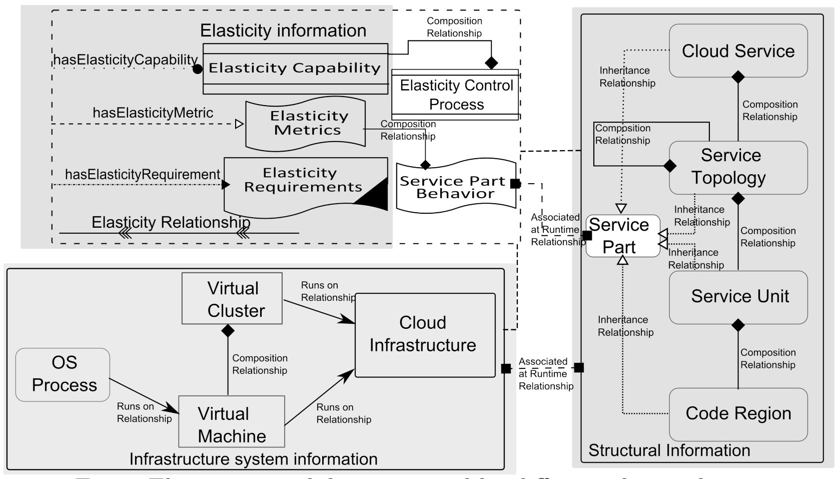

Fig. 1: Elasticity capabilities exposed by different elastic objects

behavior. The cloud service representation contains: (i) *Structural Information* regarding the cloud service, which describes the architectural structuring of the application to be deployed on the cloud, (ii) *Infrastructure System Information*, which gives runtime information regarding resources which are used by the service from cloud providers, and (iii) *Elasticity Information*, which can be associated with both structural information and infrastructure system information for describing elasticity metrics, requirements, and capabilities. The cloud service is composed of several service units [9], which can be grouped together into service topologies, which can at their turn be grouped into other service topologies until we reach the highest level of abstraction, the cloud service level. We refer to service units, service topologies, cloud services and code regions as *Service Parts* (SP).

The Elasticity Information is composed of elasticity metrics, elasticity requirements, and elasticity capabilities, each of them being associated to different service parts or infrastructure resources. The elasticity capabilities are usually grouped together as *Elasticity Control Processes*, as detailed in the next subsection, and produce specific elasticity behaviors upon the different SPs, which we model as *Service Part Behavior*. The Service Part Behavior for a defined period of time $[start, end]$ defined in Equation 3 contains, for a specific SP, all the metrics which are being monitored for the respective SP, in time and is denoted as $Behavior_{SP_i}[start, end]$. The behavior of the cloud service over a defined period of time $Behavior_{CloudService}[start, end]$ is the set of all behaviors of all SPs of the cloud service. The above information is represented through a graph, where the nodes are the concepts shown in Fig. 1 and the edges are the relationships (e.g., `Elasticity Relationship`, or `Associated at Runtime Relationship`). This *Elasticity Dependency Graph* is populated with information both *pre-deployment*, from sources like cloud service description (e.g., through TOSCA [10]), profiling information, information from cloud

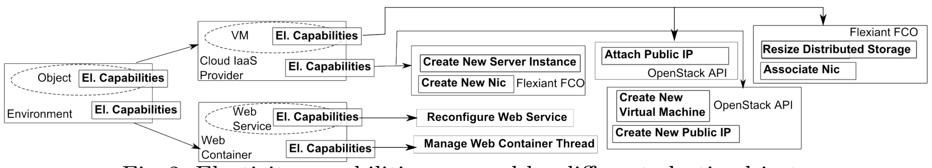

Fig. 2: Elasticity capabilities exposed by different elastic objects

providers, and *during runtime* with information regarding resources used from cloud providers, or elasticity behavior.

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

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

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

2.2 Elasticity control processes

From above model (Fig. 1), the concepts affecting the most $Behavior_{CloudService}$ are *Elasticity Capabilities* (ECs). Elasticity capabilities are the set of actions associated with a cloud service, which a cloud service stakeholder (e.g., an elasticity controller) may invoke, and affect the behavior of the cloud service. Elasticity capabilities can be exposed by: (i) different cloud service parts, (ii) cloud providers or (iii) the resources which are supplied by cloud providers. An Elasticity Capability is the abstract representation of API calls which can differ among providers and cloud services. For instance, Fig. 2 depicts the different subset of ECs provided for an exemplary web application when deployed on two different IaaS providers (Flexiant[3], or Openstack[4] private cloud), as well as the ECs exposed by the cloud service and underlying software. In each of the two mentioned providers, the cloud service needs to run, maybe on some specific environments (e.g., Tomcat web server), and all these capabilities, when enforced by an elasticity controller, will have an effect on the different parts of the cloud service (e.g., even if it is not obvious at first sight, the control actions for the web objects would have an effect on the distributed data store).

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. The ECP causes a change into the elasticity dependency graph in the virtual infrastructure related information (e.g., change in a process properties, or change in the properties of the VM). For instance, for the case of a data end composed of multiple nodes, an ECP of increasing the amount of resources, with certain parameters, would imply both for a Cassandra and an HBase database, adding the new node, and then subscribing to the cluster. The latter two are elasticity capabilities, while

[3] `www.flexiant.com`
[4] `www.openstack.org`

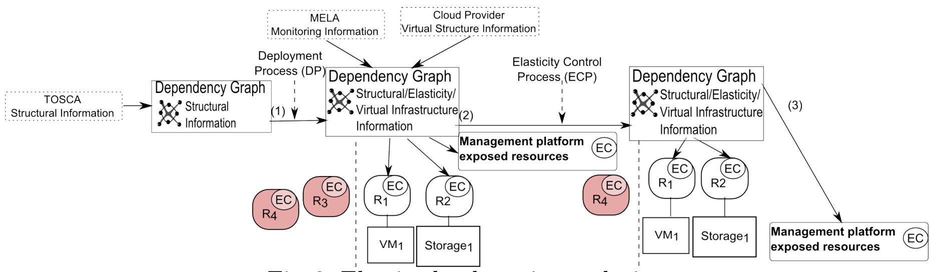
TOSCA Structural Information
Dependency Graph
Structural Information
(1)
Deployment Process (DP)
MELA Monitoring Information
Cloud Provider Virtual Structure Information
Dependency Graph
Structural/Elasticity/ Virtual Infrastructure Information
(2)
Management platform exposed resources
EC
Elasticity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

Fig. 3: Elastic cloud service evolution

the more generic increasing the amount of resources is an ECP, more abstract, which encapsulates the two elasticity capabilities.

2.3 Cloud service elasticity during runtime

In order to be able to estimate the effects of $ECPs$ upon SPs, we describe the cloud service and its environment through the elasticity dependency graph, in order to consider as many as possible from the variables which are contributing to this evolution of the cloud service behavior. Fig. 3 shows in the left side the cloud service at a pre-deployment time, when the automatic controllers know about it only the Structural Information coming from different sources (e.g., TOSCA service description). After enforcing the Deployment Process DP (e.g., create machine x, configure software z), the cloud service has associated Infrastructure System Information, which describes all the resources associated with it, and Elasticity Information, showing the metrics evolution for different SPs, and the possible $ECPs$.

Each of the infrastructure resources have associated Elasticity Capabilities (EC in Fig. 3), that describe the change to be enforced and the mechanisms for triggering it (e.g., which API can be called for making that change). In addition, the Cloud Infrastructure (e.g., Amazon) exposes ECs in order to create new resources or instantiate new services (e.g., increasing the amount of memory can be seen as an EC exposed by a VM, while creating a new VM is an EC exposed by the Cloud Infrastructure). In this context, for being able to discover the effects that an ECP_i produces in time, for each SP, and for all the metrics which are monitored for that part of the service, taking into account correlations between metrics, we need to create our dependency graph, and analyze this information for determining the effect of ECP_i, regardless on whether ECP_i is application specific, or does not appear to influence these parts of the cloud service. In fact, as we show in Section 4, the impact of different $ECPs$ over different cloud SPs or over the entire cloud service can be very interesting.

3 Determining Cloud Service Elasticity Behavior

As opposed to existing behavior learning solutions [5, 6], we are learning the behavior, of different sub-parts of the cloud service, not only the entire cloud

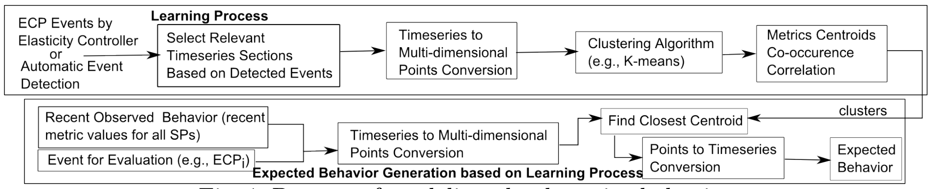
Learning Process
ECP Events by Elasticity Controller or Automatic Event Detection
Select Relevant Timeseries Sections Based on Detected Events
Timeseries to Multi-dimensional Points Conversion
Clustering Algorithm (e.g., K-means)
Metrics Centroids Co-occurence Correlation
clusters
Recent Observed Behavior (recent metric values for all SPs)
Event for Evaluation (e.g., ECP$_i$)
Timeseries to Multi-dimensional Points Conversion
Find Closest Centroid
Points to Timeseries Conversion
Expected Behavior
Expected Behavior Generation based on Learning Process

Fig. 4: Process of modeling cloud service behavior

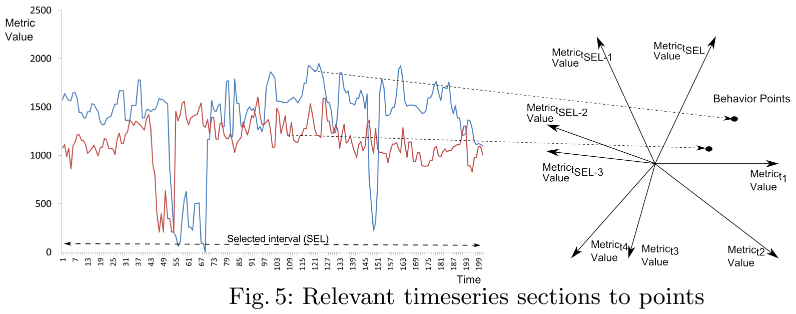

Fig. 5: Relevant timeseries sections to points

service, and their relation to different $ECPs$, not only of the ones associated with the current cloud service part. Moreover, our learning process is estimating the effect of ECP *in time*, and considering the correlations among several metrics and among several service parts. The *Learning Process* executed for determining the cloud service parts behavior is depicted in Fig. 4, and is executing 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. For evaluating the expected evolution of metrics in response to applying a specific ECP, we select for each monitored metric, for 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 enforcing an ECP. A RTS of a metric timeseries is a sub-sequence of the $M^a_{SP_i}$, from before enforcing an ECP until after its enforcement is over:

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

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

,where x is the index of the ECP and δ is the granularity of the relevant section.

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

$$BP : M_{SP} \mapsto BehaviorPointsSpace^{\delta+ECP_{time}}$$
$$BP^{a}_{SP_i}(t) = (RTS^{SP_i}_{M_a}[\theta(0)], ..., RTS^{SP_i}_{M_a}[\theta(\delta + ECP_{time})]) \quad (5)$$

Clustering process For detecting the expected behavior as a result of enforcing an ECP, we construct clusters of behavioral points $Cluster_{SP_i}$ (see Equation 7) for each service part and each ECP, not only $ECPs$ available for the current SP, based on the distance between behavior points defined in Equation 6. Since the focus of current paper is not evaluating the quality of different clustering algorithms, we choose to use a simple K-means algorithm, following the practice of having the number of clusters equal to $\sqrt{n/2}$, n being the number of objects.

$$dist(BP[x], BP[y]) = \sqrt{\sum (BP[x][j] - BP[y][j])^2} \quad (6)$$
$$Cluster^{a}_{SP_i} = \{\cup BP^{a}_{SP_i} | \min_{\forall x,y} dist(BP^{a}_{x}, BP^{a}_{y})\} \quad (7)$$

After obtaining clusters of $\delta + ECP_{time}$-dimensional points, we also create for each SP a correlation matrix, in order to know for all the metrics which clusters from different metrics are probable of appearing together (e.g., increase in data reliability is usually correlated with increase in cost). An item in the correlation matrix $CM_{SP_i}[CC_x, CC_y]$, where CC_x is the centroid of cluster x, would have as value the number of times the behavior points in clusters x respectively y were encountered together. This matrix is continuously refreshed when behavior points move from a cluster to another, or when new $ECPs$ are enforced, increasing the knowledge base.

3.2 Determining the expected behavior

In the *Expected Behavior Generation based on Learning Process* step from Fig. 4, we select latest metrics values for each SPs, $M^{a}_{SP_i}[current-\lambda, current]$, and the ECP 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* which 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.

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

The above process is executed continually, 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 the ECP (e.g., sent by the elasticity controller), or other clustering algorithms which lead to different solutions.

4 Experiments

This section presents results obtained with the ADVISE framework[5] which implements the approach presented in the previous section. The dependency graph (Fig. 1), is populated with various types of information:

- *Structural information* (e.g., TOSCA specification) from the cloud service elasticity controller and deployment tools
- *Infrastructure information* from monitoring tools like JCatascopia [11] and MELA [12])
- *Enforced ECPs* from the elasticity controller that manages the cloud service

To evaluate the functionality and effectiveness of the proposed approach, we have established a testbed comprised of two cloud services deployed on the Flexiant FlexiScale Cloud Platform[6]. On both of the two services, we enforce randomly $ECPs$ exposed by their different SPs, for ensuring a good knowledge base combining different runtime information. We compute for each ECP, at different moments in time with different resources used and different workload, the expected metrics evolution, in time, for each SP of the cloud service.

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*[7] *Load Balancer* which distributes client requests (i.e., download, upload video) across application servers; (ii) An *Application Server Tier*, where each application server is an Apache Tomcat[8] server containing the video streaming web service; (iii) A Cassandra[9] *NoSQL Distributed Data Storage Backend* from where the necessary video content is retrieved. A workload generator which produces random download/upload client requests was utilized to stress the cloud service. We have evaluated the ADVISE framework by generating client requests under a stable rate, the load depending on the type of the request and the requested video.

The second service under evaluation is a Machine-to-Machine (M2M) DaaS which processes information originating from several different types of data sensors (i.e., temperature, atmospheric pressure, or pollution). Specifically, the M2M DaaS is comprised of two service topologies, 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 service topology balancer/controller. To stress this cloud service we generate random sensor event information which is processed by the *Event Processing Service Topology*, and stored/retrieved from the *Data End Service Topology*. Table 2 and 1 show the metrics which we are monitoring and analyzing for the two cloud service, and respectively the $ECPs$ available with its associated SP.

[5]Code & supplementary material: `http://tuwiendsg.github.io/ADVISE`

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

[7] http://haproxy.1wt.eu/

[8] http://tomcat.apache.org/

[9] http://cassandra.apache.org/

Cloud Service	ECP Id	Action Sequence
Video Service	ECP_1	*Scale In Application Server Tier*: (i) select instance to remove, (ii) stop the video streaming service, (iii) remove instance from HAProxy, (iv) 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 instance, (ii) deploy and configure video streaming service to start accepting requests, (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 configuration file, (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

Table 1: Elasticity Control Processes Available for the Two Services

4.2 Behavior Estimation

Online Video Streaming Service Fig. 6 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 issues: (i) the video storage backend is underprovisioned and cannot satisfy the current number of requests which, in turn, results in client requests being queued; (ii) there is, indeed, a sudden drop in client requests which indicates that application servers are not utilized efficiently. We observe that after the scale in action occurs, indeed, the average *request throughput* and *busy thread number* rises which denotes that the estimation is correct and resources are efficiently utilized.

Similarly, the charts in Fig. 7 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). We observe that after the scale

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

Table 2: Elasticity metrics for different service parts

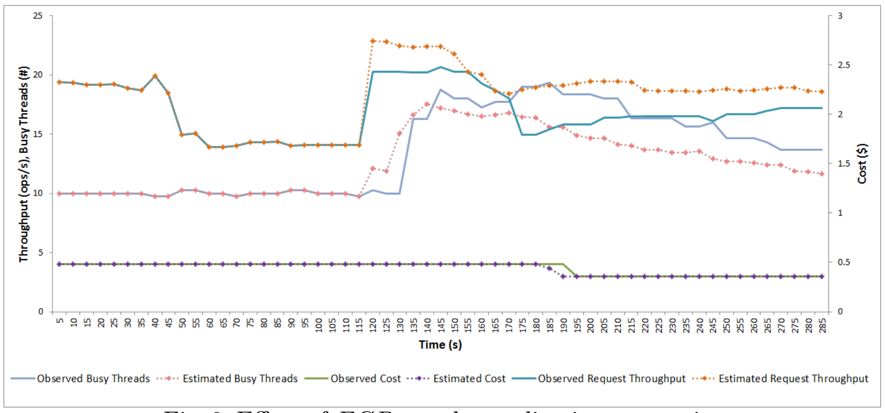

Fig. 6: Effect of ECP_1 on the application server tier

out action occurs, the actual *CPU utilization* decreases to a normal value and our estimation is correct. Finally, from Fig. 6 and 7, we conclude that ADVISE estimation successfully follows the actual behavior pattern and that in both cases, as time passes, the curves tend to converge.

M2M DaaS Fig. 8 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 estimates how the cloud service would behave in terms of expected time and expected metric fluctuations. This information is of tremendous importance for elasticity controllers, that can make informed decisions when enforcing this ECP, 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 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).

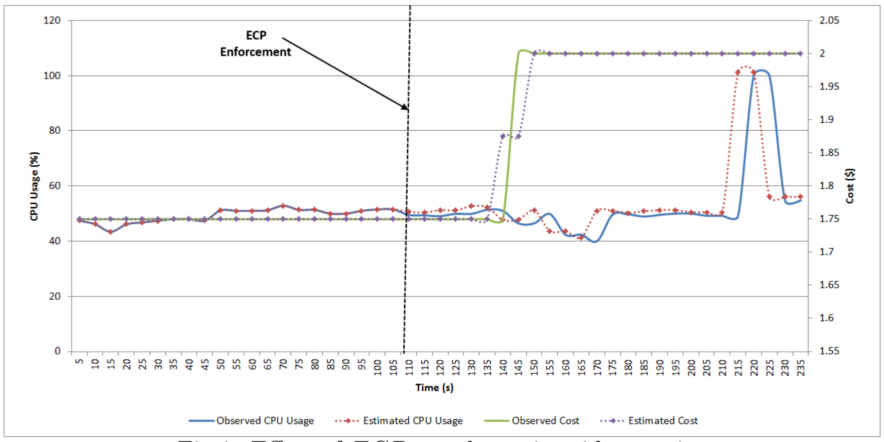

Fig. 7: Effect of ECP_4 on the entire video service

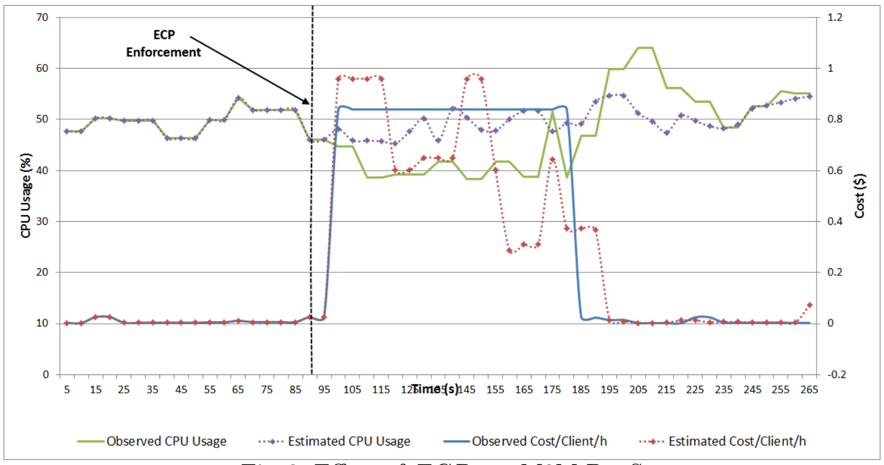

Fig. 8: Effect of ECP_7 on M2M DaaS

Our prototype can estimate the effect of an ECP exposed by a SP has on a different SP, even if apparently unrelated. Fig. 9 shows estimations on how the data controller of the `Data End Service Topology` is impacted by the data transferred at the enforcement of ECP_8. In this case, the CPU usage for the controller drops, since the new node is added to the ring, and a lot of effort in current topology 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, as is the case here, our prototype can give useful insights into how different SPs behave when enforcing $ECPs$ exposed by various SPs.

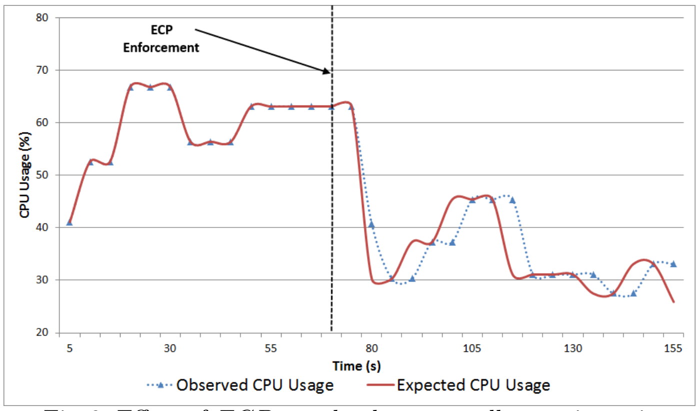

Fig. 9: Effect of ECP_8 on the data controller service unit

	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

Table 3: Elasticity control processes time statistics

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 decision process.

4.4 Quality of results

The ADVISE framework manages to estimate, in time, the elasticity behavior of different parts of a cloud service, in time, considering the correlations among metrics and the *ECPs* which are enforced. To evaluate the quality of results, we compute variance and standard deviations over a number of estimations. We

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

Table 4: ECPs effect estimation quality statistics

have chosen to compute $Variance$ and $StdDeviation$ (Equation 9), over 100 estimations as the result differs little from there on.

$$\begin{aligned} Variance_{metric_i} &= \frac{\sum(estimatedMetric_i - obsMetric_i)^2}{nbEstimations} \\ StdDev_{metric_i} &= \sqrt{Variance_{metric_i}} \end{aligned} \tag{9}$$

Table 4 shows the accuracy of the presented results. When comparing the two cloud services, the Video Service achieves much higher accuracy (smaller variance and standard deviation), since the imposed workload is considerably stable. When 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 also depends on the metrics which we monitor 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`, for which the maximum variance for the *number of clients* metric is 4.9.

Overall, even in random cloud service load situations, ADVISE framework analyses and provides accurate information for elasticity controllers, with regard to the evolution of monitored metrics at the different cloud service levels. Without this kind of estimation, the elasticity controllers usually need to use VM-level profiling information, while they have to control complex cloud services. This information, for each SP, is extremely valuable for elasticity controllers of complex cloud services, which expose complex control mechanisms.

5 Related Work

Verma et al. [4] study the impact of reconfiguration actions on system performance. They observe infrastructure level reconfiguration actions, with actions on live migration, and observe that the 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. Zhang et al. [5] propose algorithms for performance tracking of dynamic cloud applications, predicting metrics values like throughput or response time. Compared with this approach, we also take into consideration in our process the connection among different parts of the application, and we are interested of providing a prediction in time, considering metric correlations from the same part of the application, and with other different parts of the application (e.g., effect of an ECP on a completely different service topology).

Shen et al. [6] 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. [13] 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. [14] propose CloudProphet framework, which uses resource events and dependencies among them for predicting web application performance on the cloud. In contrast with this, our model incorporates different types of both architectural and deployment patterns of cloud computing state of the art.

To enrich existing research, our approach merges these two branches into a single dependency model, in which for estimating application's behavior we take into account both application patterns and the application expected behavior for those patterns in response to external stress factors.

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 a better cloud service elasticity.

As future work, we intend to integrate ADVISE with our 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 '13, New York, NY, USA, ACM (2013) 115–116
2. Tsoumakos, D., Konstantinou, I., Boumpouka, C., Sioutas, S., Koziris, N.: Automated, elastic resource provisioning for nosql clusters using tiramola. In: Cluster, Cloud and Grid Computing (CCGrid), 2013 13th IEEE/ACM International Symposium on. (2013) 34–41
3. Wang, W., Li, B., Liang, B.: 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, Berkeley, CA, USENIX (2013) 13–22
4. 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 '10, New York, NY, USA, ACM (2010) 11–16
5. Zhang, L., Meng, X., Meng, S., Tan, J.: K-scope: Online performance tracking for dynamic cloud applications. In: Presented as part of the 10th International Conference on Autonomic Computing, Berkeley, CA, USENIX (2013) 29–32
6. 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 '11, New York, NY, USA, ACM (2011) 5:1–5:14
7. Dustdar, S., Guo, Y., Satzger, B., Truong, H.L.: Principles of elastic processes. IEEE Internet Computing **15**(5) (sept.-oct. 2011) 66 –71
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.: Service-Oriented Computing. Lecture Notes in Computer Science. Springer Heidelberg
9. Tai, S., Leitner, P., Dustdar, S.: Design by Units: Abstractions for Human and Compute Resources for Elastic Systems. IEEE Internet Computing **16**(4) (2012)
10. OASIS Committee Specification Draft 01: Topology and Orchestration Specification for Cloud Applications Version 1.0. (2012)
11. 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)
12. 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)
13. 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 '11, Washington, DC, USA, IEEE Computer Society (2011) 658–665
14. 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 '11, New York, NY, USA, ACM (2011)