# On Controlling Elasticity of Cloud Applications in CELAR

*Georgiana Copil, Daniel Moldovan, Hung Duc Le, Hong-Linh Truong, Schahram Dustdar*

*Distributed Systems Group, Vienna University of Technology, Austria {e.copil, d.moldovan, d.le, truong, dustdar} @dsg.tuwien.ac.at*

*Chrystalla Sofokleous, Nicholas Loulloudes, Demetris Trihinas, George Pallis, Marios D. Dikaiakos*

*Computer Science Department, University of Cyprus {stalosof, loulloudes.n, trihinas, gpalis, mdd} @cs.ucy.ac.cy*

*Ioannis Giannakopoulos, Nikolaos Papailiou, Ioannis Konstantinou, Dimitrios Tsoumakos*

*Computing Systems Laboratory, National University of Athens {ggian, npapa, ikons,dtsouma} @cslab.ece.ntua.gr*

*Craig Sheridan*
*FLEXIANT*
*csheridan@flexiant.com*

*Evangelos Floros, Christos KK Loverdos*

*Greek Research and Technology Network, Greece*
*{efloros, loverdos}@grnet.gr*

*Kam Star*
*Playgen*
*kam@playgen.com*

*Wei Xing*
*CRUK Manchester Institute*
*wei.xing@cruk.manchester.ac.uk*

## Abstract

*Today's complex cloud applications are composed of multiple components executed in multiple cloud provider environments. For such applications, the possibility to manage and control their cost, quality and resources elasticity is of paramount importance. However, given that the cost of different services offered by cloud providers can vary a lot with their quality/performance, elasticity controllers must consider not only complex, multi-dimensional preferences and provisioning capabilities from stakeholders, but also various runtime information regarding cloud applications and their execution environments. In this chapter, we present the elasticity control approach of the EU CELAR Project, which deals with multi-dimensional elasticity requirements, and ensures multi-level elasticity control for fulfilling user requirements. We show the mechanisms of the CELAR control, from application description to multi-level elasticity control, and showcase the usefulness of CELAR for users, who can use an intuitive, user-friendly interface to describe and then to follow their application elasticity behavior controlled by CELAR.*

## 1 Introduction

With the popularity and diversity of cloud-based solutions, both cloud provider solutions and application deployments on the cloud, there is a considerable need to customize these solutions and to provide cloud users with fine-grained mechanisms of controlling their cloud applications. The concept of multi-dimensional elasticity, covering resources elasticity, cost elasticity and quality elasticity (see Dustdar et al. (2011)) and the relations among them, shows how complex the elasticity control of cloud applications actually is. Such a concept facilitates custom cloud application elasticity depending on what cloud application stakeholder (e.g., service provider) actually needs.

Many existing frameworks allow the specification of various cloud application-related information, like the cloud application complex structure (e.g., Di Nitto et al. (2013)) and functional requirements (e.g., Di Cosmo et al. (2013)) when deploying the cloud application on the cloud. Moreover, tools have been proposed for describing and deploying cloud applications (e.g., Binz et al. (2013)) on different cloud infrastructures. However, current state-of-the-art on elasticity control techniques require the specification of low-level, detailed information, e.g., on the triggering of each specific actions. For instance, Auto Scale applications provided by Amazon[1], Rackspace[2], Azure[3] or RightScale[4] enable users to specify, for each Virtual Machine they are using, scaling policies, depending on IaaS-level metrics. Proposed frameworks take into consideration cloud application level metrics, e.g., response time, but do not allow users to specify their requirements, the optimization factor being defined in an ad-hoc manner (e.g., equilibrium between the cost and response time) (e.g., Serrano et al. (2013), Simjanoska et al. (2013)). However, the requirements of the cloud application stakeholders differ and depend on a number of variables, e.g., the cost of the cloud application reported to the number of clients, or the various cloud application quality parameters (e.g., a banking cloud application differs greatly in requirements from a scientific cloud application).

In this chapter, we present elasticity control techniques developed in the EU CELAR Project[5]. Our techniques enable the cloud application stakeholders to specify the requirements at different granularities and levels and to control cloud applications at multiple levels, applying different types of elasticity control mechanisms suited for data-intensive or compute-intensive parts of the cloud application. CELAR control techniques take real-time decisions for cloud application adaptation to meet user (e.g., application developer, or service provider) elasticity requirements, facilitating an automatic adaptation process of the cloud application to "outside" stimuli (e.g., workload, increasing cost, or decreasing quality) without the need of user intervention. Moreover, not only real-time adaptation decisions are enforced but also smart deployment of the cloud application, considering cloud providers applications and estimated cost with respect to quality and performance.

The rest of this chapter is organized as follows: in Section 2 we compare existing solutions with our approach in the CELAR, in Section 3 we present CELAR users and their possible requirements with regard to the elasticity control, in Section 4 we present our elasticity specification language and show how CELAR's user interface component facilitates the description of multi-level elasticity requirements. The next section, Section 5, presents the conceptual architecture of the CELAR elasticity control module, and its techniques. We present experiments in Section 6, and conclude the chapter in Section 7.

## 2 Related Work

In this section, we take a look at current cloud application elasticity status quo regarding cloud application control. We present the elasticity capabilities of cloud providers which are part of the CELAR project, both on data and on computing resources. Next, we focus on computing resource and data resource elasticity control, and compare the state of the art with what we do for controlling elasticity in CELAR. Finally, we take a look at higher, and multiple level application control existent in literature, and compare our approach with them.

[1] http://aws.amazon.com/autoscaling
[2] http://www.rackspace.com/blog/easily-scale-your-cloud-with-rackspace-auto-scale
[3] http://www.windowsazure.com/en-us/documentation/articles/cloud-services-how-to-scale/#autoscale
[4] http://www.rightscale.com/products/automation-engine.php
[5] http://www.celarcloud.eu/

## 2.1 Computing and Data Resources Low-level Controls

We firstly consider the possibilities of runtime reconfiguration offered by the CELAR cloud providers, Flexiant[6] and ~okeanos[7]. Table 1 presents the fundamental control mechanisms available for computing and network resources, while Table 2 presents data elasticity control mechanisms. Although they have different names for the applications being offered (e.g., VM and Server refer to Virtual Machine), they have similar offerings. For instance, common elasticity control mechanisms are create/start/reboot VM, with minor differences e.g., Flexiant FCO offers Bento Boxes which are complex clusters which can be deployed as a group, while ~okeanos offers the opportunity of constructing arbitrary network topologies. Other big cloud providers (e.g., Google[8], Azure[9], or Amazon[10]) typically offer similar capabilities, in the sense of VM and disk level horizontal or vertical scaling, with variations on hot-pluggable capabilities. Although they currently offer low-level capabilities, there is a considerable effort towards offering services between IaaS and PaaS, e.g., Google managed VMs[11] , part of their PaaS services, facilitate automated management similar with the management offered for manually created VMs.

Table 1: Computing Resources Control Mechanisms

| ***Provider*** | ***Elasticity Capability*** | ***Description*** |
|---|---|---|
| *~okeanos* | *Create New VM* | *Creates a new Virtual Machine from an existing image* |
| | *Start VM* | *Starts an already created virtual machine, booting the OS* |
| | *Shutdown VM* | *Shuts down the operating system and stops the VM* |
| | *Reboot VM* | *Performs an OS restart* |
| | *Destroy VM* | *Deletes the VM* |
| | *Initialize VM Configuration* | *Number of CPUs, Size of RAM, System disk, OS, Network connectivity (dual IPV4/IPV6),* |
| | *Create private virtual L2 network* | *Creating a subnet (e.g., for constructing arbitrary network topologies)* |
| *Flexiant FCO* | *Create Bento Box* | *Template entire complex clusters and deploy at the click of a button* |
| | *Add/ Remove compute nodes to cluster* | *Flexiant offers the possibility of grouping compute nodes into clusters which are controlled/monitored as a group* |
| | *Initialize Server Configuration* | *Number of CPUs, Size of RAM, System disk, OS, Network connectivity (dual IPV4/IPV6), user, password, contextualization information* |
| | *Create Server* | *Creates a new server from an existing image* |
| | *Start Server* | *Starts an already creating server, booting the OS* |
| | *Duplicate Server* | *When Server A is duplicated, a new server (Server B) is created, and the initial configuration of Server A is applied to Server B* |
| | *Shutdown Server* | *Shuts down the operating system and stops the Server* |

[6] http://www.flexiant.com/
[7] https://okeanos.grnet.gr/
[8] https://cloud.google.com/products/compute-engine/
[9] http://azure.microsoft.com
[10] http://aws.amazon.com
[11] https://developers.google.com/cloud/managed-vms

| | | |
|---|---|---|
| | *Reboot Server* | *Performs an OS restart* |
| | *Destroy Server* | *Deletes the Server* |
| | *Manage Firewalls* | *Add/remove/configure firewalls for the server* |
| | *Manage Chef Settings for Server* | *Edit chef account settings* |
| | *Create/Manage Virtual Data Center* | *Virtual Data Center is a logical grouping of servers* |
| *Application Specific* | *Configure software x with configuration y* | *Configure software which is part of the application or on which the application depends, in order to have different quality/performance/cost parameters for the application.* |

Table 2: Data Elasticity Control Mechanisms

| ***Provider*** | ***Elasticity Capability*** | ***Description*** |
|---|---|---|
| *~okeanos* | *Storage Configurations* | *Local, distributed and centralized, out of which both SAN, NAS* |
| | *Volume creation* | *Create volume with specified size* |
| | *Volume deletion* | *Delete specified volume* |
| | *On-the-air attachment of volume* | *Attach volume to existing computing node (VM), without the need of rebooting the node* |
| | *On-the-air de-attachment of volume* | *De-attach volume from existing computing node (VM) without the need of rebooting the node* |
| | *Snapshotting existing volume* | *Create a snapshot of the specified volume (available copy-on-write of snapshotable volumes)* |
| | *Hashing snapshots* | *Facilitates deduplication, thus reducing the storage cost of each hashed object* |
| | *Resizing existing volume* | *Resize volume to specified size* |
| *Flexiant FCO* | *Storage Configurations* | *Three types of storage: local, distributed and centralized, out of which both SAN, NAS* |
| | *Create disk* | *Create disk with specified size* |
| | *Remove disk* | *Remove specified disk* |
| | *Snapshot disk* | *Take a snapshot of the disk* |
| | *Add the disk to a new or existing deployment instance* | *Add existing disk to a deployment instance (group of servers)* |
| *Data Specific* | *Clean Data* | *Remove data which is not valid for improving the data completeness and data access performance* |
| | *Move Data* | *Move data from one disk to another, from one block to another, etc.* |
| | *Other Data Specific Control Mechanisms* | *Reconfigure data in different other ways* |
| *Application Specific* | *Configure software x with configuration y* | *Configure software which is part of the application or on which the application depends, in order to have different quality/performance/cost parameters for the application.* |

## 2.2 Computing Resource Elasticity Control

To leverage the low-level elasticity capabilities of cloud infrastructures, several controllers have been developed. However, managing elasticity of cloud applications by using the most popular mechanisms of computing resources control, is not a trivial task. For small-scale application deployments, organizations can (de-) allocate resources manually, but for large-scale distributed applications which require a deployment comprised of multiple *virtual instances*, which often have complex inter-dependencies, this task must be done, inevitably, automatically.

Current computing elasticity controllers such as Amazon AutoScaling, Paraleap AzureWatch[12] and RightScale[13] can scale – automatically and seamlessly – large Cloud applications. However, their controlling actions are limited to only scaling horizontally the tiers of an application based on a small number of low-level metrics (e.g., CPU usage and memory usage). For a simple web application, such elasticity controllers are capable of only scaling the application server tier and the distributed database backend by adding/removing virtual instances, when predefined thresholds are violated. Moreover, for large-scale applications, in order to reduce costs and match the current demand, one requires from elasticity controllers to apply various complex adaptation mechanisms, which we refer to as *elasticity control plans*. These mechanisms are required to carefully assess the actual application logic with respect to its internal dependencies and (implicit) requirements towards the cloud provider APIs, including communication, consistency management and scheduling.

To facilitate complex adaptation mechanisms, an elastic compute resource provisioner is required to not only limit its decisions on low-level monitoring information. Instead, it is required to assess heterogeneous types of monitoring information of different granularity, from low-level system metrics (e.g., CPU, memory, network utilization) to high-level application specific metrics (e.g., latency, throughput, availability), which are collected across multiple levels (physical, virtualization, application level) in a Cloud environment at different time intervals, as Trihinas et al. (2014) do. To accommodate these limitations, our work incorporates JCatascopia (presented in detail in Trihinas et al. (2014)), a fully-automated, multi-layer, interoperable cloud monitoring system which provides access to monitoring information through its REST API.

To enforce complex adaptation mechanisms, decisions originating from an elasticity controller must also be aware of what are the offerings and limitations of the underlying IaaS provider. Specifically, the controller must consider: (i) what are the resizing actions permitted per resource and (ii) the quotas for each user/tenant. Knowing the elasticity capabilities of each IaaS resource is of extreme importance when determining which elasticity mechanism should be enforced. For example, let us consider two IaaS providers (Provider A and Provider B) where only the first provider offers users the capability of vertically scaling virtual instances by allocating more memory, while both offer horizontally scaling capabilities. If we consider a three-tier web application deployed on Provider B, the control mechanism can only scale horizontally the Application Server Tier when memory utilization increases. For Provider A though, the decision-making mechanism can take advantage of Provider A's extra capabilities and decide upon either scaling horizontally the Application Server Tier or, enlarging the allocated memory of existing instances. This approach takes cost into consideration since resizing existing VM(s) may be cheaper than constantly initializing small virtual instances. Additionally, it is important for elasticity controllers to also consider the per tenant quotas such as: (i) the total capacity of resources that a tenant can allocate; and (ii) the multiplicity of resources that can be concurrently allocated at any given time. In continuation of the previous example, if the permitted number of allocated VMs per tenant is low, our application deployed on Provider B will face quota problems when scaling to satisfy very high demands,

[12] https://www.paraleap.com
[13] http://www.rightscale.com

whereas for Provider A, an intelligent elasticity controller can scale the application both vertically and horizontally to satisfy an even higher demand. To accommodate these limitations, our work constructs an information management tool (described in detail in Trihinas et al. (2013)) which provides access to IaaS specific information.

The inherent dynamicity in the run-time topology of elastic cloud applications raises several issues in run-time control. As elastic applications scale out/in due to elasticity requirements, their underlying virtual infrastructure is subject to run-time changes due to additional/removal of virtual resources (e.g., virtual machines). Thus, cloud application monitoring must avoid associating monitoring information only with virtual resources, as these resources are volatile, and are not present for the whole lifetime of the application. For example, when the application usage is low, one application component could use only one virtual machine, but during peak times would allocate more resources, and deallocate them when load decreases. The other extreme of monitoring just the application level metrics (e.g, response time) is also insufficient, as such high level metrics do not give any indicator on the performance of the underlying virtual infrastructure. Thus, systems for monitoring elastic cloud applications must follow a multi-level monitoring approach. Both virtual infrastructure and application level monitoring data must be collected, and structured according to application's logical structure, as done by Moldovan et al. (2013). Evaluating the cost of an application running in a cloud environment is challenging due to the diversity and heterogeneity of pricing schemes employed by various cloud providers (e.g., Provider A may charge per I/O operation, while Provider B might charge only per storage size). This heterogeneity generates a gap between the monitoring metrics collected by a monitoring system and the metrics targeted by cloud billing schemes. Moreover, evaluating the cost of the application requires information about particular cloud pricing schemes, information that cannot be monitored directly by a cloud monitoring system. To address these issues, our work provides MELA (Moldovan et al. (2013)), which uses monitoring information collected from cloud monitoring tools and the cloud application structure, to provide a cross-layered, multi-level view over the performance and cost of elastic cloud applications.

### 2.3 Data Resources Elasticity Control

Data-related elasticity controls of cloud application usually entail, at system level, removal/addition of data nodes in clusters of data. Elastically scaling data resources in the cloud requires a data-aware approach in order to obtain the full benefit of extra added resources. The first and most important thing that needs to be addressed during resource adjustment is uneven data distributions: when data nodes join or leave from a data-storage component, they create imbalances in the initial data distribution. Even when resources do not change, unpredictable data access patterns often create unbalanced distributions that degrade performance. In that cases, load balancing approaches that redistribute data between nodes are necessary.

Consistent hashing techniques described by Karger et al. (1997) are a common and effective solution for data control. The majority of modern NoSQL stores (e.g., Lakshman et al. (2010), DeCandia et al. (2007)) make use of such techniques to equally allocate data and incoming requests to the available nodes. Although hashing initially solves the data to machines allocation problem, there are many situations in which this proves suboptimal. Hashing destroys locality and thus, it cannot be employed in situations where semantically close items need to be stored in an order-preserving way. When an order-preserving partitioner is desired, different load balancing schemes need to be devised in order to support range queries. Range queries are present in many popular applications. Therefore, algorithms and systems which handle this case are of great importance. In the literature, there are many load balancing algorithms (e.g., Bharambe et al. (2004), Aspnes et al. (2004), Ganesan et al. (2004), Karger et al (2004), Konstantinou et al. (2011)) which support range queries.

The need to support range queries highlights another problem which belongs to the load balancing family. Although data placement can be balanced, there may be imbalances in the data request load. Ananthanarayanan et al. (2011) show that in a highly skewed data access distribution, where a

small portion of popular data may get the majority of the applied load, the system performance may degrade even in over provisioned infrastructures.

DBalancer proposed by Konstantinou et al. (2013) is a generic and automated system, offering load balancing in NoSQL datastores, which we choose to use and extend. DBalancer is a generic distributed module that performs fast and cost-efficient load balancing on top of any distributed NoSQL datastore. The two main features of DBalancer are the datastore and algorithm abstraction. DBalancer is completely independent of the underlying NoSQL datastore.

### 2.4 Complex Service Elasticity Control

Elasticity has been outlined to be a complex property (see Dustdar et al. (2011)) composed of quality elasticity, cost elasticity and resources elasticity. This section shows research on elasticity control which also considers cost and quality in the control process. Schatzberg et al. (2012) raise issues that appear in cloud elasticity control and outline that in cloud computing elasticity is an important area of research, which will facilitate the development of applications that would fully benefit from the advantages of cloud computing and from on-demand resources allocation. The different perspectives of cloud applications performance/cost/quality measurement are outlined by Li et al. (2012) who propose a list of categories of metrics which are used for evaluating cloud applications. Their retrieved cloud application evaluation metrics are scattered over three aspects of cloud applications: economics having as subdimensions cost and elasticity evaluation metrics, performance with subdimensions communication, computation, memory, storage evaluation metrics and security evaluation metrics. The abstract metrics are associated to measurable metrics for easier grasp of reality and for being able to actually compute the abstract metrics.

Truong et al. (2010) estimate the cost of application hosting on the cloud considering different sub-costs which may interfere during the lifetime of the application. Villegas et al. [Villegas 2012] propose a framework for conducting empirical research in different IaaS clouds, comparing different allocation and provisioning policies. The authors emphasize the importance of understanding the performance and cost associated with different provisioning or allocation policies, for being able to properly manage their application's workloads.

Gonzalez et al. (2012) propose cloud infrastructure-level virtual machine management for increasing the VM availability. The authors also provide a study on how different properties of the cloud infrastructure affect the VM availability. Chaisiri et al. (2012) focus on the complexity of selecting cloud applications under different provisioning plans, such as reservation and on-demand, defining an optimal cloud resource provisioning algorithm that can provision resources in multiple provisioning stages. Using deterministic equivalent formulation, sample-average approximation, and Benders decomposition, their proposed solution minimizes the total cost of resource provisioning in cloud computing environments.

## 3 Motivating Scenarios

We focus on user scenarios which we encountered in CELAR, namely: (i) the needs of a cancer research application, and (ii) the requirements of a gaming application. For these cases, the applications are designed such that they facilitate as many elasticity capabilities, in order to facilitate better elasticity control.

### 3.1 Cancer Research Application

The first application, SCAN, described in detail in Xing et al. (2014), is a cancer research application designed by the Cancer Research UK Manchester Institute, which analyzes large-scale population genome data for helping doctors to determine personalized treatments. The SCAN pipeline consists of four types of data processes: i) Genome data process; ii) Proteome data process; iii) Cell Image data process; iv) Integrative network analysis. It employs a set of biological application tools for

those various data processes, such as Burrows-Wheeler Aligner (BWA) for gene alignment, Genome Analysis Toolkit (GATK) for e.g., gene variations detection, The Global Proteome Machine for proteomic data analyses, MaxQuant, CellProfiler for cell image analyses, or Cytoscape for data integration.

There are two major challenges regarding cloud-based deployments of such research pipelines. First, different stages of the pipeline may require substantially different levels and types of resources. For example, mapping of deep sequencing data to genome annotation via a relational database such as ENSEMBL[14] relies on the ability to perform frequent joins across multiple tables containing millions of rows, while computation of downstream statistics is often dependent on repeated numerical calculations over permuted data. Second, a specific bio-component within a SCAN stage may have different resource needs due to the size and complexity of the data for different SCAN runs. For example, SCAN mutation detection process will take different time for various type of genome data, e.g., 4 CPU/hours for Whole Exome Sequencing data (WES) or 10 CPU/hours for Whole Genome Sequencing (WGS) data.

To address the challenges described above, SCAN has been designed as an elasticity-ready bio-computing application so that it can be intelligently orchestrated for adjusting resources to the various situations which can be encountered. For instance, considering the fact that cancer diagnosis and treatment is "time-sensitive", sometimes doctors may need the result of SCAN for a patient in a particular period. Therefore SCAN should be executed according to user specified priories. It is thus important to be able to decide on the adequate amount and type of resources, depending on various metrics, e.g., available money, desired time, or desired accuracy. Moreover, SCAN is comprised of a wide range of bio-applications and may require a large amount of heterogeneous computing resources. The SCAN users may need to query information about execution of bio-applications within different cloud infrastructures in order to assist SCAN users to define, for example, policy of the execution.

Based on the application description above, our control will ensure the following: (i) deciding the appropriate size of resources, (ii) ensuring predefined levels of service quality, (iii) ensuring that the SCAN pipeline runs within desired costs, and (iv) deciding the concurrency level and appropriate time periods of different stages. Moreover, since SCAN is comprised of pipes (i.e., components grouped together), the control needs to facilitate the fulfillment of multi-level requirements (e.g., a specific pipe needs to finish executing, with certain quality, before another pipe), and controlling high level metrics (e.g., overall application cost, quality indicators over specific pipes).

The SCAN application needs to benefit from the on-demand storage capabilities offered by the IaaS providers (~okeanos or Flexiant), as well as application-specific control mechanisms, this way offering personalized treatments within time and cost constraints. SCAN performance and cost can be customized according to real-time elasticity metrics, thus resulting in a personalized control of the application. For understanding the relation between requirements regarding SCAN execution, and the performance/cost obtained, CELAR's user interface component can be used to browse historical execution data. Moreover, cost and functionalities offered by different IaaS providers can be compared and elasticity control actions taken during the execution of the SCAN pipeline can be analyzed.

## 3.2 Gaming Application

Playgen's Data Play is a gaming application, described in detail in Cox et al. (2014). The DataPlay application is designed with elasticity in mind. The main elasticity capabilities designed and embedded in DataPlay are horizontal and vertical scaling of game components, such as the Game Server, the Data Processing component, and the Data Access layer. For enforcing such capabilities, implemented elasticity actions target both the virtual infrastructure (e.g., adding/removing virtual machines), and the application level (e.g., reconfiguring load balancers or data storage).

[14] http://www.ensembl.org

Starting from industry known guidelines, Data Play requirements are `response time<1.5 seconds`, `I/O Performance >= 100 MBps`, and cost as small as possible. CELAR will analyze the behavior of all Data Play components' instances, and, leveraging on the embedded elasticity capabilities of the Data Play, take appropriate actions to ensure the performance and reduce cost of running the Data Play in cloud. Starting from the game requirements, our controller will extract system-level requirements (e.g., CPU usage, memory usage, disk I/O performance) and application level requirements for the individual game components. Having a complete view over system and application level requirements, CELAR will monitor and enforce the supplied requirements using the game's elasticity capabilities. DataPlay is centered on users exploring data (Volatile and Persistent data), thus introducing data-related elasticity concerns. Persistent data is static, or changed with a very low frequency (a couple times a year), but it is frequently accessed, highlighting the need for data consistency. Volatile data is created for each DataPlay user, and, for performance, holds temporary data .If a client is manipulating a dataset, then the application treats that as a different table for speed reasons, but if the client's session expires then that table is destroyed. Therefore, for this application we have a continuous increase/decrease in data depending on the client number, on the size of the datasets they are interested in and on the time they use that data for.

Large volumes of data can also come in at any time, for instance from tweets and RSS feed updates. However, the volatile data is copied for individual users depending on their interest and gameplay. For this kind of data usage, data freshness is an important factor, as one needs to have as fresh as possible trending-related data, especially if the data s/he uses has been cached for performance reasons.

# 4 Elasticity Requirements Specification

## 4.1 SYBL Overview

For describing what types of elasticity controls could be required, considering the complexity of CELAR user's elasticity requirements, we examine various types of elasticity requirements from different stakeholders, at different granularities, presented in detail in Copil et al. (2013a). Elasticity requirements are abstract or high level demands formulated by application stakeholders (e.g., application provider, application developer) which affect the application pathway in the elasticity space presented in detail in Moldovan et al. (2013). Although current state of the art (e.g., Amazon AutoScale) facilitates description of low-level, infrastructure-related requirements, the application stakeholder should to be able to specify requirements concerning more abstract metrics (e.g., the cost per application user that the stakeholder needs to pay per hour).

SYBL is a language for elasticity requirements specification, having three types of constructs at its core, enabling the specification of elasticity requirements: (i) MONITORING enables the designation of different metrics or formulas of metrics which should be monitored; (ii) CONSTRAINT specifies the desired state of the application, while (iii) STRATEGY specifies the desired behavior of the application or of different application parts. More information regarding the SYBL language is available in Copil et al. (2013a). SYBL allows the specification of elasticity requirements at three levels: application unit level for component related elasticity requirements, service topology elasticity requirements related to groups of components and application level for application related elasticity requirements. At service unit level, the SYBL user (e.g., service provider, or service developer) can specify requirements for the component which is of interest (e.g., for a business unit we can have `STRATEGY CASE NumberOfClients<100 AND small(ResponseTime) : minimize (cost)`). For service topology level, SYBL user can specify higher level goals which target higher level metrics (e.g., `CONSTRAINT DataAccessSkewness < 90%`), while at cloud service level s/he can specify complex requirements for the entire cloud service, targeting the overall behavior of the cloud service (e.g., `STRATEGY maximize (cost/clientNb)` )

The SYBL language is not tied to any specific implementation language (e.g., SYBL elasticity requirements can be seen as Java annotations, C# annotations, or Python decorators). Moreover, the

SYBL elasticity requirements can be injected into any cloud application description language (e.g., TOSCA standard proposed by OASIS (2013)) or can be specified separately through XML description. The current language interpretation mechanism is implemented in Java, and supports TOSCA-injected, XML-based, or Java annotation-based elasticity requirements specification.

Figure 1 shows an excerpt from a TOSCA policy based specification, describing the constraint that the cost for the `PilotCloudService` should be below 100$. The SYBL elasticity requirements can be easily integrated within TOSCA policies, and interpreted by the CELAR Decision Module. rSYBL understands two types of SYBL requirements injected into TOSCA: simple, as SYBL text (Figure 1) .

```
<tosca:ServiceTemplate name=”PilotCloudService”>
      <tosca:Policy name=”Co1” policyType=”SYBLConstraint”>
            Co1:CONSTRAINT Cost &lt 100\$
      </tosca:Policy>
      …
</tosca:ServiceTemplate>
```

Figure 1: SYBL elasticity requirement in TOSCA

We therefore facilitate the user to specify SYBL elasticity requirements with the help of CELAR user interface, as part of the process of cloud application description, as presented below.

## 4.2 SYBL Elasticity Requirements Specification with c-Eclipse

To simplify the task of the developer, we integrate elasticity requirement specification into c-Eclipse, which facilitates the user to describe his/her application, requirements for it and monitor application evolution at runtime (c-Eclipse is described in detail in Sofokleous et al. (2014)). c-Eclipse enables the specification of SYBL elasticity requirements and their injection into TOSCA XML application descriptions. The TOSCA language does not directly specify how to define elasticity requirements for Cloud applications. The way c-Eclipse achieves elasticity specification in TOSCA is by making use of TOSCA’s Policy element. TOSCA defines "Policies" as the means by which we can express non-functional behavior or quality-of-services for an application (see OASIS TOSCA Specification (2013)). Thus, we make use of the two types of elasticity requirements defined in the SYBL XML schema (Constraint and Strategy), and inject them into TOSCA as Policy elements of the corresponding types.

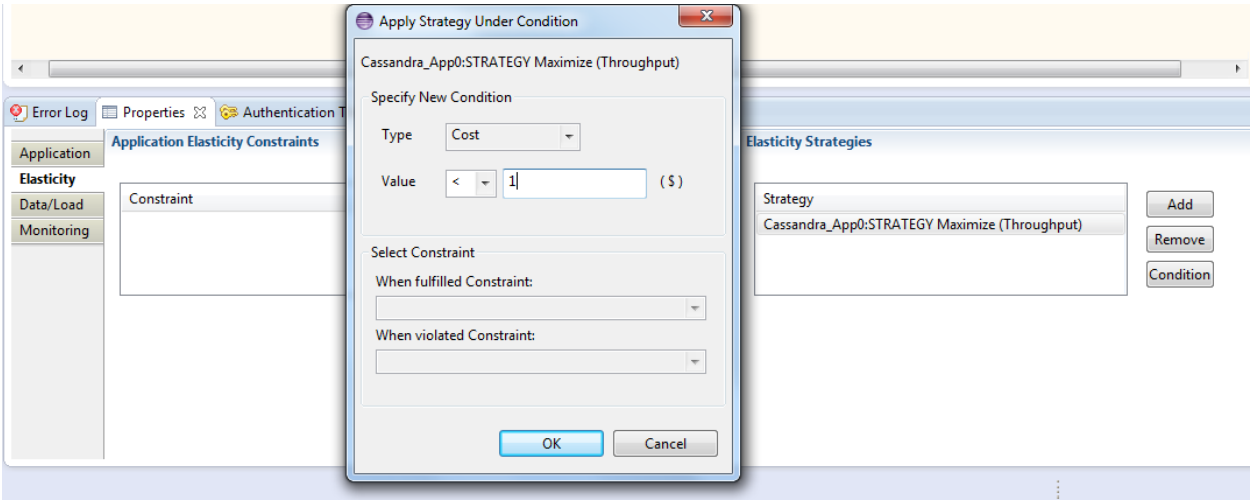


Figure 2: Elasticity Requirements Specification with c-Eclipse

Figure 2 presents the *Properties View* of c-Eclipse, specifically the *Elasticity Tab*, through which users can define the elasticity requirements of their application in an intuitive, user-friendly manner.

Users can use simple low level metrics offered by the platform, such as CPU Usage, to complex user-defined metrics (e.g., cost/client/h), and specify the desired requirements for these metrics. They can also define more complex metrics by combining simple metrics with mathematical operators to design new metrics. Both types of metrics can be used to formulate the *Constraints* and *Strategies* for an application, using the SYBL language defined in Copil et al. (2013).

The user-specified elasticity requirements are automatically translated into XML SYBL requirements which in turn are injected into the XML TOSCA description of the application. The code snippet below reflects the elasticity *Strategy* specified through c-Eclipse, shown in Figure 3. In this strategy, the user wants to `maximize` throughput for an application component (right side of Figure 2), when the cost is less than 1 $/h (in the center of Figure 2, the "Apply Strategy under Condition" window).

```
<tosca:PolicyTemplate type="pol:ElasticityStrategy" id="Throughput_Strategy">
  <tosca:Properties>
            <sybl:StrategyProperties>
                  <Condition>
                        <BinaryRestrictionsConjunction Type="LessThan">
                              <LeftHandSide>
                                    <Metric>cost</Metric>
                              </LeftHandSide>
                              <RightHandSide>
                                    <Number Metric="$/h" >1</Number>
                              </RightHandSide>
                        </BinaryRestrictionsConjunction>
                  </Condition>
                  <ToEnforce ActionName="Maximize" Parameter="Throughput"/>
            </sybl:StrategyProperties>
  </tosca:Properties>
</tosca:PolicyTemplate>
```

Figure 3: Elasticity strategy expressed through policy template by c-Eclipse

Elasticity requirements can be linked to simple application components, composite components or the entire application, depending on which graphical element from the application description is selected when the user specifies the requirements.

## 5 Multi-level and Multi-dimensional Elasticity Control

In this section we focus on the mechanisms used in CELAR for multi-level control of the cloud application, for fulfilling user's elasticity requirements.

### 5.1 CELAR Elasticity Control Overview

CELAR proposes application elasticity management, from deployment to runtime control, in an automated fashion. CELAR targets the control of applications deployed in a single cloud. For each cloud where the user has an account and at least one application deployed, consuming cloud provider resources, CELAR deploys an orchestration instance (CELAR Orchestration VM in Figure 4), hosting all CELAR components necessary for deploying, monitoring, analyzing and controlling application's elasticity.

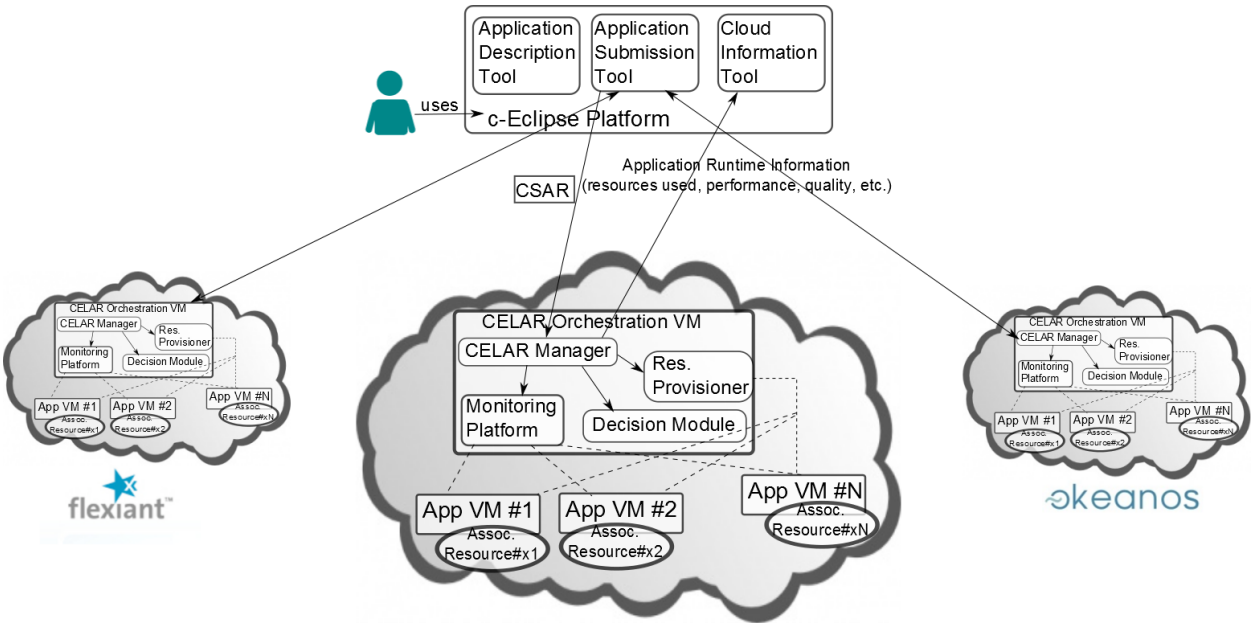


Figure 4: Communication among CELAR Components

Figure 4 shows a snapshot of a CELAR-based deployment, containing all CELAR components, the application which is being controlled and communication among them, with examples for ~okeanos and Flexiant cloud providers. CELAR user first describes his/her application through c-Eclipse, in the Application Description Tool. This description includes application topology, elasticity requirements at the different levels of the cloud application, and specific artifacts of the application (e.g., web services, or configuration scripts). All information from application description step is described using TOSCA standard defined by OASIS Technical Committee (2013), the description together with the artifacts being packed into a Cloud Application Archive (CSAR) and using the c-Eclipse Application Submission Tool, the cloud provider is selected, the authentication information set and the cloud application deployment call is sent to the CELAR Manager from the CELAR Orchestration VM on the selected cloud. For the same user, we can have a single orchestration instance per cloud provider, which controls and monitors all the user's applications. The CELAR Manager has the mission of coordinating the communication among CELAR modules: (i) it receives the cloud application archive from c-Eclipse, forwards it to decision module in case the user needs suggestions regarding application configurations, and then sends it to Resource Provisioner for allocating the necessary resources, (ii) whenever the Decision Module, which controls the cloud application, decides that control processes need to be enforced for fulfilling user-specified requirements, the CELAR Manager coordinates the enforcement of the generated action plan with the Resource Provisioner.

The process of controlling the cloud application is depicted in Figure 5, from c-Eclipse application description, to deployment configuration and elasticity control. Whenever the CELAR user would like a more complex application configuration describing the resources used or with the software artifacts to be deployed, it can ask, through c-Eclipse, for a smart deployment strategy, containing complete information for deployment (e.g., resource configuration, or missing artifacts). After the deployment, the application is monitored and analyzed continuously, and elasticity control is enforced for fulfilling elasticity requirements.

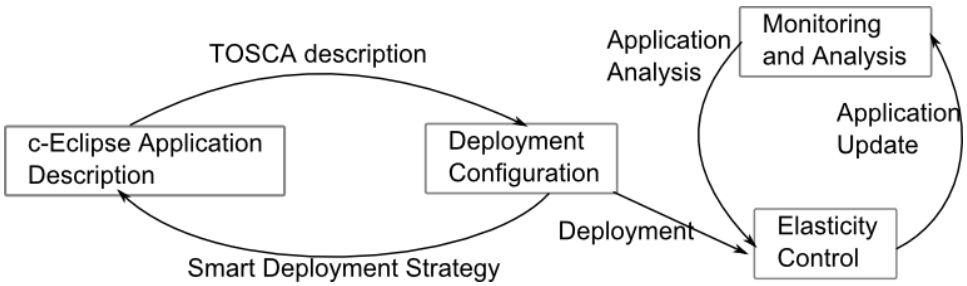


Figure 5: Elasticity control process – from description to control

For analyzing and controlling the cloud service, we model various application-related information into a dependency graph, based on the model published in Copil et al. (2013b), depicted in Figure 6. At runtime, the information is represented as a dependency graph, with each concept instance (e.g., Composite Component) from the model being a node, while the relationships (e.g., `hasElasticityCapabilities` in Figure 6) are edges connecting them.

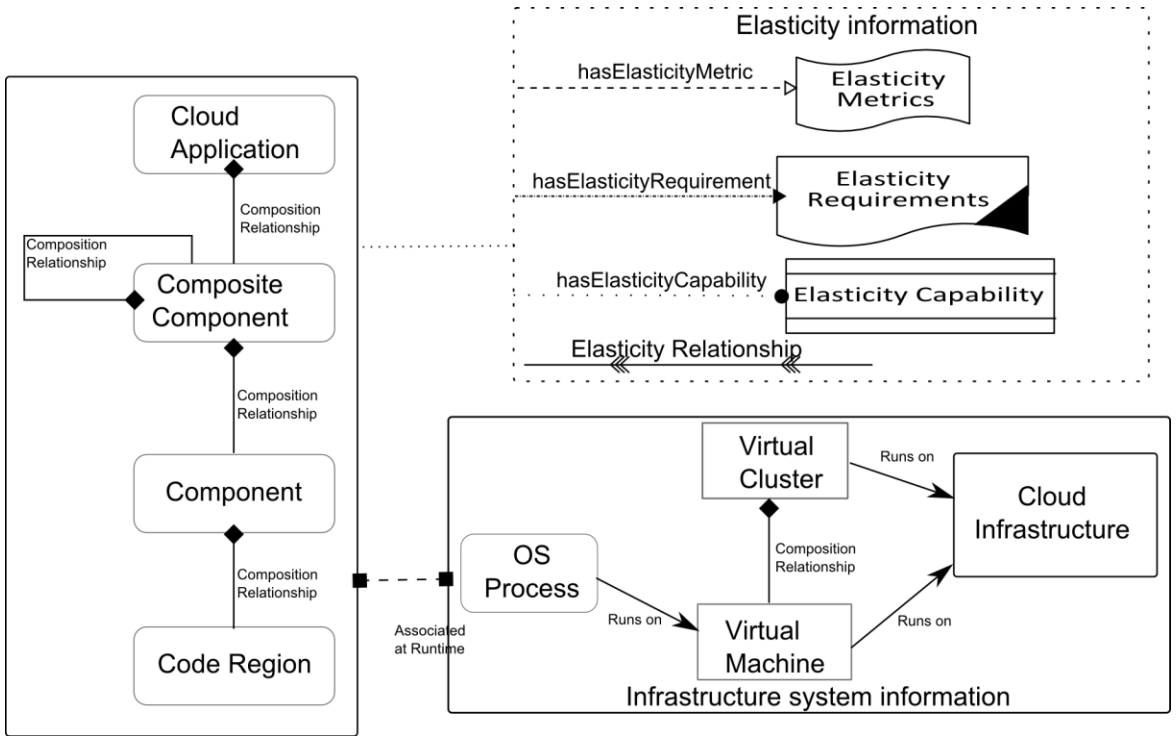


Figure 6: Cloud Application Model – Copil et al. (2013b)

The structural information captures *components* (e.g., NoSQL database node, or a tool in the SCAN pipeline) and groups of semantically connected components into *composite components* (e.g., business layer, or a SCAN stage). To each of these, during runtime, are associated resources like processes, virtual machines (equivalent to servers on Flexiant cloud), virtual clusters (equivalent to virtual data centers in Flexiant).

## 5.2 Cloud Application Deployment Configuration

For configuring cloud application deployment, we have developed a service, SALSA (Le et al. (2014)), generating smart deployment configurations, by analyzing application structure and elasticity requirements. This service takes a high level application description, application profiling information, and available cloud description information and generates a deployment configuration.

Figure 7 depicts the simple flow of generating a deployment plan from the high level application description. The trigger for a smart deployment and necessary information is sent by c-Eclipse when a new application deployment or a re-deployment of running components is needed.

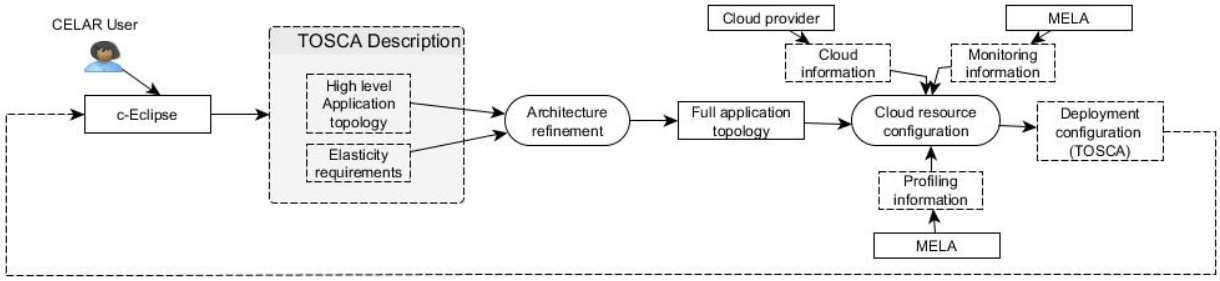


Figure 7: Flow of Deployment Configuration

The input for SALSA Service is a TOSCA description from c-Eclipse which consists of a high level application description, containing only structural and application-specific information (e.g., application artifacts), without a description of resources needed for deployment. The input is processed via `Architecture refinement` and `Cloud resource configuration` modules, and produces a deployment configuration and sends it to c-Eclipse.

The high level application description defines components in an abstract way. For instance, cloud resources can be described using specific categories such as compute, storage, network; software dependencies can be represented by types such as web container, database, API libraries, which are all to be found in c-Eclipse as types of software requirements. For each deployment configuration step (see Figure 5), we interpret SYBL requirements, in order to detect deployment preferences (e.g., optimize cost, maximize resource usage, or maximize latency), and guide our deployment configuration with identified preferences. The architecture refinement step in Figure 7 aims to enrich the application topology with artifacts/software using CELAR repository of existing artifacts (e.g., Tomcat web server). The output of this step is a full application topology with all the needed artifacts. Moreover, in the cloud resource configuration step of Figure 7, we associate the required resources to each components/artifacts previously selected, using cloud provider and application profiling information. The resulted configuration is expressed as a TOSCA description, and returned to c-Eclipse for being analyzed and modified as needed by the CELAR user.

### 5.3 Cloud Application Monitoring and Analysis

For application monitoring and analysis we use MELA, described in detail in Moldovan et al. (2013), which analyzes the elasticity of cloud applications, focusing on the three elasticity dimensions: cost, quality and performance. MELA provides elasticity analysis capabilities on the aggregated monitoring data coming from the Cloud Information and Performance Monitor, determining the elasticity boundaries, space and pathway of cloud application. This information is used by rSYBL (see Section 5.4) in controlling the elasticity of such applications.

For analyzing and controlling the cloud application, *elasticity space boundaries* are determined for all application components, composite components, and whole application, and are equal to the maximum and minimum encountered metric values when the elasticity requirements were respected. Thus, starting from supplied user requirements, MELA determines and continuously updates requirements for the rest of the application components, requirements then enforced by rSYBL, as described in Section 5.4.

For elasticity control of cloud applications we also use the *elasticity pathway* function, which gives an indicator on the historical behavior of the cloud application and correlations between the

application's metrics. The elasticity pathway information provides a base for refining user-defined requirements, and validating the Decision Module's control strategy. In the current prototype of the MELA we adapt as elasticity pathway function, an unsupervised behavior learning technique using self-organizing maps (SOMs) proposed by Dean et al. (2012), and classify monitoring snapshots by encountering rate in DOMINANT, NON-DOMINANT, and RARE. Such a pathway is important for understanding if the regular behavior of the application fulfills user-defined elasticity requirements.

### 5.4 Cloud Application Elasticity Control

Considering the model of the application described through the runtime dependency graph presented in the previous subsection, we use rSYBL elasticity control, described in detail in Copil et al. (2013b), to enable multiple levels elasticity control of the described application, based on the flow shown in Figure 8. The elasticity requirements are evaluated and conflicts which may appear among them are resolved. After that, an action plan is generated, consisting of actions which would enable the fulfillment of specified elasticity requirements.

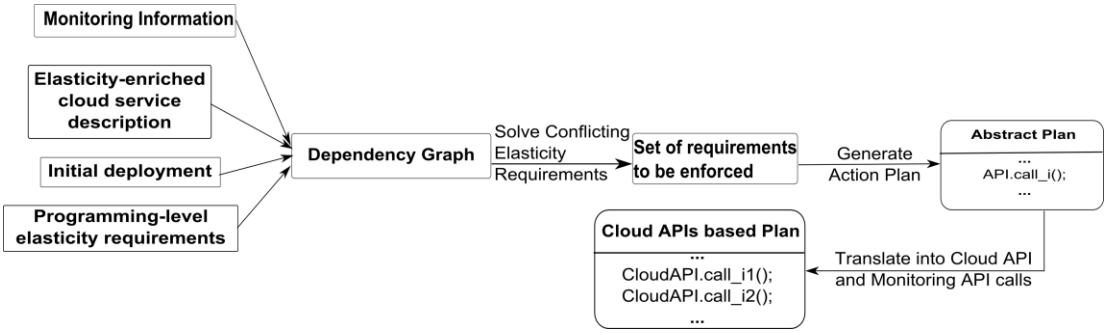


Figure 8: Flow of Elasticity Control

Let us consider a simple example shown in Figure 9 of controlling the entire application, e.g., by the system designer. The described elasticity requirements, `Co1`, `Co2`, and `Co3` are not conflicting, and actions are searched for fulfilling these requirements. Possible actions are, for instance, for the case the running time is higher than 10 hours and the cost is still in acceptable limits, to scale-out for the computation composite component, increasing the processing speed. An example of an action plan, shown in Figure 9 could be:

$$ActionPlan1 = [\,[\,increaseReplication],[\,scaleOut, setThreadPool = 100]].$$

This action plan would address performance issues for the second elasticity requirement `Co2`, and availability issues for the third elasticity requirement `Co3`. Each of the generated abstract actions are mapped into complex API calls. For instance, `increaseReplication` action would consist of calls for adding and configuring a new database node and configuring the cluster for higher replication, while the `scaleOut` action would be the addition of a new virtual machine, deployment of the `ComputationEnd` component on the new machine, and necessary calls for the new instance of the component to join the computation topology cluster.

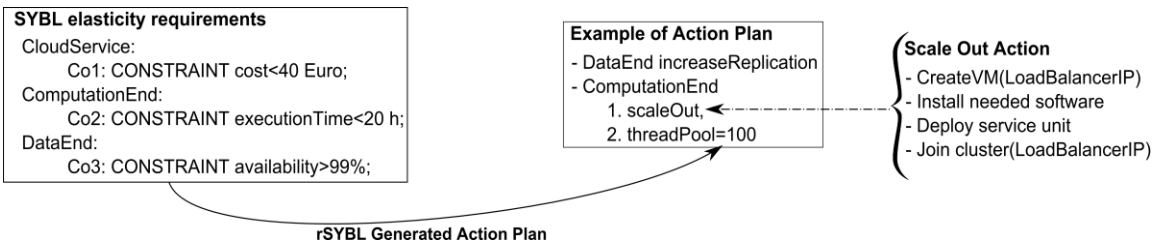


Figure 9: Action Plan Example

## 6 Experiments

The two applications described in Section 3 are currently being developed. Therefore we choose to showcase CELAR control approach on a Machine-to-Machine (M2M) DaaS Service. The M2M DaaS Service is quite complex, containing two composite components, one application server-based and one which is a NoSQL. This is similar to the gaming application presented in Section 3.2, which also has requirements regarding application-level metrics like response time and latency. Moreover, the M2M DaaS composite components are similar to pipes in SCAN application presented in Section 3.1, in which the SCAN developer wants to introduce requirements at pipe-level as well as at component level, thus having multi-level elasticity control for the SCAN application.

Considering a CELAR user that wants to deploy this M2M DaaS in the cloud and expects an elastic application behavior, the CELAR user needs to describe two types of information: structural information regarding application artifacts, and elasticity requirements at the different application level. The M2M DaaS, shown in Figure 10, is comprised of two composite components, an `Event Processing Composite Component` and a `Data End Composite Component`. Each composite component consists of two components, one with a processing goal, and the other acting as the composite component balancer/controller. To stress this application we generate random sensor event information which is processed by the `Event Processing Composite Component`, and stored/retrieved from the `Data End Composite Component`.

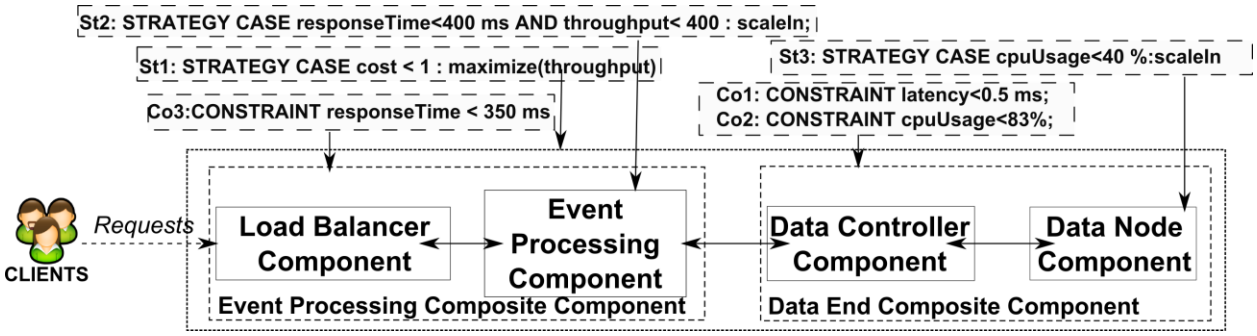


Figure 10: Application Used for Evaluation

Moreover, the CELAR user is interested in specifying a number of elasticity requirements, both at component, composite component, and at whole application level. The requirement specified at whole application level (`St1`) specifies as a strategy to increase as much as possible the throughput, but under specific cost condition. In the upper part of Figure 10 shows the various elasticity requirements which we associate to the different levels of M2M application. For having the application elasticity controlled by CELAR, the M2M application as well as these elasticity requirements need to be described with c-Eclipse, as we describe in Section 6.1. After describing the application and pressing the deploy button, the application is controlled following the approach presented in Section 5, control results being presented in Section 6.2.

### *6.1 Application Description with c-Eclipse*

The c-Eclipse framework provides an intuitive, user-friendly interface through which users can describe their applications for deployment over cloud platforms.

The c-Eclipse user interface is depicted in Figure 11. At the left-hand side, the CELAR user can see the CELAR Project View where all the files related to an application description are organized in a hierarchy. The Palette, shown at the right-hand side, includes most of the elements required for creating application descriptions, categorized under different Palette sections. By simply dragging and dropping pictorial elements from the Palette onto the center Canvas, users can create a graphical representation of an application. Additional information can be provided for each element via the Properties View (see in

Figure 11). Application descriptions are translated on the fly into XML, according to the open TOSCA specification for cloud applications.

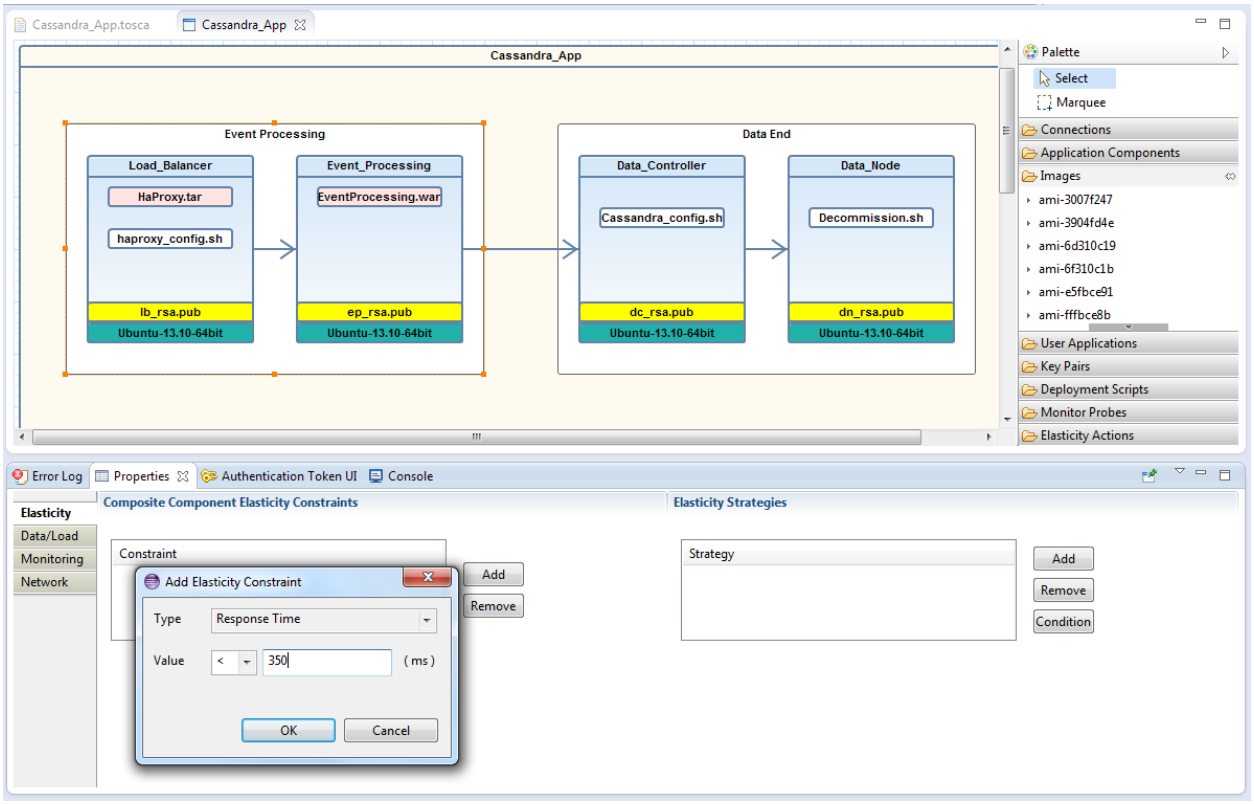


Figure 11: Application Used for Evaluation Described in c-Eclipse

The first step in describing an application is to define the application's structure/topology, following the abstract application composition-based model described in Section 5.2.1. To do so, the user must use components and composite components from the Palette's Components section and then create the relationships between these components by using relationships from the Palette's Connections section. Once the application structure is defined, the user can define the application's properties such as the VM images (shown in the Palettes Images section) and other executables to be installed on the defined application components (Palette's Deployment Scripts section). Moreover, s/he can describe the important monitoring metrics at each application level (Palette's Monitoring Probes section), together with the elasticity actions to be applied when scaling the application's deployment (Palette's Elasticity Actions section), and the time when these actions should be applied. Specifically:

- At Component Level, the user can define the following:
  - *VM Image* that will be used by the underlying platform when materializing instances of the component (green color box).
  - *Key Pairs* generated by the user that will be used by the underlying platform when deploying the component. Thus, a user can make use of the key pair later to access the deployed component (yellow color box).
  - *User Applications*, such as .jar and .war files, that will be used by the underlying platform when materializing instances of the component (orange color box).
- At Application, Component and Composite Component Level, the user can define the following:
  - *Deployment Scripts* that will be executed by the underlying platform when initializing instances of the component (pink color box).

- *Monitoring Probes* that will be used by the Monitoring System to capture and return the corresponding metrics to the user. Furthermore, the metrics referred by the probes can be used in the specification of elasticity policies.
- *Elasticity Actions* that can be applied to the components. Elasticity actions can also be used in the specification of elasticity policies.

Figure 11 shows the c-Eclipse application description, following the structure depicted in Figure 10. For achieving this c-Eclipse application description, the user will first drag a composite component from the Palette's Components section onto the Canvas to create the `Event Processing` component. Then, he will drag two simple components and drop them inside the composite component one for the `Load Balancer` component and one for the `Event Processing` component. In a similar way the user can create the `Data End` composite component with the two simple components inside it for the `Data Controller` and the `Data Node`.

Apart from the structure of the application, the user can specify other application properties, such as its elasticity policies. For example, by using the Properties View of c-Eclipse (bottom of Figure 11) the user can define the constraint of keeping the `Response Time` for the `Event Processing Composite Component` below 350 ms.

### *6.2 Controlling the Application with CELAR Decision Module*

After describing the application as above, with the help of c-Eclipse, the CELAR user chooses the cloud provider to be used, and specifies his/her credentials, and with a simple press of a button, the application, together with all the necessary CELAR tools are deployed in the cloud. After this, the CELAR user can observe the evolution of application metrics, which is being controlled with the approach presented in Section 5.

Figure 12 depicts a view from the MELA user interface, which is integrated into c-Eclipse for CELAR users to be able to follow cloud application behavior during runtime. The CELAR user can observe various metrics, at the different cloud application levels,

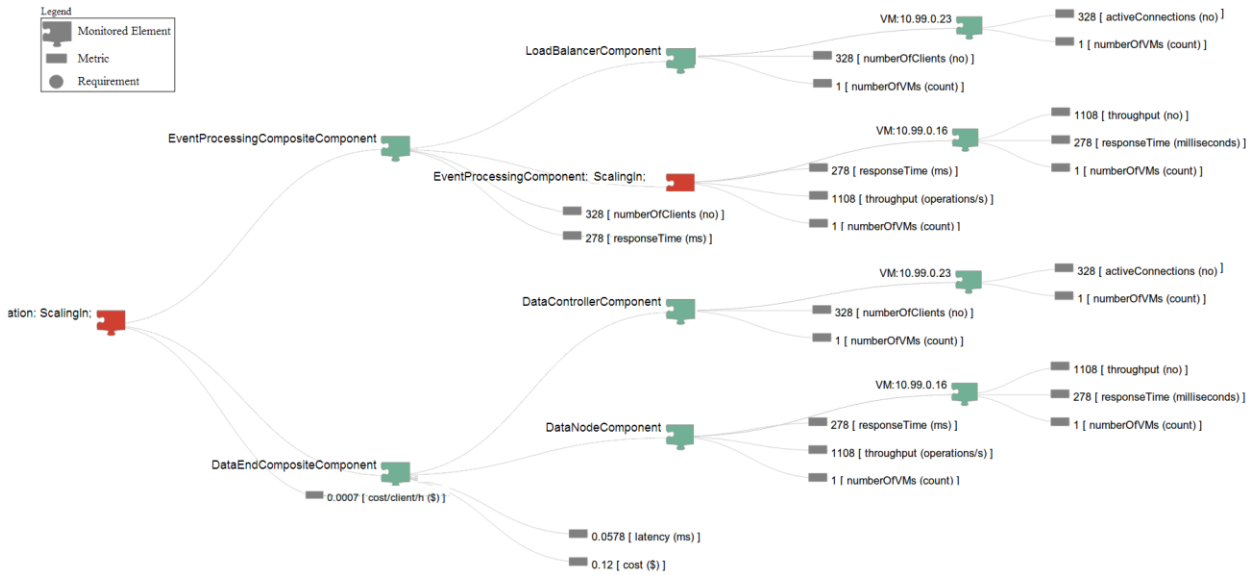


Figure 12: Cloud application elasticity control enforcement

By clicking on different components or complex components, the user is lead to a new view, in which s/he can observe various charts showing metrics evolution in time, and statistical data. Due to the scaling actions enforced by the CELAR Decision Module, the response time is able to stay within the

required boundaries, as shown in the left side of Figure 11, at a relatively stable value without increasing more than acceptable for a too high period. The user can observe that due to CELAR control, there is a correlation between the number of VMs and the number of clients, as depicted in Figure 13, thus showing that the Decision Module is able to adapt the application in order to accommodate a varying demand. This is strengthened by the elasticity pathway depicted in Figure 15. From the pathway's "x" axis, the situation encounter rate, i.e., the percentage of time that situation was encountered, one can see that in 90% of the situations, the response time was maintained within acceptable values.

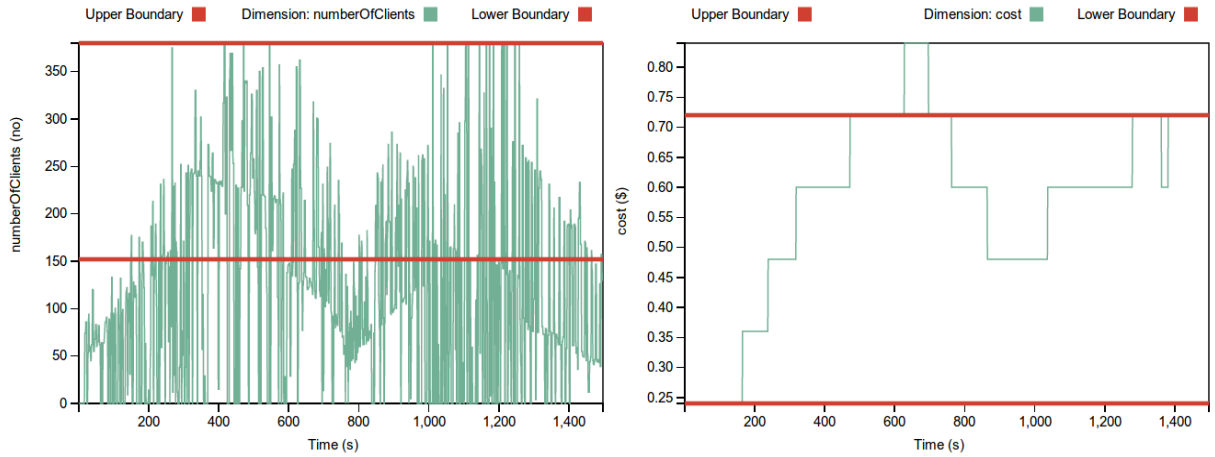


Figure 13: Elasticity Control shown at `Event Processing` Composite Component

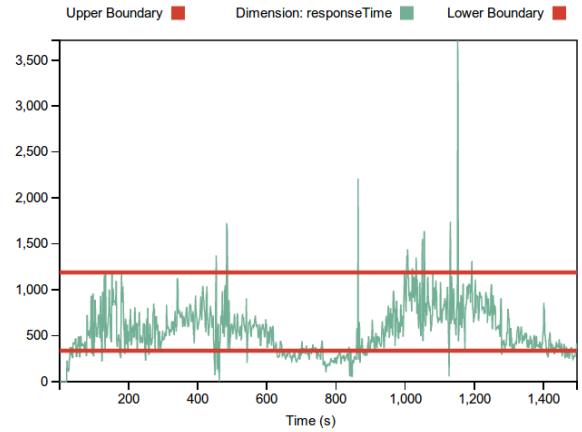


Figure 14: Response Time for `Event Processing Composite Component`

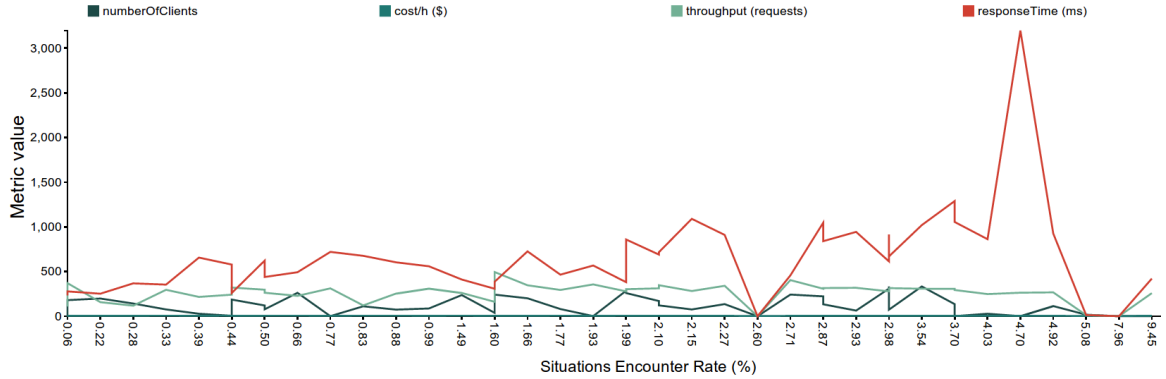


Figure 15: Elasticity Pathway for Event Processing Composite Component

CELAR facilitates the intuitive, user-friendly description of cloud applications to be elastically controlled, together with their elasticity requirements, which can be both expressive for advanced users and simple for inexperienced ones. Using this description, CELAR analyzes and controls the application, managing cloud resources as well as application configurations for fulfilling user's requirements. Moreover, the CELAR user is continually informed on the cloud service behavior, being able to better understand the application and the consequences of different requirement preferences.

## 7 Conclusions and Future Work

In this chapter we presented CELAR approach to cloud application elasticity control. We have shown that the complexity of cloud application elasticity control is highly dependent on the application complexity, on the underlying infrastructure possibilities, and on the requirements that the cloud application stakeholders have. We have shown how CELAR facilitates the description of cloud applications, and the description of stakeholder requirements with reference to various application parts. Moreover, we have presented our control approach, integrating multi-level elasticity monitoring, analysis and control, for fulfilling the specified requirements.

As CELAR project is ongoing, we will further focus on studying and developing additional analysis, enforcement and control mechanisms, tailored to improve the elasticity of a wider range of cloud applications. Moreover, for improving the quality of our elasticity control plans, we will study and develop mechanisms for estimating the behavior of cloud application and individual components. We plan to provide CELAR both as an integrated platform for designing, deploying, monitoring and controlling elastic cloud services, and as individual components which can be embedded in existing platforms.

## References


Ananthanarayanan, G., Agarwal, S., Kandula, S., Greenberg, A., Stoica, I., Harlan, D., & Harris, E. (2011) Scarlett: Coping with Skewed Content Popularity in MapReduce Clusters. In *Proceedings of the sixth conference on Computer systems*. ACM, New York, NY, USA, 287-300.

Aspnes, J., Kirsch, J., & Krishnamurthy, A. (2004). Load Balancing and Locality in Range-Queriable Data Structures". In *Proceedings of the Twenty-third Annual ACM Symposium on Principles of Distributed Computing*, PODC '04, pages 115–124, St. John's, Newfoundland, Canada, 2004. ACM.

Bharambe, A. R., Agrawal, M., & Seshan, S. (2004). Mercury: Supporting Scalable Multi-Attribute Range Queries. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '04, pages 353–366, Portland, Oregon, USA, 2004. ACM.

Binz, T., Breitenbücher, U., Haupt, F., Kopp, O., Leymann, F., Nowak. A., & Wagner, S. (2013) OpenTOSCA – A Runtime for TOSCA-Based Cloud services. In *Proceedings of the 11th International Conference*, ICSOC 2013, Berlin, Germany, December 2-5, 2013, pp 692-695, doi: 10.1007/978-3-642-45005-1_62

Chaisiri, S., Lee, B.-S., & Niyato, D. (2012) Optimization of Resource Provisioning Cost in Cloud Computing, *IEEE Transactions on Services Computing*, vol. 5, no. 2, pp. 164-177, 2012 doi: 10.1109/TSC.2011.7

Copil, G., Moldovan, D., Truong, H.-L., & Dustdar., S. (2013a) SYBL: an Extensible Language for Controlling Elasticity in Cloud Applications. *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing - CCGRID2013*, Delft, the Netherlands, May 14-16, 2013.doi:10.1109/CCGrid.2013.42

Copil, G., Moldovan, D., Truong, H.-L., & Dustdar S. (2013b) Multi-level Elasticity Control of Cloud Services,the *11th International Conference on Service Oriented Computing*. Berlin, Germany, on 2-5 December, 2013. doi:978-3-642-45005-1

Cox, B., Allsopp, J., Moldovan, D., Star, K., Garcia ,U., & Le, D. H. (2014) CELAR Deliverable: Cloud Policy Game Design Document. http://www.celarcloud.eu/wp-content/uploads/2014/04/celar_d7.1_finalrelease_1.pdf

DeCandia, G., Hastorun, D., Jampani, M., Kakulapati, G., Lakshman, A., Pilchin, A., Sivasubramanian, S., Vosshall, P., & Vogels., W. (2007) Dynamo: amazon's highly available key-value store. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles* (SOSP '07). ACM, New York, NY, USA, 205-220

Di Cosmo, R., Mauro, J., Zacchiroli, S., & Zavattaro, G. (2013) Component Reconfiguration in the Presence of Conflict. In *Proceedings of ICALP 2013*, Part II, pages 187 -- 198. Springer, 2013

Di Nitto, E. (2013) Supporting the Development and Operation of Multi-Cloud services: The MODAClouds Approach. *15th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC)*, 23-26 Sept. 2013

Dustdar, S., Guo Y., Satzger, B., & Truong, H.-L. (2011) Principles of Elastic Processes, *Internet Computing, IEEE* , vol.15, no.5, pp.66,71, Sept.-Oct. 2011

Gambi, A., Moldovan, D., Copil, G., Truong, H.-L. & Dustdar, S. (2013) On estimating actuation delays in elastic computing systems. In *Proceedings of the 8th International Symposium on Software Engineering for Adaptive and Self-Managing Systems* (SEAMS '13). IEEE Press, Piscataway, NJ, USA, 33-42

Gonzalez, A. J., & Helvik, B. E. (2012) System management to comply with SLA availability guarantees in cloud computing. *2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)*, vol., no., pp.325,332, 3-6 Dec. 2012 http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6427508&isnumber=6427477

Karger, D., Lehman, E., Leighton, T., Panigrahy, R., Levine, M., & Lewin, D. (1997) Consistent hashing and random trees: distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing* (STOC '97). ACM, New York, NY, USA, 654-663.

Karger, D., & Ruhl, M. (2004) Simple efficient load balancing algorithms for peer-to-peer systems. In *Proceedings of the sixteenth annual ACM symposium on Parallelism in algorithms and architectures* (pp. 36-43). ACM.

Konstantinou, I., Tsoumakos, D., Mytilinis, I., & Koziris, N. (2011) Fast and cost-effective online load-balancing in distributed range-queriable system. *IEEE Transactions on Parallel and Distributed Systems*, 22(8), 1350-1364.

Konstantinou, I., Tsoumakos, D., Mytilinis, I., & Koziris, N. (2013) DBalancer: distributed load balancing for NoSQL data-stores. In *Proceedings of the 2013 international conference on Management of data* (pp. 1037-1040). ACM.

Lakshman, A., & Malik, P. (2010) Cassandra: A Decentralized Structured Storage System. *SIGOPS Oper. Syst. Rev.* 44, 2 (April 2010), 35-40.

Le, D.-H., Truong, H.-L., Copil, G., Moser, O., Nastic, S., Gambi, A. & Dustdar S. (2014) SALSA: A dynamic configuration tool for cloud-based applications, *On submission*

Li, Z., O'Brien, L., Zhang, H., & Cai, R. (2012) On a Catalogue of Metrics for Evaluating Commercial Cloud Services.*2012 ACM/IEEE 13th International Conference on Grid Computing (GRID)*, pp.164,173, 20-23 Sept. 2012 doi: 10.1109/Grid.2012.15

Moldovan, D., Copil, G., Truong, H.-L., & Dustdar, S. (2013) MELA: Monitoring and Analyzing Elasticity of Cloud Services, *5'th International Conference on Cloud Computing*, CloudCom. Bristol, UK, 2-5 December, 2013

Schatzberg, D., Appavoo, J., Krieger, O., & Van Hensbergen, E. (2012) Why Elasticity Matters. *Technical Report BUCS-TR-2012-006*, Computer Science Department, Boston University, April 15, 2012.

Serrano, D., Bouchenak, S., Kouki, Y., Ledoux, T., Lejeune, J., Sopena, J., Arantes, L., & Sens, P. (2013) Towards QoS-Oriented SLA Guarantees for Online Cloud Services, *13th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, 2013, vol., no., pp.50,57, 13-16 May 2013, doi: 10.1109/CCGrid.2013.66

Simjanoska, M., Ristov, S., Velkoski, G., & Gusev, M. (2013) Scaling the performance and cost while scaling the load and resources in the cloud, *Information & Communication Technology Electronics & Microelectronics (MIPRO)*, 2013 36th International Convention on , vol., no., pp.151,156, 20-24 May 2013

Sofokleous, C., Loulloudes, N., Trihinas, D., Pallis, G., & Dikaiakos, M. (2014) c-Eclipse: An Open-Source Management Framework for Cloud Applications, *EuroPar 2014*, Porto, Portugal 2014

OASIS, Topology and Orchestration Specification for Cloud Applications (TOSCA), (2013) http://docs.oasis-open.org/tosca/TOSCA/v1.0/cs01/TOSCA-v1.0-cs01.html

Trihinas, D., Loulloudes, N., Moldovan, D., Sofokleous, S., Pallis, G., & Dikaiakos, M. D. (2013) CELAR Deliverable: Cloud Monitoring Tool V1, http://www.celarcloud.eu/wp-content/uploads/2013/11/Cloud-Monitoring-Tool-V1.pdf

Trihinas, D., Pallis, G., & Dikaiakos, M. D. (2014a) JCatascopia: Monitoring Elastically Adaptive Applications in the Cloud, in *14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (to appear)*, 2014.

Villegas, D., Antoniou, A., Sadjadi, S. M., & Iosup, A. (2012) An Analysis of Provisioning and Allocation Policies for Infrastructure-as-a-Service Clouds.In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)* (CCGRID '12). IEEE Computer Society, Washington, DC, USA, 612-619. DOI=10.1109/CCGrid.2012.46

Xing, W., Tsoumakos, D., Sofokleous, S., Liabotis, I., Floros, V., & Loverdos, C. (2014) CELAR Deliverable: Translational Cancer Detection Pipeline Design http://www.celarcloud.eu/wp-content/uploads/2014/05/celar_d8.1_finalrelease.pdf