Automatic, multi-grained elasticity-provisioning for the Cloud

Decision-Making Module V1

Deliverable no.: D5.2
Date: 26 March 2014

CELAR is funded by the European Commission DG-INFSO
Seventh Framework Programmed, Contract no.: 317790

Contents

List of Figures

List of Tables

List of Abbreviations

API	Application Programming Interface
AWS	Amazon Web Services
DBalancer	Data Balancer
EBS	Elastic Block Storage
ECP	Elasticity Control Plans
eSTM	Elasticity State Transition Model
FCO	Flexiant Cloud Orchestrator
FR	Functional Requirement
IaaS	Infrastructure as a Service
M2M	Machine to Machine
MELA	Monitoring ELasticity of cloud Applications
NAS	Network Attached Storage
NoSQL	No Structured Query Language Database
PaaS	Platform as a Service
REST	Representative State Transfer
rSYBL	runtime of SYBL (Simple Yet Beautiful Language)
SaaS	Software as a Service
SALSA	Settling And Launching Sky Applications
SAN	Storage Area Network
SOM	Self Organizing Map
SYBL	Simple Yet Beautiful Language
TOSCA	Topology and Orchestration Specification for Cloud Applications
VM	Virtual Machine

Deliverable Title	Decision Making Module V1
Filename	CELAR_D5.2_finalrelease
Author(s)	Georgiana Copil, Daniel Moldovan, Hung Duc Le, Hong Linh Truong, Schahram Dustdar, Demetris Trihinas, Stalo Sofokleous, Nicholas Loulloudes, Athanasios Foudoulis, Craig Sheridan, Evangelos Floros
Date	26.3.2014

Start of the project:1.10.2012
Duration: 36 Months
Project coordinator organization: ATHENA RESEARCH AND INNOVATION CENTER IN INFORMATION COMMUNICATION & KNOWLEDGE TECHNOLOGIES (ATHENA)

Deliverable title: Decision-Making Module V1
Deliverable no.: D5.2

Due date of deliverable: 31 March 2014
Actual submission date: 26 March 2014

Dissemination Level

x	PU	Public
	PP	Restricted to other programme participants (including the Commission Services)
	RE	Restricted to a group specified by the consortium (including the Commission Services)
	CO	Confidential, only for members of the consortium (including the Commission Services)

Deliverable status version control

Version	Date	Author
1.1	26.03.2014	Georgiana Copil, Daniel Moldovan, Hung Duc Le, Hong Linh Truong, Schahram Dustdar, Demetris Trihinas, Stalo Sofokleous, Nicholas Loulloudes , Athanasios Foudoulis, Craig Sheridan, Evangelos Floros
1.0	24.03.2014	Georgiana Copil, Daniel Moldovan, Hung Duc Le, Hong Linh Truong, Schahram Dustdar, Demetris Trihinas, Stalo Sofokleous, Nicholas Loulloudes , Athanasios Foudoulis, Craig Sheridan, Evangelos Floros
0.9	19.03.2014	Georgiana Copil, Daniel Moldovan, Hung Duc Le, Hong Linh Truong, Schahram Dustdar, Demetris Trihinas, Stalo Sofokleous, Nicholas Loulloudes , Athanasios Foudoulis, Craig Sheridan, Evangelos Floros
0.8	18.03.2014	Georgiana Copil, Daniel Moldovan, Hung Duc Le, Hong Linh Truong, Schahram Dustdar, Demetris Trihinas, Stalo Sofokleous, Nicholas Loulloudes , Athanasios Foudoulis, Craig Sheridan, Evangelos Floros
0.7	17.03.2014	Georgiana Copil, Daniel Moldovan, Hung Duc Le, Hong Linh Truong, Schahram Dustdar, Demetris Trihinas, Stalo Sofokleous, Nicholas Loulloudes , Athanasios Foudoulis, Craig Sheridan, Evangelos Floros
0.6	10.03.2014	Georgiana Copil, Daniel Moldovan, Hung Duc Le, Hong Linh Truong, Schahram Dustdar, Demetris Trihinas, Stalo Sofokleous, Nicholas Loulloudes , Athanasios Foudoulis, Craig Sheridan, Evangelos Floros
0.5	7.03.2014	Georgiana Copil, Daniel Moldovan, Hung Duc Le, Hong Linh Truong, Schahram Dustdar, Demetris Trihinas, Stalo Sofokleous, Nicholas Loulloudes , Athanasios Foudoulis, Craig Sheridan, Evangelos Floros
0.4	18.2.2014	Georgiana Copil, Daniel Moldovan, Hung Duc Le, Hong Linh Truong, Schahram Dustdar, Demetris Trihinas, Stalo Sofokleous, Nicholas Loulloudes , Athanasios Foudoulis, Craig Sheridan, Evangelos Floros
0.3	11.2.2014	Georgiana Copil, Hong Linh Truong, Daniel Moldovan, Hung Duc Le, Craig Sheridan, Evangelos Floros
0.2	22.1.2014	Georgiana Copil, Craig Sheridan, Evangelos Floros
0.1	16.1.2014	Georgiana Copil

Abstract

This document describes the first implemented version of the Decision Module, and presents the research progress achieved from deliverable D5.1 [D5.1]. We report our research progress on computational resources elasticity and on data elasticity. We present the refinements of the Decision Module architecture w.r.t. its iterative implementation process, and show how the design described in D5.1 [D5.1] has been realized. We describe our implementation, giving detailed descriptions of technologies used, internal components and interactions within Decision Module as well as with other CELAR components. The first version of the Decision Module prototype has been implemented and initial tests have been carried out, showing that the Decision Module scales the application according to the user-defined requirements and the application load.

Keywords

Elasticity, computing elasticity, data elasticity, learning, prototype, rSYBL, MELA, SALSA

1. Introduction

This deliverable is a continuation of D5.1 [D5.1], which describes the design of the Decision Module. Figure 1 shows the core decision flow presented in D5.1 [D5.1], starting from gathering necessary information (e.g., monitoring information, and elasticity requirements) to the generation of elasticity control plans. In this deliverable, the core flow of the Decision Module presented in detail in D5.1 [D5.1] has been improved in several aspects, such as smart deployment configurations generation, monitoring data analysis, modeling effects of applying different control actions, analyzing existing elasticity control mechanisms and integrating the different elasticity control mechanisms.

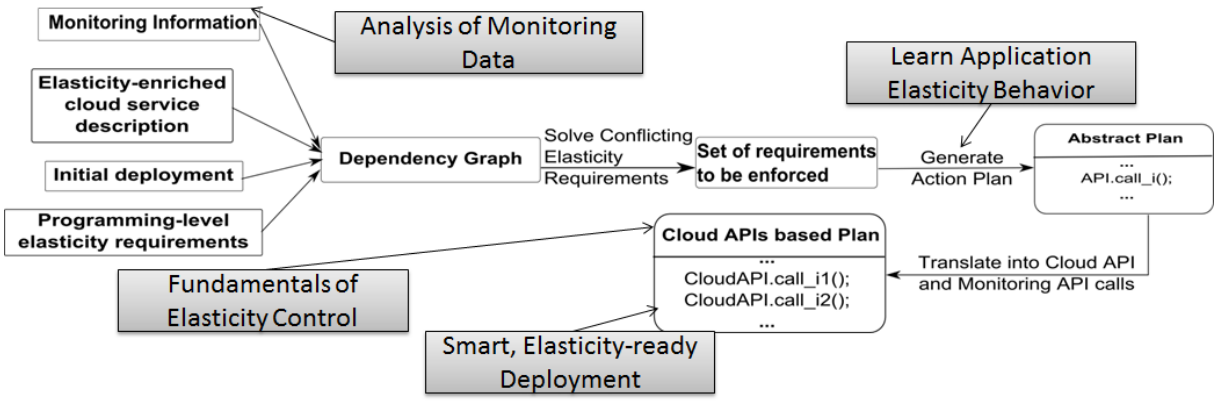

Figure 1: Decision Flow

This deliverable presents the research progress achieved so far within WP5 (Sections 3 and 4, gray components from Figure 1), and describes the implementation details of the Decision Module with the design presented in D5.1 (Section 5), which has been shortly described in MS11 [MS11]. We also introduce new use-cases and requirements for the Decision Module, in addition to those described in D5.1 (Section 3).

Applications to be controlled are potentially complex, as they are composed of multiple components of various types, with different relations among them and with the virtual infrastructure (as detailed in D5.1 Section 4.1 [D5.1]).

We present new requirements (in addition to the ones already described in D5.1) for the Decision Module in order to facilitate data-aware elasticity control (Section 2). We describe the fundamentals of measurement and control for computing (Section 3.1) and data (Section 3.3) elasticity. In other words, we show how we can manage the different application components in different contexts. For instance, as a web service component can be deployed within a web container, for this component a number of control actions are possible: (i) controlling elasticity of the web service (e.g., changing its configurations), (ii) controlling the web container (e.g., controlling threads pool for web server), (iii) controlling the virtual machine (e.g., adding more resources, or modifying associated network/storage), (iv) controlling the virtual data center (e.g., changing its location and availability zone), and (v) controlling using cloud-provider specific APIs (e.g., billing control and user-specific controls).

Moreover, when examining data components which focus on data processing, data access or data storage, we have other issues to consider. For instance the fact that the cost is computed depending on the number of I/Os, that the distributed storage has

different performance than local storage and different control actions available, and that the metrics related to data are different from the usual ones referring to computation (e.g., we have a rich set of metrics related to data which are application-specific).
Using the concepts described in Section 3, we propose in Section 4 *new mechanisms* for the elasticity control of cloud applications, considering the different levels at which we can control elastic applications, and the types of components or composite components (e.g., for composite component 2 we should choose a control mechanism suitable for data elasticity control, while for component 1 we can choose a more generic one fit for computing elasticity control) that we are controlling.
The implementation of the above *new mechanisms* will be described in detail in Deliverable D5.3, while, from the implementation point of view, this deliverable contains in Section 5 the detailed description of Decision Module V1, achieved in correspondence to the design described in D5.1.
The document is structured as follows: Section 2 presents new or refined requirements, in relation to data elasticity, Section 3 presents fundamental mechanisms of controlling data and compute elasticity, Section 4 describes mechanisms of integrating different types of elasticity control. Section 5 describes current implementation of the Decision Module, while Section 6 concludes the deliverable.
This report is partially based on several papers which have been published or under submission [Copil, 2013b] [Moldovan, 2013], [Moldovan, 2014], [Copil, 2014a], [Copil, 2014b].

2. Refined Decision Module Requirements

We have refined the functional requirements presented in Section 3.3 of D5.1 to also reflect the data-oriented approach of elasticity control, which is part of Task T5.3 of WP5, starting in M8 of the project.

To study the impact of data-related decisions on the overall application, we analyze data-related use-cases from our CELAR IaaS providers' experience, and possible requirements from CELAR users, Playgen and UNIMAN, with gaming and respectively cancer research applications.

2.1 Data Related Use-Cases

2.1.1 Common User of a IaaS Cloud Provider

Let us consider Bob who is a user of a cloud provider company. Bob has a process that creates 10GB of data every day, in a batch job, but he wants to keep his historical data (e.g., the last two years) online in case he needs it again. He needs extremely fast I/O for doing his daily tasks, but doesn't mind if his data is being archived to slower disks on the same virtual machine for the future. If he needs it again he will only need to copy one day's over back onto faster disks. He is willing to pay a price premium for the faster disks, but accordingly would also like for it to be cheaper for the archive disk.

- Service Provider perspective

  This use case could already be covered with appropriate disk I/O charging, but it may well be likely because of Bob's high I/O requirements that he wants to run his daily jobs on an SSD array, which of course wouldn't make any sense for the long term storage. The Service Provider would like to have high, normal and low I/O capable storage which are charged at different price units, but all of which can be provisioned to the same VM.

- CELAR perspective

  Considering Bob's perspective and requirements, and the service provider description available the in Information System module, CELAR decides on a smart deployment which provisions both fast and slow disks to be used by the user.

This use case shows that the user wants diversified data products, which the service provider is willing to provide. However, the user always wants automatic management of changing between these different products, of resizing on demand, and controlling upon the user requirements.

2.1.2 Cancer Research Application

Let us consider the WP8 application (please see deliverable D8.1 [D8.1] for more details). It is a scientific application which has the form of SCAN pipelines to capture and analyze genomic, proteomic and clinical information to find the "right" patient treatment plan for cancer patients.

SCAN pipeline comprises four processes: A) NGS data process with Linux system; B) Mass Spectrometry sample data process with Windows system; C) Cell Image data process with Linux Web server; D) Integrative network analysis with Linux system.

UNIMAN, which develops the SCAN pipeline, knows how required resources depend on the intermediary data, and the Decision Module can profit from this knowledge. For example, "BWA" will need 16 CPUs + 32G RAM for each pair of sample data; "GATK" will ask for 8 CPUs + 64G RAM for one bam file. Depending on the size of intermediary data, and maybe on the quality of data the Decision Module can allocate more or less

resources, depending also on cost requirements.
Moreover, depending on the overall cost requirements, the Decision Module can start the SCAN pipeline with more or less initial data, producing the results in varying time and with varying cost.

2.1.3 Gaming Application

For Playgen's gaming application (see deliverable D7.1 [D7.1] for more details), there is a huge amount of data being created all the time. Some of it may only be around for a short while, while the user decides it is valuable and uses it in their exploration and what is deemed of no value can then be destroyed.
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. There are two types of data: Volatile and Persistent. Some of the datasets are persistent, data not being changed as for the user's point of view. Gamification also falls into the persistent category. However, the volatile data, including, e.g., RSS feeds, Twitter, results from other archives are tracked in order to find interesting subjects, and, unless they are deemed useful, they can be removed. For this kind of data, *data freshness* is an important factor, as one needs to have as fresh as possible trending-related data.
Data consistency is really important since clients need to be able to rely on the data they are analyzing. For the data tables, they do not need to be fresh as they do not change once they are imported unless it is a temporary table for the client to edit tables on their end.

2.2 Refined Requirements

Considering the use-cases presented in Section2.1 we have refined the functional requirements (FR) presented in Section 3.3 of Deliverable D5.1 [D5.1], in order to further incorporate ongoing research and CELAR user requirements related to data.

FR 1—Using user-defined data-specific elasticity capabilities
The CELAR user can define, with the help of the CELAR expert, configuration/reconfiguration scripts to be run as part of an elasticity capability of the application, or as singleton elasticity capabilities (e.g., reconfiguring a component, using a different algorithm/mechanism inside the application).

FR 2—Consider multi-grained control effects on multiple application parts
The user will be able to describe his/her own elasticity capabilities, actions to be taken at different moments in time or simply actions that are application specific and that affect the application in some way or another. The user would not be expected to also provide effects of the respective actions, and thus the Decision Module needs to know which the impact of each action is, even if it is user-defined. The Decision Module employs the learning process described in Section 4.2, for learning the behavior of the application when enforcing different elasticity control plans.

FR 3— Give higher priority in the decision process to data-related metrics for the data-related application parts and computing-related metrics for the computing-related application parts

Even though it might seem that all actions can be treated in the same way, since they affect the monitored metrics of the application in some way or another, in fact, acknowledging their differences might help improve the decision process. For instance, we need to treat differently data-related application parts (components, composite components or the entire application), with their elasticity capabilities from compute-related application parts. Even if we may have the same decision algorithms, the actual mechanisms differ, as we need to look at different types of metrics, give higher weights to the ones which are specific to the application part, and use control mechanisms which are specific to compute, respectively data resources.

FR 4—Evaluate and consider in the decision process how data specific application behavior affects computing parts of the application and vice-versa

As we are introducing two types of decision mechanisms, data-oriented elasticity control and respectively compute-oriented elasticity control, we need to consider also how data-specific control mechanisms applied on the data-oriented part of the application affect the compute-oriented part, and vice-versa. This needs to be done in order to be able to generate complex elasticity control action plans, for instance if we take an action that might decrease the quality of compute part (e.g., component or composite component) but improve reliability of data-oriented part, we need to be able to also take corrective decisions in advanced for the compute-oriented application part.

3. Fundamental Control Mechanisms for Computing and Data Elasticity

This section is a continuation of D5.1 Section 4, presenting novel concepts, ongoing research and early results for both computing resource elasticity and data elasticity.

3.1 CELAR Providers Computing and Data Resources Elasticity Offerings

We firstly consider the possibilities of runtime reconfiguration offered by the CELAR cloud providers, Flexiant and ~okeanos. Table 1 presents the fundamental control mechanisms available for computing and network resources, while Table 2 presents data elasticity control mechanisms, as exposed by CELAR providers. These control mechanisms are used, together with application-specific control mechanisms, in order to construct control processes which are triggered by the Decision Module, in order to control application elasticity. Although they have different names for the services being offered (e.g., VM and Server refer to Virtual Machine), they have similar offerings, like create/start/reboot VM, with minor differences like 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.

We not only enumerate possible control mechanisms, but also show possible configurations out of which the Decision Module can choose for providing a smart deployment configuration. Of course, the chosen configurations for computing, network and data resources affect the quality and cost for each of the components with the respective resources associated.

Table 1: Compute 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
	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 = logical grouping of servers

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)

3.2 Computing Resource Elasticity

3.2.1 State of the Art

Managing elastic 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 [AutoScaling], Paraleap AzureWatch [AzureWatch] and RightScale [RightScale] 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 (i.e., 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 [Sutter, 2012]. Computing resources control mechanisms, for a two tier application with application server tier and data tier, are as follows: (i) vertically scale the application server tier and database backend by adding/removing to/from current instances resources such as memory and disk storage, respectively; (ii) scale the web container(s) on each application server by configuring parameters such as: thread pool size, virtual heap space, etc. Similarly, configuring parameters related to the database backend like: virtual heap space, buffer cache size, etc.; (iii) change the structure of the application. For instance, if our application initially consists of one application server, when adding more instances a load balancer is required; (iv) move application to different location or availability zone; (v) consider a hybrid approach where the controller based on the demand scales the application using a combination of the previous mechanisms (e.g. scale horizontally but use larger instances). When deciding on this multitude of control mechanisms we need to take into account the relation between data and computing resources control, and the manner in which the control affects computing and respectively data metrics.

3.2.2 Decision Module Development

To facilitate complex adaptation mechanisms the Decision Module is required to take decisions based on not only low-level monitoring information. Instead, it is required to assess heterogeneous types of monitoring information of different granularity, from low-level system metrics (i.e. CPU, memory, and network utilization) to high-level application specific metrics (i.e. latency, throughput, and availability), which are collected across multiple levels(physical, virtualization, and application level) in a Cloud environment at different time intervals [Trihinas, 2014].

To enforce complex adaption mechanisms, decisions originating from an elastic resource controller must also be aware of what are the offerings and limitations of the underlying

IaaS provider. Specifically, the Decision Module must acknowledge: (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. Additionally, it is important for the decision module 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. For example, let us consider two IaaS providers (Provider A and Provider B – as in the study in previous Section, Section 3.1) where only the first provider offers its users with the capability of vertically scaling virtual instances by allocating more memory. For the cloud application deployed on Provider B, the decision-making 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 a fine-grained approach by enlarging the allocated memory segments of existing instances. This approach takes cost into consideration since resizing existing VM(s) may be cheaper than constantly initializing small virtual instances.

3.3 Data Resource Elasticity

3.3.1 State of the Art

Data elasticity, i.e., the elasticity properties of a data oriented component such as a NoSQL data-store or a distributed data base, can be seen in two different axes, namely the *clustering elasticity* and the *data-modeling* elasticity [NoSQL Cassandra].
The *clustering elasticity* has to do with what the system needs to perform when more nodes are added or removed from the data-store: in order to gain the maximum benefits of extra resources, a data-movement operation needs to be performed in order to have a more balanced data distribution between the cluster's nodes. Nevertheless, even when there are no node additions, a change on the data access or data placement distributions require a data re-organization.
On the other hand, the *data-modeling elasticity* deals with the adjustment of the initial data-schema, i.e., the semantic organization of data into columns/rows/hierarchies, etc. In a semi-unstructured scenario where the schema can be adapted, data-modeling elasticity is necessary to capture data needs. Nevertheless, this requires a deep understanding of the semantics of the data itself, rendering this adaptation not practical and not widely applicable. In the following, when we refer to data resource elasticity, we will consider only clustering elasticity.
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 nodes/storage 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.
As it is already implied, load balancing can be approached in different ways. One important aspect of it is the data placement skew. In a NoSQL cluster, data are expected to be distributed equally among the nodes. Nodes with proportionally larger amount of

data may present degraded performance. This problem is well studied and towards this direction, there is a lot of work that has been done.
For data elasticity control, next to traditional approaches of scaling NoSQL databases, we also focus on data balancing techniques, since they might be more valuable for situations where the data is, e.g., highly skewed, and scaling the NoSQL database could have little or no effect. In order to understand what the existing approaches for data balancing control are, we study current state of the art.
Consistent hashing techniques [Karger, 97] are a common and effective solution for data control. The majority of modern NoSQL stores [Lakshman, 2010], [DeCandia, 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 [Bharambe, 2004], [Aspnes, 2004], [Ganesan, 2004], [Karger, 2004], [Konstantinou, 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. In a highly skewed data access distribution, where a small portion of popular data may get the majority of the applied load [Ananthanarayanan, 2011], the system performance may degrade even in over provisioned infrastructures.
Considering CELAR requirements focusing on NoSQL data, in what follows we discuss the properties of DBalancer [Konstantinou, 2013], 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.
The independency from the algorithm is another feature of DBalancer. The DBalancer currently supports the NIXMIG algorithm [Konstantinou, 2011] and Item Balancing algorithm [Karger, 2004] in order to achieve balancing. Nevertheless, new load balancing algorithms can be easily defined by giving a set of message types and actions along with the appropriate trigger conditions. What is more, the DBalancer does not need any central co-ordination point in order to operate. It is installed on a per-node basis and is functional even when only a portion of the network's nodes is operational.
Furthermore, it has been designed to have the least impact on the running node. Only a small number of sync and probing messages are utilized in order to coordinate the load balancing procedure.

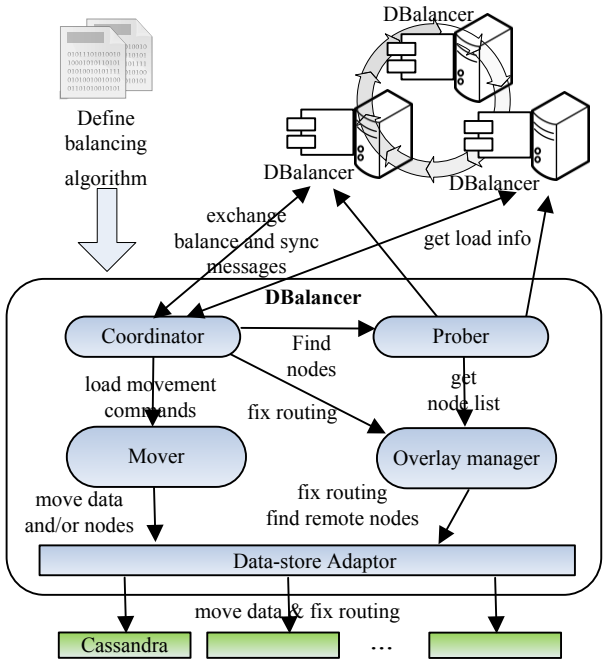

Figure 2: DBalancer Architecture [Konstantinou, 2013]

The DBalancer (see DBalancer architecture in Figure 2) features an architecture that is illustrated in the following figure. The DBalancer consists of the following modules:

- *Coordinator Module*: It initiates or participates in balancing decisions. It takes as input and implements the user's load balancing algorithm. It utilizes JCatascopia to collect load status, current state. This module decides each node's role in the balancing procedure according to the collected messages and load statuses. It also resolves all conflicts that may occur because of concurrent load balancing attempts and makes sure that the whole procedure is being carried out.
- *Prober Module:* It locates remote nodes and examines their load. Remote nodes are found by contacting the Overlay manager module, which contains information about other data-store nodes. It exchanges simple messages containing load statuses and maintains a list with the observed nodes and their respective loads. This list is used by the coordinator module when it wishes to locate remote underloaded nodes. The prober module will be integrated with the JCatascopia module.
- *Mover module:* it translates higher level load movement commands into specific item exchanges or node migrations. When the specific actions have been calculated, they are forwarded to the lower-level Data-store adaptor module which translates them into application specific calls.
- *Overlay Manager Module:* It abstracts lower-level data-store specific routing information and operations. It connects with the Data-store Adaptor module to get and set routing table information. The Coordinator module contacts this module when the load movement commands require overlay maintenance operations. For instance, during node migrations the routing table entries need to be refreshed. Moreover, this module also provides a list with node addresses to the prober module which acts as a pool for possible message receivers.
- *Data-store Adaptor module:* It hides the complexities of the underlying data-store implementations, by offering a set of abstract operations to the higher level modules. For each supported data-store, a set of different commands need to be

implemented or utilized. For instance, a node movement operation in the Cassandra case results in a nodetool move command, whereas a data movement operation in the HBase case results in a set of HDFS data block movements between different DataNodes.

The DBalancer's modules will be extended and integrated with the Elasticity Provisioning Platform's modules.

3.3.2 Decision Module Development

One important issue related to load balancing that needs to be discussed is the optimality of the proposed solutions. Most research load balancing systems employ approaches that seem to be suboptimal. Even worse, productive NoSQL systems do not use any mechanism to optimize the load balancing procedure. For instance, Cassandra's load balancing [Cassandra, 2013] which targets the data placement skew is performed only during node additions by transferring half of the keys of the most loaded node, or with a user-executed script. In order to balance the cluster, the controller should change the token of some nodes. This operation cannot happen in parallel. The order with which the user will modify the token of the overloaded nodes can greatly affect performance. In practice, this operation is an overkill for performance, since it based on the judgment of the user and is not carried out by an automated system.

In order to minimize the amount of transferred data and provide strong optimality guarantees, it may be needed to abandon the fully distributed approach to the problem, in favor of the full picture of load distribution. Typically, load balancing schemes were used in P2P systems, where there are requirements for running in a fully distributed fashion. Still, not only these strict requirements about decentralization are an overkill in typical NoSQL deployments but also lead to suboptimal data movement and placement during load balancing. For example, in a distributed system, it is advised, for scalability reasons, to avoid the use of a central load registry, since its maintenance is considered expensive, and can also be a SPOF (single point of failure) element. This single requirement prevents nodes from having a full picture of the network workload and inevitably leads to suboptimal decisions during load balancing. Obviously, this requirement can be relaxed in typical NoSQL deployments, both because of the different scale and the network connectivity quality.

Having realized the need for an optimal solution for the data placement skew problem, we propose a generic approach to redistribute data among nodes. The main idea is to utilize general purpose data placement commands typically found in modern data stores, in order to offer a general approach that can be applied to any data store. As DBalancer already makes use of such commands and possesses a general interface which can interact with most state-of-the-art NoSQL datastores, our proposed approach can be implemented as an extra module, built on top of DBalancer. This module will implement an algorithm which will provide load balancing with minimum data transfers and at minimum balancing time.

As a first step, we can use the monitoring information offered by JCatascopia to identify unbalanced clusters. Once imbalances are detected, the goal is to calculate the optimal data placement strategy to bring the system to a balanced state by exploring possible data movement plans.

Since the optimal solution may be computationally hard and may demand the whole view of the system, the proposed solution cannot be applied to situations where the load changes too fast and thus requires a constant workload adaptation by performing data intensive operations. According to the workload variations, we will investigate whether

the benefits of a load balancing operation surpass the performance degradations caused by the balancing procedure itself. In other words, we will identify whether it is advantageous to start a load balancing operation, according to the variations of the observed recent workload.
This new module will utilize the DBalancer's capability to communicate with the NoSQL datastore. Moreover, it will have a global knowledge of the system and its load. Thus, it can compute the optimal plan for data redistribution and it can execute it, without any user involvement and no matter what is the underlying datastore.

4. Computing and Data Elasticity Control Mechanisms for Cloud Applications

For elasticity control, we use the application model presented in D5.1 Section 4.1 [D5.1], in order to be able to process all the various types of information concerning the cloud application, including application structure, used virtual resources, elasticity metrics and the capabilities of components or of resources to change, which is related to the way application and parts of application (e.g., component, composite component) behave and to the manner in which it can be controlled. This representational model is at runtime a graph, with various types of edges (e.g., "composition relationship" edge, or "runs on" edge).

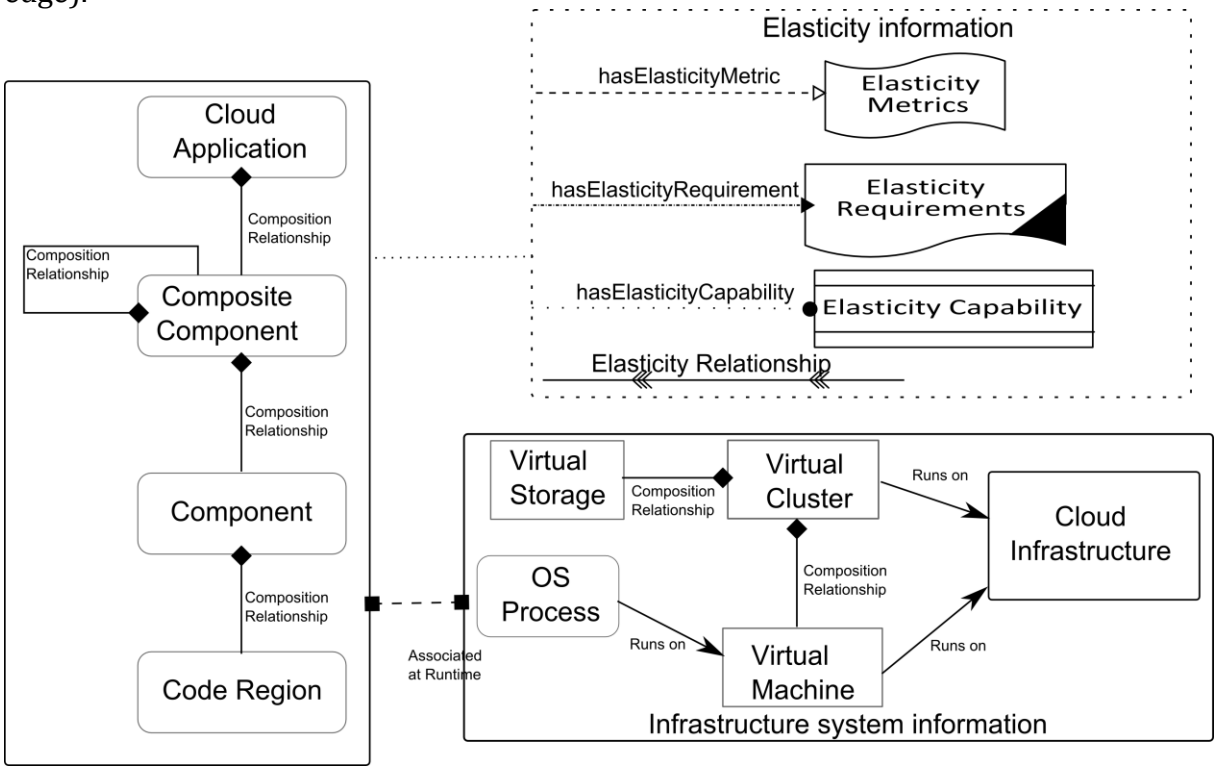

Figure 3: Cloud Application Model [D5.1]

4.1 Data and Compute Elasticity Control Mechanisms for Cloud Applications

Using all the mechanisms of control covered in Sections 3.1 and 3.3 which are focusing on application component level, we design different decision mechanisms which are fit for data-oriented control, for compute-oriented control, and for coordination among them. As shown in Figure 4, for taking control decisions on data-related components or composite components, we need to take into account data-related monitoring information and to enforce data-specific control mechanisms (e.g., rebalancing).

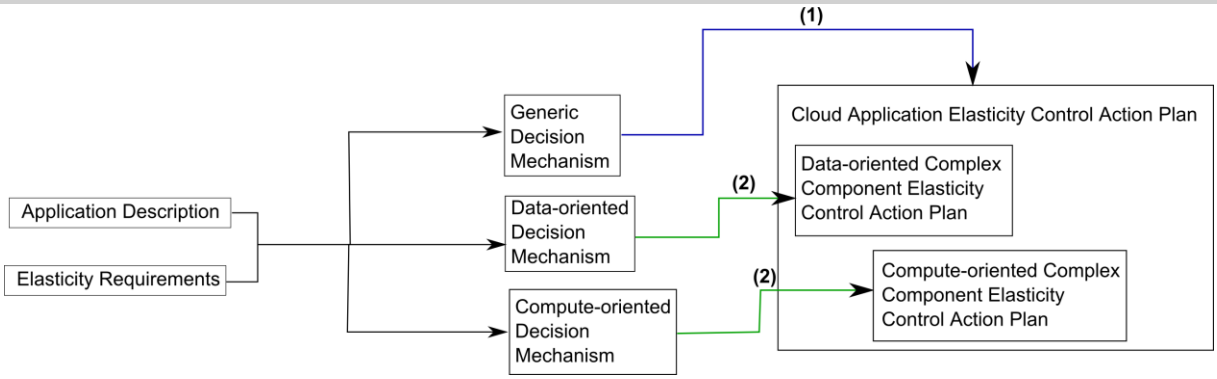

Figure 4: Different Mechanisms of Elasticity Control Depending on the Composite Components Types

We therefore have a pluggable mechanism, together with a coordinating one (`Generic Decision Mechanism` in Figure 4), which, depending on the application-related information received from c-Eclipse and from the Profiler from Elasticity Provisioning Platform described in detail in Deliverable D3.1 [D3.1], decides which type of control mechanism needs to be used on each composite component.
Moreover, considering the action effect on each of the application parts (e.g., scaling at data layer effect upon the business layer), we decide which corrective actions to apply on each level of the application, taking into consideration user's elasticity requirements.
The `Generic Decision Mechanism` handles this coordination among `Data-oriented Decision Mechanism` and `Compute-oriented Decision Mechanism`, balancing for instance data control with the necessary compute-level actions and vice-versa.

4.1.1 User-Driven Control of Application Elasticity

c-Eclipse (Deliverable D2.1 [D2.1]) provides mechanisms for Cloud application developers to specify custom control mechanisms for applications, at different levels (e.g., component, composite component). Once they have described the structure of their application according to the abstract application composition model described in D5.1, Section 4.1.1 [D5.1], they are allowed to specify elasticity requirements at any level of their application's structure. Some of the elasticity requirements can be generic (i.e. an elasticity strategy to minimize deployment costs), while others are IaaS provider specific if the elasticity actions to be enforced are supported only by the selected provider(s) (i.e. memory ballooning for a specific application component). Generic elasticity requirements can be specified via the Properties View of c-Eclipse, during the application description process D2.1 [D2.1]. Provider specific elasticity requirements can be specified again through the Properties View of c-Eclipse during the application submission process[1], when the provider to deploy the application is selected.
The elasticity requirements specified in c-Eclipse are transformed into SYBL directives, as described in D5.1, Section 4.1.2 [D5.1] and injected into the TOSCA application description as policies [TOSCA Policies]. To this extent, we have defined two types of elasticity-oriented TOSCA policies using SYBL: i) *Elasticity Constraint* and ii) *Elasticity Strategy*. The Elasticity Constraint type is used to express the desirable state of an application, related to cost, performance and other application-quality metrics. Here the application user does not specify the exact actions to be enforced when a constraint is violated or fulfilled. Instead, the appropriate actions are determined by the underlying

[1] A detailed description of how elasticity requirements are defined by c-Eclipse users will be documented in D2.2

Decision-making Module. The Elasticity Strategy type, on the other hand, is used to express the application behavior that should be enforced by the Decision Module. We therefore enable custom behavior stirring by the CELAR user, whenever s/he has preferences with reference to the elasticity of cost, quality and resources. As opposed to the very cumbersome policies with thresholds on generic metrics that Amazon AutoScale[2] provides, CELAR enables the specification of high level descriptions on custom user metrics, as well as optimization strategies to be triggered when application has reached a specific state. The Decision Model uses these requirements and determined application behavior to be able to construct the appropriate elasticity control plan.

4.2 Elasticity Behavior Measurement

For determining the application behavior in different circumstances, especially in the context of enforcing different control actions, we propose a methodology which analyses the historical application performance (e.g., how were metrics for one composite component affected when enforcing elasticity control plan x), and determines the expected elasticity behavior (containing metrics values, in time) for different cloud service parts (e.g., component, or composite component).

Behavior of cloud applications has been studied by many research works. For instance, Verma et al. [Verma, 2010] study the impact of reconfiguration actions on system performance, Wang et al. [Wang, 2013] propose a method for evaluating the cost over different combinations of reserved and on-demand instances, Zhang et al [Zhang, 2013] propose algorithms for performance tracking of dynamic cloud applications, predicting metrics values like throughput or response time. Compared with these approaches, we also take into consideration in our process the connection among different parts of the application, and we are interested in 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 action on a completely different composite component).

4.2.1 eSTM Description and Management

In order to be able to learn from decisions taken previously and from the profiling information sent by the Elasticity Provisioning Platform Profiler, the Decision Module has to represent all this complex, highly correlated information and consider the correlations in the learning process. We define an elasticity State Transition Model (*eSTM*), which we detail bellow, which is used for representing both static, pre-deployment information, and runtime information which also contains transitions among runtime snapshots.

Figure 5 shows, on the left side, the application at a pre-deployment time, when the elasticity controller (rSYBL) knows about it only through the *Static Description* of the application.

The *Static Description* (e.g., TOSCA description) is usually known by cloud application stakeholders (e.g., cloud application provider, or cloud application developer). This is valuable information since, for instance, most of the web applications which are three tiers and have similar configurations would behave similarly when deployed on the cloud (e.g., different measured metrics exposing patterns at scale in/out operations [Gambi, 2013]). At runtime, for studying the cloud application behavior, we need multiple types of information for properly understanding the behavior of the

[2] http://aws.amazon.com/autoscaling/

application. This information is encapsulated in a *Runtime Description*, and contains both the elasticity behavior of the cloud application, e.g., the cloud application Elasticity Space obtained from MELA [Moldovan,2013], and the deployment topology together with all the provider-offered applications which are used.

On the *Runtime Description* we can apply various *Elasticity Control Plans (ECP)*[D5.1], which will transform the current Runtime Description into a new one, while keeping the same Static Description, with different deployment topology or differently configured components or composite components. We assume the static description as being fixed during runtime, and that only the runtime description will change due to enforced elasticity control processes or due to load. After enforcing the Deployment Plan (e.g., create machine x, configure software z, in case of error re-configure software z in configuration t), the application has associated a *Runtime Description*, which describes all the resources associated with it and the metrics evolution.

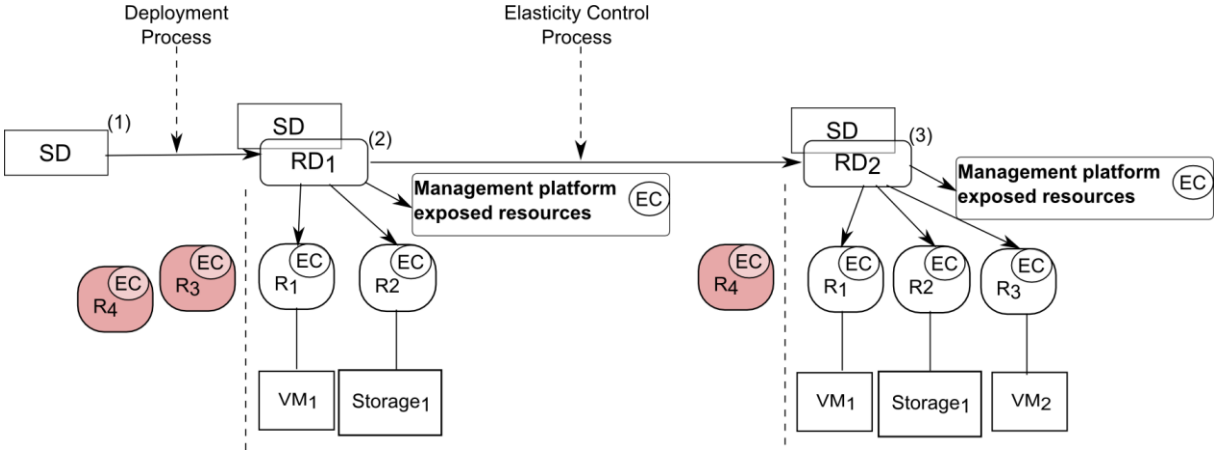

Figure 5: Elasticity Control on Cloud Applications

4.2.1.1 eSTM Concepts

The *Static Description* (*SD*) is a sub-graph in which the nodes are application parts (components or composite components), and the edges contain information on the relationship and communication between the components (e.g., composition, communication pattern).

The *Runtime Description*(*RD*) is a sub-graph of *eSTM* which has as nodes virtual resources which are associated with the application parts (e.g., component of the application), as well as Elasticity Behavior *EB* associated with *SP*, and as edges the association relationships among them, and with the nodes of *SD*. An example of a possible *RD* can be the initial deployment snapshot, with the associations between high level components and the resources used (e.g., VM with their characteristics, disks, or networks). A *SD* can have associated many *RDs*, depending on application complexity: a single *RD* can be composed of many *RDs*. The resources used by the application during runtime have a great impact on application's cost and quality metrics, thus on the Elasticity Behavior of the application. Moreover, the *SD* and *RDs* provide a thorough description of the application both pre-deployment and during the runtime execution, thus constituting a strong base knowledge for application elasticity behavior learning.

The *Elasticity Behavior* (*EB*) describes the evolution of metrics within a *RD*, with respect to the external factors (e.g., workloads or elasticity control processes), it being a list of tuples $(time, metric_{i_1} \ldots metric_{i_n})$ for each part of the application. The *EB* describes the encountered and expected changes in an application (e.g., change in metrics values, change in topologies, or change in resource properties) for a specific *RD*. The *EB* includes

metrics evolution for the components and composite components belonging to the application. For instance, for a NoSQL cluster that is described as a composite component, an example of *EB* would be $EB_{t_i-t_n} = [(t_i, latency_{t_i}, cpuUsage_{t_i}, cost_{t_i}), \dots, (t_n, latency_{t_n}, cpuUsage_{t_n}, cost_{t_n})]$. This $EB_{t_i-t_n}$ is captured only for specific events, like enforcement of an elasticity control plan or discovery of an application behavioral pattern.

Elasticity Control Plans (*ECP*) (detailed in Section 4.2, Deliverable 5.1 [D5.1]) are sequences of elasticity capabilities $ECP_i = [EC_{i_1} \rightarrow EC_{i_2} \rightarrow \cdots EC_{i_n}]$ which can be abstracted into higher level capabilities having predictable effects on the application. The *ECPs* enable the transition from one *RD* to another. 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 the more generic increasing the amount of resources is an *ECP*, more abstract, which encapsulates the two elasticity capabilities.

The *Transition* between two *RDs* is a function which maps, for each current *RD*, an *ECP* to a new *RD*. Since we assume that distinct *ECP* produce distinct RD_{target} for the same RD_{source} (otherwise the *ECP* would coincide), the defined function is injective and we can define an inverse which maps $RD_{target}$to an *ECP*. In the underlying implementation, the transition function is a process of discovering for each RD_{source} and given the *ECP* to be enforced, the expected *RD*. The inverse function is the exactly inverse process, this time benefiting the knowledge accumulated in the forward process.

$$T_{RD_{source}}: ECP \rightarrow RD,\ T_{RD_{source}}(ECP_i) = RD_{target} \quad (1)$$

$$T_{RD_{source}}{}^{-1}: RD \rightarrow ECP,\ T_{RD_{source}}{}^{-1}(RD_{target}) = ECP_i \quad (2)$$

4.2.1.2 eSTM Management

The application is described by the CELAR user in c-Eclipse in terms of its structure, elasticity capabilities and requirements associated to it. This information is fed into the *eSTM* repository. If data is available for profiling (e.g. the workload to execute, the elasticity capabilities to enforce), the profiling phase gathers information regarding deployment characteristics and behavior (e.g. evolution of metrics in response to specific stress factors) of the application on different cloud providers, enforcing different elasticity capabilities and under different external circumstances (e.g. different workload). After profiling information is gathered and stored into the *eSTM* repository, the application is deployed on the cloud, and continually monitored for finding information on its behavior. The continually updated *eSTM* repository information is used by the cloud application controller for adapting the cloud application cost and quality to the elasticity requirements specified by the cloud application developer/provider.

For learning cloud application behavioral characteristics, we gather various types of information:

- *Static descriptions* (e.g., a TOSCA specification) from the Application User describing his cloud application via c-Eclipse
- *Runtime monitoring information* (e.g., elasticity space information) from The CELAR Cloud Information and Performance layer.
- *Data for profiling* (e.g., test loads, APIs for elasticity control processes) from the user

- *Enforced elasticity control processes* from the elasticity controller that manages the cloud application

We use this information for populating the *eSTM*, and start the analysis process, that extracts behavioral patterns for transitions from a runtime description to another, and the necessary elasticity control processes needed for obtaining the desired runtime description.

4.2.2 Analysis of the Application Elasticity Behavior

Using the *eSTM* described above, the Decision Module analyzes history and produces expected effects, in time, for the different elasticity control plans. Initially, the eSTM is populated as described above, and with profiling information gathered using the CELAR profiler which is described in more details in Deliverable D3.2, Section 3 [D3.2]. The learning process is part of the Learning Unit of the Decision Module (proposed architecture deliverable D5.1 [D5.1]), and is a continuous process. After the cloud application is deployed on a cloud infrastructure, the Decision Module is in charge with taking decisions as response to resource and quality requirements of the cloud application.

4.2.2.1 Learning Process

In the learning process, we use information gathered during the profiling phase and during runtime execution, and analyze, for each composite or simple component the evolution of monitored metrics. When the Decision Module needs to find out the expected behavior for all application parts (e.g., composite component, or component), it describes recent information regarding application behavior for all application parts, and the elasticity control plan which is considered to be enforced.

In the learning process, we identify relevant parts in the metrics evolution in time, where we have had enforcements of elasticity control plans. We select *Relevant Timeseries Section* defined as follows: A *Relevant Timeseries Section* of a metric timeseries is a part of the metric's continuous evolution, which shows metric values at a specific event:

$$RTS_{metric_i} = \{metric_i^{x-\frac{SEL+ECP_{estimatedTime}}{2}}, \dots, metric_i^{ECP_{startTime}}, \dots, metric_i^{ECP_{endTime}}, \dots, metric_i^{x+\frac{SEL+ECP_{estimatedTime}}{2}}\} \quad (1)$$

In Equation 1, we see the snapshot which is being selected for the given ECP, which includes the entire elasticity control process enforcement time, plus a predetermined SEL time.

We select relevant sections for the current ECP using the same granularity, resulting in a set of metric timeseries for each part of the cloud application (composite component, or component) and elasticity control process. For being able to cluster them in order to detect correlations of evolution of metrics, we represent them as $(SEL + ECP_{estimatedTime})$-dimensional points (see Figure 6), where each point in time as a dimension of the resulting point. This way, the correlation among points belonging to an RTS, as well as their order is well determined and is used in the algorithm for determining their connection. Using the multi-dimensional points, we apply a clustering algorithm (in this case K-means) in order to find the most representative multi-dimensional point, which encodes the metric evolution in time.

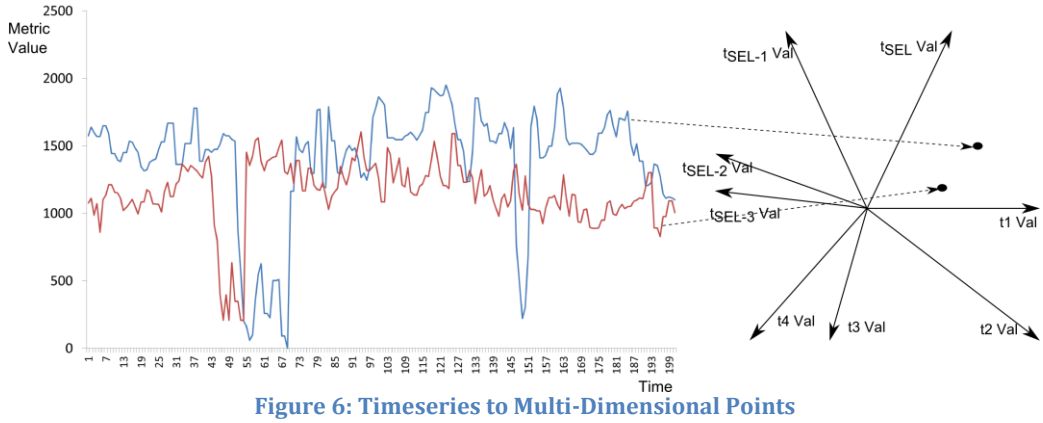

Figure 6: Timeseries to Multi-Dimensional Points

We have selected K-means as the current clustering algorithm used, with a cluster number equal to $\sqrt{SEL + ECP_{estimatedTime}}$. We are planning to compare it with other algorithms in order to determine on its suitability. After obtaining clusters of $(SEL + ECP_{estimatedTime})$-dimensional points, we create for each cloud application part 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). For knowing the effect of an ECP on a part of the cloud application, we find the tuple of cluster centroids from the clusters constructed as described above which are closest to the current metrics behavior for the part of the cloud application we are focusing on (e.g., at business layer, it would be the sequence of throughput and response time values for the last x minutes), and which have appeared together throughout the execution of the cloud application. Next, we transform the tuple of centroids into a tuple of timeseries, which show the expected evolution of current application part for the selected ECP.

4.2.1 Preliminary results

Considering a machine-to-machine (M2M) DaaS application, hosted in the cloud, and processing information coming from various data sensors, we design the application and simulate clients which send sensor information. The M2M DaaS application consists of two composite components, an `Event Processing Service Topology` and a `Data End Service Topology`. Each of these service topologies are composed of two simple components, one with a processing goal, and the other acting as a balancer or a controller of the topology. The `Event Processing Service Topology` contains Web Services which provide access to the data in order to read existing information and add new sensor information, and a Load Balancer which in this case is using HAProxy[3], for distributing the requests to the Web Services. The `Data End Service Topology` consists of a Cassandra cluster, which has components with two types of roles: Cassandra Node, and Cassandra Seed which manages the coordination among nodes. 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`. The available ECP are: `Scale In/Out Event Processing Service Unit` and `Scale In/Out Data Node Service Unit`. In what follows we show initial results on how our algorithms estimate that the different parts of the application get affected by the enforcement of the ECPs.

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

Figure 7 shows how an ECP targeting the component level affects the entire M2M Application. The Cost/Client/h (cost per client per hour) is a complex metric which depicts to the controller or the service provider 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 the number of clients, the model enables the elasticity controller to know how the M2M Application would behave in terms of expected time and expected metric fluctuations.

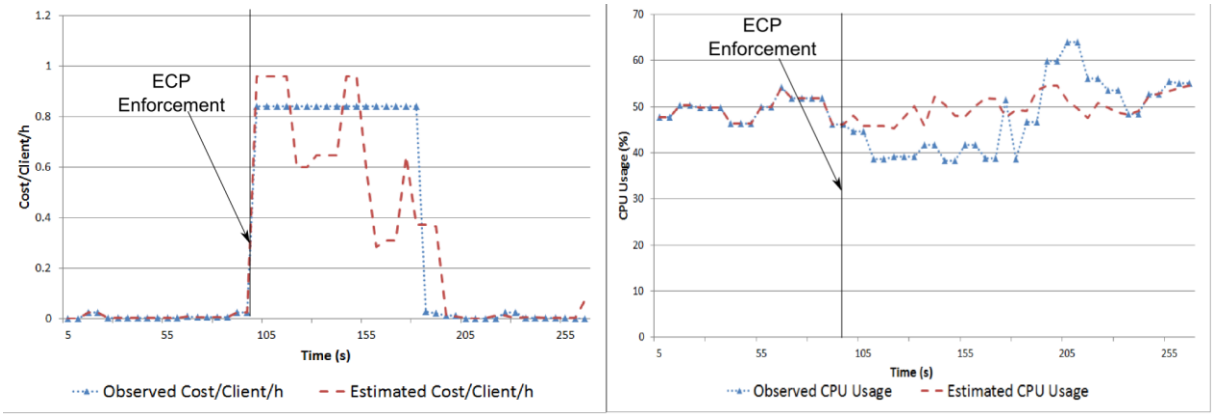

Figure 7: Effect of Data Node Scale In on the Entire M2M Application

Figure 8 shows the estimations on how the data controller of the `Data End Service Topology` is impacted by the data transferred at the enforcement of a Scale Out at Data Node level, which consists of creating a new VM, configuring the NoSQL database, acknowledging the cluster controller and joining the cluster. Therefore, even in circumstances of random workload, as is the case here, our learning process can give useful insights into how different parts of the cloud service behave when enforcing ECPs on various application parts. For the next Decision Module release, we will integrate the eSTM based learning process in the Decision Module, and improve the learning process (e.g., choose other clustering mechanisms, or learn from the different correlations among application parts).

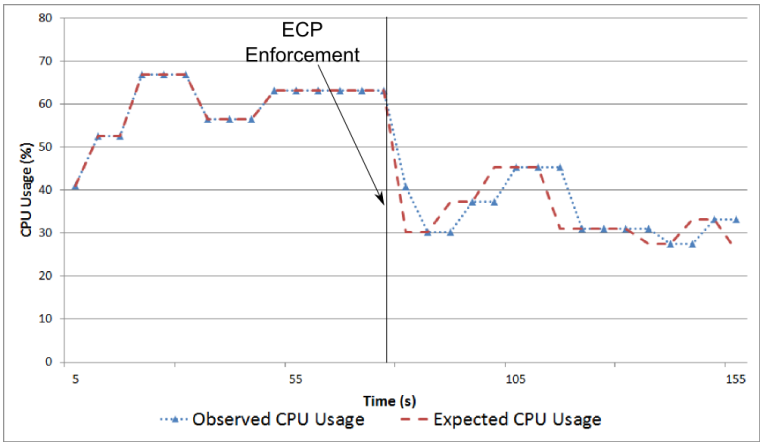

Figure 8: Effect of the Data Node Scale Out on the Data Controller

4.3 Elasticity Analysis

Based on the structured monitoring data provided by MELA-DataService implementing CELAR Multi-level metrics evaluation module, we provide an elasticity analysis mechanism. While the MELA-DataService provides monitoring snapshots containing multi-level monitoring data organized after the application structure, such snapshots do not provide information about the elasticity behavior of the application in terms of user-defined elasticity requirements fulfillment. Such information is used in the decision process of the CELAR Decision Module.

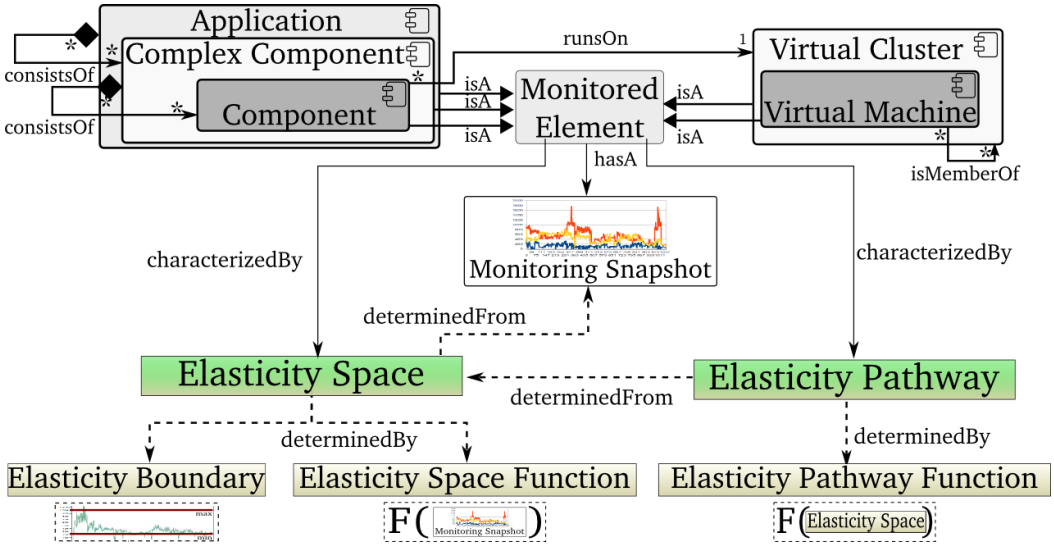

Figure 9: Elasticity Concepts Association with Cloud Application's Components

Thus, we introduce the MELA-AnalysisService, which analyzes the elasticity of cloud applications, and determines their elasticity space, elasticity boundaries, and elasticity pathway. The elasticity space, boundaries, and pathway are determined for each application component, complex components, and the whole application, providing a multi-level view over the application's elasticity (Figure 9). This view is used for analyzing the behavior of the application, whether or not it behaves within expected boundaries, in order to know what decisions to take, and in which part of the application these decisions should be enforced.

4.3.1 Elasticity Boundary

An elasticity boundary is an upper and lower limit over the application's metric's values in which user-defined elasticity requirements are fulfilled. Boundaries describe the upper and lower allowed values over a set of metrics for a component or the whole Cloud application, and are used in determining what behavioral limits are to be enforced for the application by the Decision Module.

Conceptually, an elasticity boundary is defined as: $elBoundary = \langle c_i^u, c_i^l \rangle, \langle q_j^u, q_j^l \rangle, \langle r_k^u, r_k^l \rangle$, where c_i^u *and* c_i^l denote the upper bound and the lower bound of cost metric c_i, respectively, q_j^u *and* q_j^l for quality metric q_j, and r_k^u *and* r_k^l for resource metric r_k.

We discriminate between two types of elasticity boundaries: (i) **user-defined elasticity boundary** and (ii) **evaluated elasticity boundary**. In a user-defined elasticity boundary, the boundary metrics and bounds are specified by user-defined elasticity requirements, indicating the parameters under which the cloud application is elastic. An

evaluated elasticity boundary is determined from monitoring snapshots, using the user-defined boundary.
Thus, such a boundary indicates encountered bounds on the monitored metrics not targeted by user requirements, in which the user requirements were respected. This gives requirements the decision module should enforce for application components for which the user does not know such limits. The CELAR Decision Module uses the determined elasticity boundaries to refine its control elasticity mechanism, by adding beside user-defined requirements the determined boundaries. Thus, even if the user has limited knowledge, we determine boundaries over the rest of the application. For example, from requirements only over application response time, we can determine boundaries over data end latency and CPU usage. The determined boundaries are then used in controlling the elasticity of the data end, for which we had no initial requirements.

4.3.2 Elasticity Space

Given a set of monitoring snapshots and a user-defined elasticity boundary, we need to understand when a monitored element is in elastic behavior, if its behavior violates the elasticity boundaries, i.e. it fulfills user-defined requirements. An elasticity space is determined via an elasticity space function and captures all runtime metrics described in the user-defined and determined elasticity boundaries.
An elasticity space function is designed to extract useful information about the overall behavior of the cloud application when elasticity requirements are fulfilled. A space would contain only metrics that have an impact on the metrics targeted by the user-defined boundaries. The elasticity space acts as a base for further elasticity analysis, as it stores the historical behavior of elastic applications.
More concretely, an elasticity space contains all monitored data recorded for the application components and modules, abstracted from the underlying virtual machines. As virtual machines can appear/disappear as a result to elasticity actions, keeping data associated with the logical structure of the elastic application provides the means of analyzing the behavior of individual application components, and not of individual virtual machines. More concrete details on the elasticity space are captured in the description of the implemented elasticity space function in the MELA-AnalysisService prototype, in Section 5.2.4.1.

4.3.3 Elasticity Pathway

While the elasticity space enables elasticity analysis, it does not provide insight into relationships between metrics influencing the elastic behavior over time. In order to characterize the elastic behavior of cloud application from specific views/perspectives, we define the concept **elasticity pathway**.
An elasticity pathway function must describe the evolution of an elastic cloud application over time. Such a function is necessary for evaluating the historic behavior of a cloud application, given a certain control strategy, with respect to user-defined requirements, as to understand if the control strategy was effective or not. Moreover, such a function gives a foundation for predicting the behavior of the cloud application, serving as a base for refining control strategies.
Given a specific view on metrics, $V = \{m_i, .., m_j\}$, an elasticity pathway for V characterizes the elasticity relationship among $m_i$over the time. Formally, an elasticity pathway is determined by a function which takes as input an elasticity space $ElSpace$ and a view V over the space's metrics, and returns another function describing

behavioral patterns or characteristics of the elastic cloud application. As the elasticity pathway function is applied over an elasticity space, the quality of the determined pathway is heavily influenced by the size and data in the elasticity space.
More concretely, an elasticity pathway could indicate that the response time of an application and its CPU usage are high in x% of the encountered situations, indicating there might be a correlation between them. More concrete details can be found captured in the description of the implemented elasticity pathway in the MELA-AnalysisService prototype, in Section 5.2.4.2.

4.4 Smart Elasticity-aware Deployment

4.4.1 Elasticity-aware Deployment Processes

The role of the smart elasticity–aware deployment is to provide smart deployment configurations based on elasticity characteristics described by the CELAR user in c-Eclipse. In case the CELAR user does not know exactly what amount of resources should be allocated for his/her application, s/he can provide simple information (e.g., application structure, what metrics are important for each application component or complex component), as well as more advanced information if available (e.g., hints on component load types), and the Decision Module outputs the configuration which is best for him/her, considering information coming from cloud providers as well as profiling information from the CELAR Profiler (described in Section 3.3.3 Deliverable D3.2 [D3.2]) . The simple description of the application, for the case the CELAR user is not that well acquainted with the application, could contain application structure at architecture level and include no information about underlying software configuration, this information being added by the smart deployment service.
The first step of the smart deployment is the *architecture refinement.* Normally, the application description will contain the full deployment topology which includes the application components, software dependencies, and cloud resources. In fact, some of these components can be discovered and generated at deployment time based on some preferences, such that the user does not need to specify the full deployment stack for all his/her components. For example, if an application is designed to run with different types of database systems, a particular database can be selected appropriately for a deployment instance.
The second step is *cloud resource configuration*. For a smart deployment we take into account the elasticity preferences for the deployment that can influence the elasticity capabilities of the application. Some preference dimensions are resources, the time in which the action is performed, the resizing types (e.g., horizontal or vertical) and other elasticity capabilities, the constraints or dependencies between elasticity capabilities. Based on these preferences, the deployment service will target to an optimization such as minimizing cost, maintain the quota of the resources.
Out of the two smart deployment steps above, the first one will be based on the knowledge regarding the dependency among cloud applications. The second one is more challenging, the following subsection showing an analysis of necessary steps in order to configure the application with the best cloud resources.

4.4.2 Cloud Resource Configuration

We refer to resources offered by IaaS provider together with their configuration as cloud units (e.g., a virtual machine flavor of a particular provider). For each application component, a resource configuration is the quantity of resources of different types that

need to be provisioned for deploying the component. We denote a resource configuration as $C = \{R_1, \cdots, R_n\}$, where n is the number of resource types.

4.4.2.1 Deployment Preferences and Application Knowledge

The CELAR user can specify through c-Eclipse information which can be regarded either as deployment preferences (e.g., I want this application component to have low cost because it is not that important) or as knowledge about the application –the knowledge about the application was referred in D1.1 [D1.1] as application hints (e.g., I know that my application component needs at least 5 GB of RAM to run). We enumerate the types of preferences and application knowledge description we take into consideration in generating the deployment configuration:

- *Workload type* (for application components, or composite components): `computation intensive`, `data intensive` or `network intensive`. This information is used by the smart deployment algorithm, which achieves a tradeoff between the different types of resources. For example if a component has computation intensive workload, cloud resources that have more CPU will be preferred when deploying the component.
- *Oscillation* (indicating how elastic the application is expected to be during runtime): $oscillation = \{amplitude, frequency\}$. The amplitude represents the number of cloud units which are expected to be requested in a short period and will be provisioned by the Resource Provisioner. The frequency is the estimated rate of elasticity actions per hour, or in other words, how frequent the application is scaled or reconfigured. In other hand, the frequency also shows how quick resource provisioning should be for resources associated with each application component. An amplitude close to zero indicates that the component will not request more resources during runtime, while a frequency close to zero indicates that the resources associated with the component are rarely changed.
- *Resource Preferences:* are boundaries over minimum and maximum resources needed by an application component. If this knowledge is not available, the minimum and maximum offered by cloud provider will be taken into considerations.

The above-mentioned information can also be extracted from the information provided by the CELAR Profiler, reducing the load on the user in terms of information specification.

4.4.2.2 Resource Configuration Mechanism

The objective of the smart deployment is to find an optimal resource configuration that achieves the highest performance and quality characteristics at the lowest cost.

In the extreme case (e.g., we either care about cost or performance), the best configuration for cost is consists of minimum resources required, $C^{min} = \{R_1^{min}, \cdots, R_n^{min}\}$, while the best configuration for performance is taking maximum resources: $C^{max} = \{R_1^{max}, \cdots, R_n^{max}\}$. Using these two cases, the search space for the deployment configuration is defined as $ES \subset (C^{min}, C^{max})$, the solution being a $C^{Solution} = \{\forall i \in \{1 \cdots n\} R_i | R_i \in (R_i^{min}, R_i^{max})\}$.

The resource configuration mechanism consists of several steps, taking into account the user preferences:

1. Analyzing the oscillation preferences $\{amplitude, frequency\}$, we find the configuration $C^{initial}$ for each application component where the cost is the smallest, and which accommodates the expected average amplitude of change

considering the frequency (e.g., if it takes 5 minutes to create a resource, and we might need 14 units/h, we need to consider a different resource).

2. Based on the workload types/hints, we refine the resulted configuration $C^{initial}$ by giving higher priority in detriment to the cost to the specific types of resources which characterize the workload type (e.g., set the configuration with more CPU if the workload for the respective component is computing intensive), and obtain $C^{solution}$.
3. Choosing and returning, from provider offerings, the cloud units closest to the configuration $C^{solution}$ resulted from step 2.

5. Decision Module Prototype V1

5.1 Decision Module Architecture Refinements

We refine the architecture described in D5.1 [D5.1] by introducing two different services, in addition to the existing rSYBL core service: MELA Analysis Service and SALSA Service. These two services are in charge with two major tasks, elasticity behavior analysis and cloud application deployment, and we see them as distinct tools which can be used outside of CELAR.

5.1.1 Decision Module Architecture

The Decision Module contains three main services:

- *rSYBL Service*, which is the Decision Module core
- *MELA Analysis Service*, which is in charge which analyzing monitoring data, detecting application Elasticity Space, Pathway and Dependencies
- *SALSA Service*, which is in charge with generating from a shallow application description given by CELAR user, a description of everything necessary for a full installation of the application, and the necessary resources for the application, considering the fact that the application would need to scale/reconfigure.

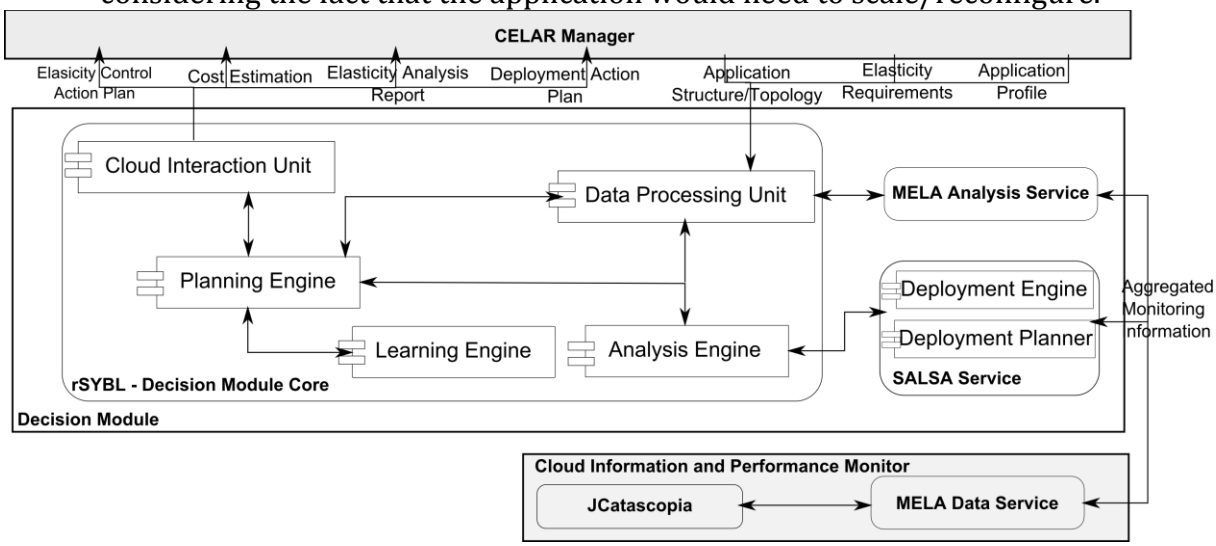

Figure 10: Decision Module Refined Architecture

These three Decision Module services are described in detail in Section 5.2, together with their implementation details and interactions.

5.1.2 Refined Interaction with other CELAR Modules

Figure 11 shows the sequence of steps necessary for generating a smart deployment description. The CELAR Manager notifies the Decision Module that a new application is to be deployed, sending it the description of the application. The description can be incomplete (containing only a part of the software to be installed on VMs, no scripts or instructions for installation) but it can also contain user's elasticity requirements, the application profile (if available) which is received by the CELAR Manager from the Profiler, and the elasticity capabilities to be integrated. Using this information, the Decision Module (in this case SALSA Service from the Decision Module) generates a deployment description.

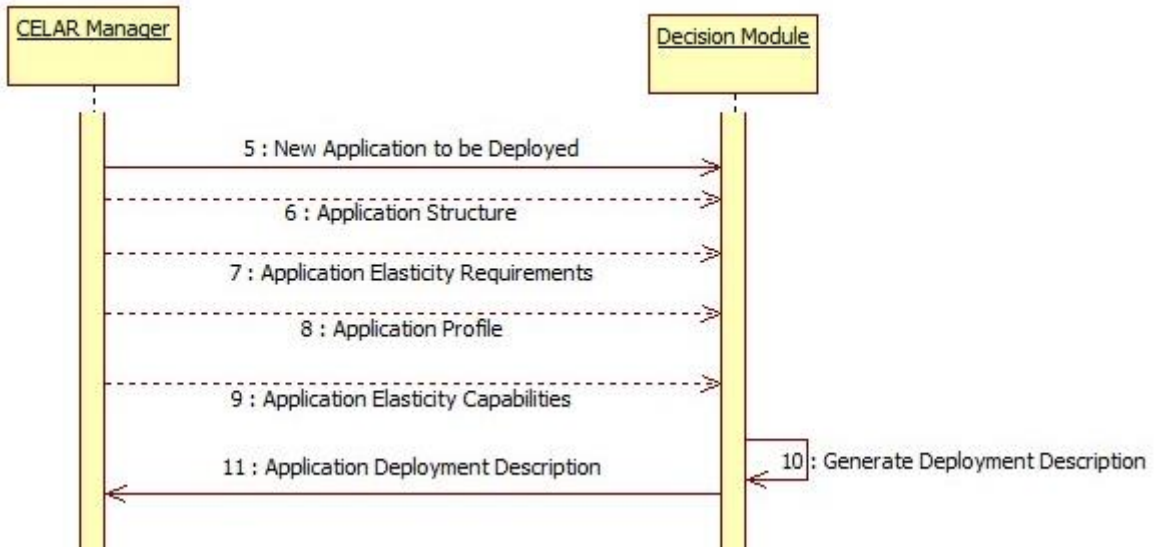

Figure 11: Smart Deployment Scenario

Figure 12 shows how, during runtime, the Decision Module continually analyses the state of the application with respect to requirements, and takes both corrective and preventive actions (steps 11-16).

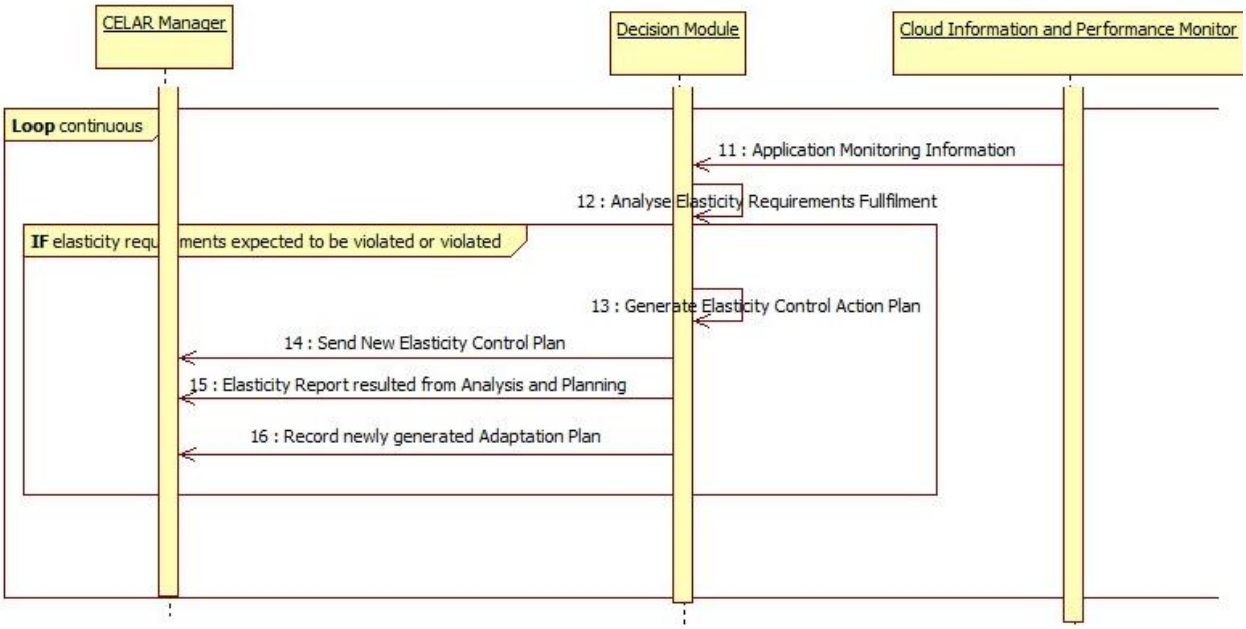

Figure 12: Elasticity Control at Runtime

5.2 Implementation

This section describes the three main services which compose the decision module. The source code is open-source, freely available in the CELAR GitHub repository https://github.com/CELAR/decision-module, and is a first-version of the Decision Module implementation. This repository hosts the CELAR-specific code, which depends on the code from TUWIEN GitHub repository https://github.com/tuwiendsg, that hosts all three services for being used together as well as separate, as result of the CELAR project.

A demonstration of rSYBL together with MELA was published in ICSOC proceedings [SYBL+MELA, 2013] and is available online[4].

5.2.1 Input from other CELAR Modules

Table 3presents the REST API calls exposed through the CELAR Monitoring System web service[5] to the Decision Module and any other interested CELAR modules.

Table 3: CELAR Monitoring System API Related to Decision Module

URI	Request Type	Parameters	Return Type	Description
/agents	GET	-	List of Agents in deployment [JSON Array]	Get Agents/VMs in current deployment
/{agentID}/availableMetrics	GET	-	List of metrics exposed by Agent [JSON Array]	Get available/offered metrics (and their metadata) for an Agent
/metrics	POST	comma separated list of metric IDs (in request body)	List of metrics with their timestamp and latest reported value [JSON Array]	Get latest metric values for a number of Metrics
/metrics/{metricID}	GET	-	Latest value for a specific metric [JSON]	Get latest value of a single metric
/metrics/{metricID}	GET	tstart, tend	List of values for a specific metric in the given timeframe [JSON Array]	Get values of a single metric for a time interval

Table 4 presents information that will be exposed through the CELAR Information System API to the Decision Module[6] and any other CELAR interested modules.

Table 4: CELAR Information System API Related to Decision Module

Provided Data	Parameters	Description
IaaS provider resource quotas	resource id	A list of quotas for each resource that the IaaS provider offers for all tenant groups
Resource information	resource id	All available information and metadata for a resource
Resource available resizing	resource id	Resizing actions that can be

[4] http://dsg.tuwien.ac.at/research/viecom/prototypes/demo/melaAndSYBLWMV.wmv

[5] The Monitoring System web service is deployed on the Application Orchestration VM for each application deployment. For more information please see D1.1 [D1.1] and D4.1[D4.1].

[6] The development of the Information System is still in its early stages. A complete and detailed description of its API will be presented in D4.2

actions		applied to a resource
User quota per resource	user id, resource id	Quota for given resource based on the user's tenant group
Available software resources	-	A list of the IaaS provider's available automatic software deployment scripts (if any)
Available software resources per category	category id	A list of the IaaS provider's available automatic software deployment scripts (if any) for a specific category (e.g. Databases)
Available virtual hardware resources	-	A list of the IaaS provider's available virtual hardware resources (e.g. network interfaces)
Resizing action history	user id, application id, tstart, tend	List of previously applied resizing actions (and metadata) for a specific timeframe
Available monitoring probes		A list of the IaaS provider's available monitoring probes (if any)

5.2.2 Output towards other CELAR Modules

The Decision Module outputs elasticity-related information for the current application, as well as elasticity control action plans and smart deployment configurations (see Table 5).

Table 5: Decision Module Information Sent to the other CELAR Modules

Provided Data	Towards	Description
Elasticity analysis report of the current application	CELAR Manager, to be sent to c-Eclipse	A list of statistical information regarding the application, starting from action effects on different components or composite components, to loads, elasticity of cost with respect to different quality parameters, etc.
Smart deployment description	CELAR Manager, to be sent to Resource Provisioner	Full TOSCA description of the application, with all the necessary software and resources configurations.
Elasticity control action plan	CELAR Manager, to be sent to Resource Provisioner	List of actions from the ones existing in the CELAR database, both application specific and cloud provider specific, to control the application elasticity in terms of cost, quality and resources.

5.2.3 rSYBL Service

rSYBL (runtime SYBL) is a framework for controlling the elasticity of cloud applications, considering user-formulated SYBL elasticity requirements. rSYBL takes as input the description of the application and of the available infrastructure providers, and provides runtime elasticity control. rSYBL is highly configurable, and can work with different monitoring solutions (e.g., Ganglia, MELA), and different enforcement platforms (e.g., CELAR Orchestrator, Flexiant FCO User API, OpenStack).

5.2.3.1 SYBL Elasticity Requirements Specification

For describing what types of elasticity controls could be required and the complexity of elasticity requirements, we examine various types of elasticity requirements from different stakeholders, at different granularities, presented in detail in [Copil, 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 [Moldovan, 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 allows the specification of elasticity requirements at three levels: service unit level for component related elasticity requirements, service topology for composite component related elasticity requirements and service level for application related elasticity requirements.

```
Constraint := constraintName : CONSTRAINT ComplexCondition
Monitoring := monitoringName : MONITORING varName=MetricFormula
Strategy := strategyName : STRATEGY WHEN ComplexCondition :action(parameterList)|
        strategyName : STRATEGY WAIT ComplexCondition|
        strategyName : STRATEGY STOP|
        strategyName : STRATEGY RESUME

MetricFormula := metric | number | metricFormula MathOperator metric | metricFormula
        MathOperator number
ComplexCondition := Condition | ComplexCondition BitwiseOperator Condition|
        (ComplexCondition BitwiseOperator Condition)
Condition := metric RelationOperator number| number RelationOperator metric |
        Violated(name)|Fulfilled(name)
MathOperator := + | - | * | /
BitwiseOperator := OR | AND | XOR | NOT
RelationOperator := <|>|>=|<=|==|!=
```

Figure 13: SYBL in BNF

5.2.3.1.1 Examples of SYBL Elasticity Requirements

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) 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 14 shows a constraint specified for the composite component with ID `WebServiceTopology`. The elasticity requirement sets the preferred response time below 450 ms. We define this elasticity requirement as a subtype of Java Annotation,

triggered at runtime when the annotated method is executed and caught and interpreted using AspectJ.

```
@SYBL_ServiceTopologyDirective(annotatedEntityID="
    MyServiceTopology", constraints="Co3:CONSTRAINT responseTime
    < 450 ms;")
```

Figure 14: Example of Elasticity Requirements in SYBL as Java Annotations

Figure 15 shows a strategy specified for the composite component with ID DataEndServiceTopology. The elasticity requirement is a conditional strategy, which enforces the action scalein for the composite component when both responseTime and the average throughput are above predefined values.

```
<SYBLSpecification id="DataEndServiceTopology" type="
    ServiceTopology">
    <Strategy Id="St1">
        <Condition>
            <BinaryRestriction Type="smallerThan">
                <LeftHandSide>
                    <Metric>throughputAverage</Metric>
                </LeftHandSide>
                <RightHandSide>
                    <Number>300</Number>
                </RightHandSide>
            </BinaryRestriction>
            <BinaryRestriction Type="smallerThan">
                <LeftHandSide>
                    <Metric>responseTime</Metric>
                </LeftHandSide>
                <RightHandSide>
                    <Number>360</Number>
                </RightHandSide>
            </BinaryRestriction>
        </Condition>
        <ToEnforce ActionName="scalein" />
    </Strategy>
</SYBLSpecification>
```

Figure 15: SYBL in XML

The constraint shown in Figure 16specifies that the cost for the `PilotCloudService` should be below 100$. The SYBL elasticity requirements can be easily integrated within TOSCA [TOSCA] policies, and interpreted by the elasticity controller. rSYBL understands two types of SYBL requirements injected into TOSCA: simple, as SYBL text (Figure 16) or with SYBL requirement in XML form (Figure 17).

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

Figure 16: Example of TOSCA description with SYBL Elasticity Requirements

```
<tosca:Policy type="ElasticityConstraint"
   id="CPU_Constraint">
   <tosca:Properties>
      <ElasticityConstraintProperties>
         <ToEnforce>
            <BinaryRestriction Type="LessThan">
               <LeftHandSide>
                  <Metric>CPU_Usage</Metric>
               </LeftHandSide>
               <RightHandSide>
                  <Number>80</Number>
               </RightHandSide>
            </BinaryRestriction>
         </ToEnforce>
      </ElasticityConstraintProperties>
   </tosca:Properties>
</tosca:PolicyTemplate>
```

Figure 17: Example of TOSCA Description with SYBL in XML Format

5.2.3.2 Multi-level Elasticity Control

Considering the model of the application described through the runtime dependency graph presented at the beginning of Section 5, we enable multiple levels elasticity control of the described application, based on the flow shown in Figure 18. 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.

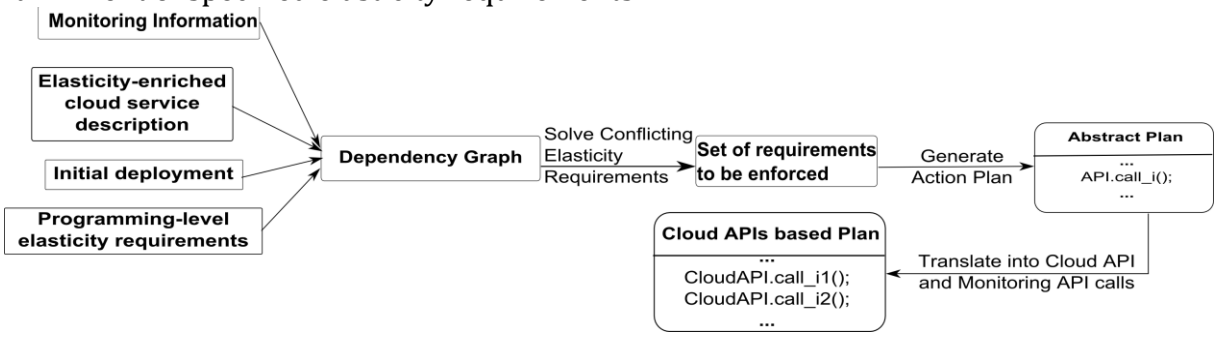

Figure 18: Flow of Elasticity Control

Let us consider a simple example shown in Figure 19 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 service topology, increasing the processing speed. An example of an action plan, shown in Figure 19 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 service unit on the new machine, and necessary calls for the new instance of the service unit to join the computation topology cluster.

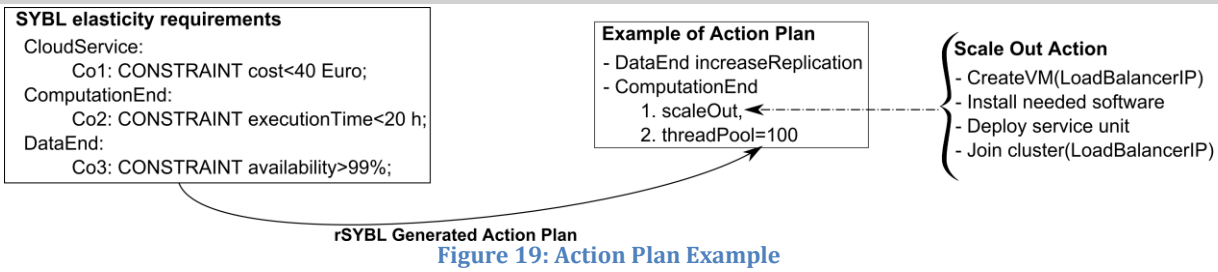

Figure 19: Action Plan Example

5.2.3.3 Configuring and Running rSYBL

A detailed description on how to configure and run rSYBL with an application, for controlling the application can be found on the repository's wiki https://github.com/tuwiendsg/rSYBL/wiki. In order to control the application with already monitoring tools and enforcement tools for which rSYBL already has plugins, the only needed configuration consists of selecting the proper tools through the configuration files, writing (if necessary) configuration data for them, and deploying rSYBL.

In case the user wants to use new metrics, s/he only needs to send this metric to MELA or Ganglia, and then describe requirements depending on it.

For the developer case, when we want to deploy rSYBL on a new infrastructure, with different monitoring data, plug-ins can be implemented easily, and if specified so, they will be loaded by rSYBL at runtime in order to be used. The necessary steps for developing new rSYBL plugins are described in chapter 3 of the user guide and of the wiki page.

5.2.3.4 rSYBL Service API

rSYBL is exposed as a RESTful service for sending the necessary information from c-Eclipse side, and for pushing out-of-the ordinary events (e.g., resources of a virtual machine being abnormally highly used) when discovered by the Cloud Information and Performance Monitor.

Table 6: rSYBL API

Nr.	RESTful API: /rsybl/restWs/				Description
	Type	Resource URL	Consumes	Produces	
1	PUT	/appDeploymentDescription	application/ xml	-	Setting the deployment topology, after the deployment was done as a "smart-deployment" with the help of SALSA or as described by the user
2	PUT	/setApplicationDescriptionInfoToscaBased or /appDescription	application/ xml	-	Call used to send the application description along with the userpolicies and strategies tothe Decision Module (information sent as TOSCA-first URL- or as CELAR-internally agreed representation-second URL)
3	PUT	/pushEvent	application/ xml	-	Sent by the Cloud Information and Performance Monitor in case out-of the ordinary events are detected

5.2.4 MELA-Analysis Service

The MELA-AnalysisService analyzes the elasticity of cloud applications, focusing on the three elasticity dimensions: Cost, Quality and Performance. MELA-AnalysisService provides elasticity analysis capabilities using data retrieved from the MELA-DataService belonging to the Cloud Information and Performance Monitor (D4.1), determining the elasticity boundaries, space and pathway of cloud application. This information is used by rSYBL (D5.1) in controlling the elasticity of such applications.

MELA-AnalysisService is implemented in Java as a module of MELA, located on GitHub at https://github.com/tuwiendsg/MELA. MELA is structured using Maven (http://maven.apache.org/) as a single .pom file with multiple modules. MELA-AnalysisService provides beside elasticity analysis functionality, a graphical representation of the elasticity analysis result implemented using D3_JS, and a RESTful API for configuration and retrieving the elasticity analysis results in a programmatic manner.

5.2.4.1 Elasticity Space Function

In the current prototype, the elasticity space boundaries are determined for all application components, complex 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-Analysis Service determines and continuously updates requirements for the rest of the application components, requirements then enforced by rSYBL.

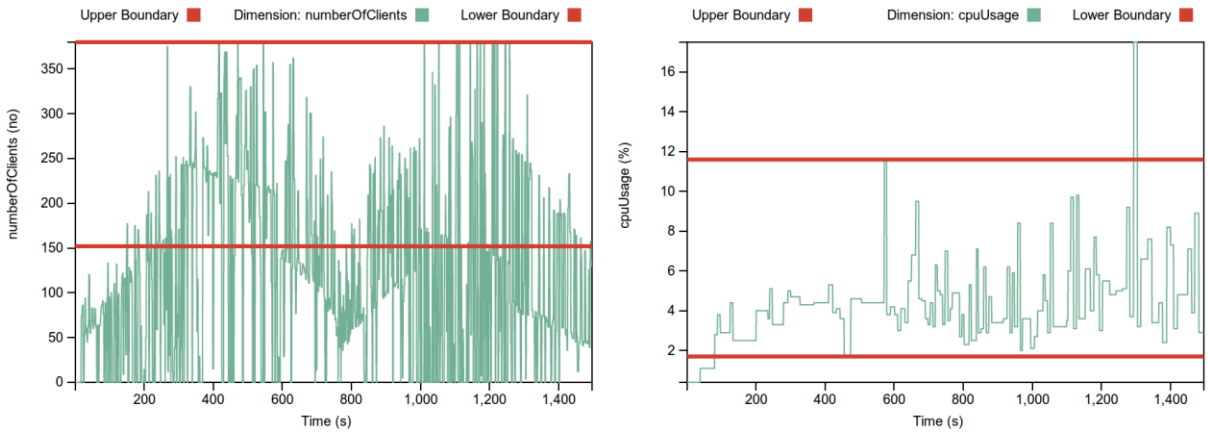

Figure 20: Application Elasticity Space and Boundary in Time Example

In Figure 20 we depict an example of an elasticity space for an application component. The space has two dimensions: `numberOfClients`, and `cpuUsage`. The space's elasticity boundaries are marked by red lines. rSYBL will enforce these elasticity boundaries through elasticity actions, thus maintaining the behavior of the cloud application in user-defined requirements.

5.2.4.2 Elasticity Pathway Function

The elasticity pathway function gives an indicator on the historical behavior of the cloud service 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-Analysis Service we adapt as elasticity pathway function, an unsupervised behavior learning technique using self-organizing maps (SOMs) presented by [Dean 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 respects user-defined elasticity requirements.

A self-organizing map (SOM) is an artificial neural network trained using unsupervised learning to produce a low-dimensional (typically two-dimensional) representation of the input space of the training samples, called a map. The components of SOM are called neurons. Associated with each neuron is a weight vector of the same dimension as the input data vectors and a position in the map. The usual arrangement of nodes is a two-dimensional regular spacing in a hexagonal or rectangular grid. The procedure for placing a vector from data space onto the map is to find the node with the closest (smallest distance metric) weight vector to the data space vector.

As SOMs are unsupervised neural networks that map multi-dimensional spaces into low dimensional ones, we use them for grouping monitoring snapshots. These characteristics make SOMs suitable for classifying monitored snapshots, as snapshots can contain many different metrics. Thus, we use as input data vector groups of metric values derived from the elasticity space of the elastic cloud application.

The value of each SOM's neuron is a group of metric values, derived from the monitored values extracted from elasticity space dimensions monitored snapshots. Each new metric values group is mapped to the neuron from which it has the smallest distance. For computing the distance, first the values of the elasticity space metrics are normalized. Then, a Euclidean distance is used to compute the distance of a new metric values groups to the neurons in the SOM. The determined closest neuron and its SOM neighbors are updated using the function:

$$V_{new}(group) = V_{old}(group) + A * N(group)(V(snapshot) - V_{old}(group))$$

, where A is a discount factor, and $N(group)$ is a neighbourhood function determining the degree with which a group value is updated.

In unsupervised learning the initialization of the system is important. As we classify groups after the number of snapshots mapped to it, we do not initialize the SOM with random values, as it might generate groups which are close together in value but far away in terms of location in the SOM. In such a case similar snapshots might be assigned to separate groups, diluting the number of snapshot mapped to a SOM entry. Thus, we initialize the SOM with snapshot groups having all metrics equal to 0, and rely on its self-adaptive nature to map the input data.

We use a neighborhood function of 1 for the directly targeted group and of $\frac{1}{neighborLevel * neighborsCount}$ for the group neighbors. Updating the neighbors creates new groups, mapping the input data better. The discount learning factor is $\frac{1}{neighborLevel}$, the neighborhood is 2 and the map size is $10x10$.

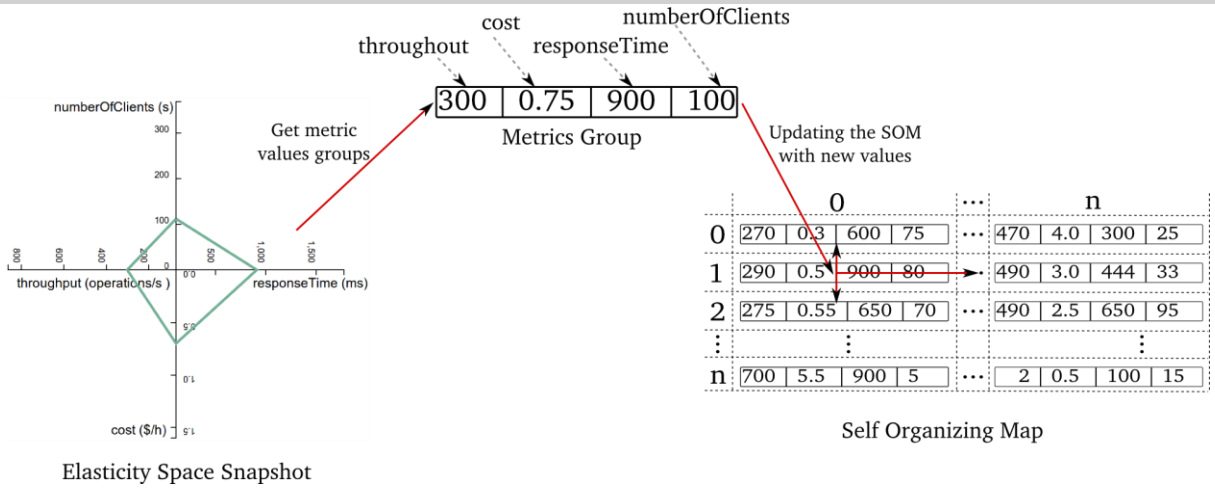

Figure 21: Constructing and updating SOM with metric values

In Figure 21 we depict the process of extracting groups of metric values from the elasticity space, and using those groups to train the SOM. After the SOM is trained with all monitored data extracted from the elasticity, a filtering step merges groups with same value, consolidating the monitored snapshots. Similar monitoring snapshot groups are extracted and the average absolute deviation (the average of the absolute deviations) of the number of snapshots mapped per group is computed. Using the average absolute deviation (AAD), a group is RARE if its deviation is negative and its absolute is higher than the AAD, DOMINANT if its deviation is higher than the AAD, and NON-DOMINANT otherwise. While finer grained classes can be defined, we argue that these three are enough to give insight in the elastic behavior of cloud applications.

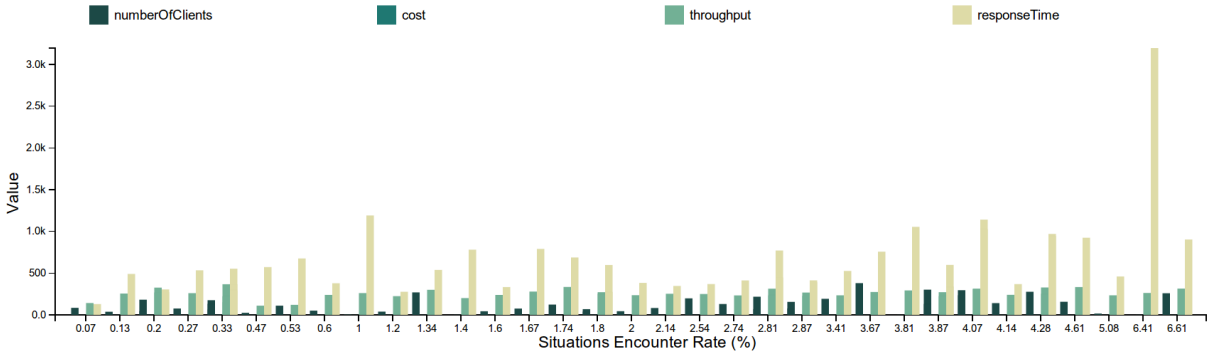

Figure 22: Application Elasticity Pathway Example

In Figure 22 we show an example of determined elasticity pathway for the elasticity space example presented before. The pathway example shows the result of updating the SOM, each situation group having a Situation Encounter Rate which indicates how often value pairs similar to the SOM nodes were encountered. For example, from this pathway we can see that a combination of very high response time, low number of clients and low throughput is encountered in approximately 6.41% of the situations, indicating that the application does not behave in an elastic manner under all circumstances.

Correlated with the elasticity boundaries, the elasticity pathway gives insight in how much of the time the application behavior is within the boundaries, and what is the correlation between the different metrics in the elasticity space. This insight is crucial in understanding the behavior of the application in time. Moreover, from the determined metric values situations, we can see if and how metrics are correlated, which we can use

in predicting the application behavior and refining control strategies. For example, if the user requirements state a response time of "x" and a throughput of "y", but the pathway states that only in 30% of the situations such values are encountered together, we can deduce that the given requirements can't be fully fulfilled.

5.2.4.3 MELA Analysis Service API

MELA provides RESTful API both for application specific configuration such as application structure, user elasticity requirements, metrics composition rules submission, and for elasticity analysis such as elasticity boundary, space and pathway computation.

Table 7: MELA-Analysis Service API

#	RESTful API: /MELA-AnalysisService/REST_WS/				Description
	Type	Resource URL	Consumes	Produces	
1	POST	/elasticitypathway	application /xml	application /json	Returns the elasticity pathway for a supplied application component
2	POST	/elasticitypathwayxml	application /xml	application /xml	Same as above, but returns XML
3	POST	/elasticityspace	application /xml	application /json	Returns the elasticity space for a supplied Monitored Element (application component)
4	POST	/elasticityspacexml	application /xml	application /xml	Same as above, but returns XML
5	POST	/elasticityspacecompletexml	application /xml	application /xml	Same as above, but elasticity space also contains recorded monitored data
6	PUT	/metricscompositionrules	application /xml	-	Submits metric composition rules to MELA, for structuring monitoring information
7	PUT	/ servicedescription	application /xml	-	Submits the application structure
8	POST	/ servicedescription	application /xml	-	Updates the application structure after elasticity actions enforcement
9	PUT	/servicerequirements	application /xml	-	Submits the application requirement based on which the elasticity space is determined
10	GET	/metrics	String	application /xml	Retrieves the metrics currently being collected for a Monitored Element ID (application component)
14	GET	/servicestructure	-	application /xml	Retrieves latest application structure
17	POST	/addexecutingactions	application /xml	-	Notifies MELA that an elasticity action is being enforced
18	POST	/removeexecutingactions	application /xml	-	Notifies MELA that an elasticity action enforcement has completed

5.2.5 SALSA Service

SALSA is a service generating smart deployment configurations, by analyzing elasticity requirements for deployment. SALSA takes a shallow application description, application profiling information, and available cloud description information and generates a deployment configuration.

Figure 23depicts 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 rSYBL when a new application deployment or a re-deployment of running components is needed.

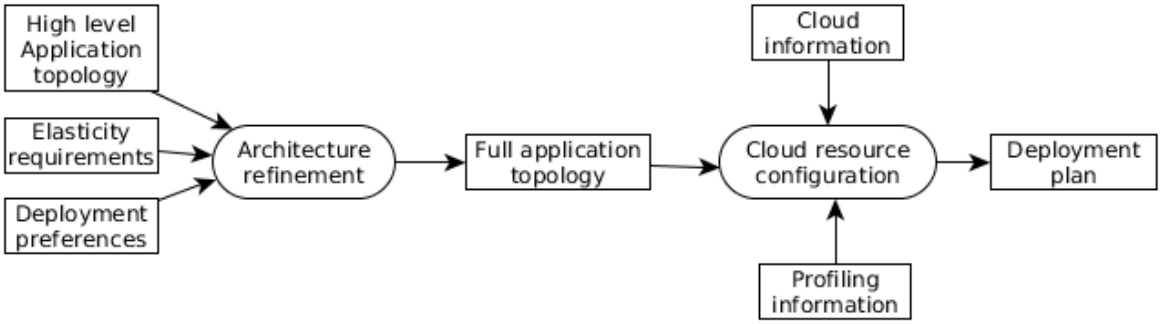

Figure 23: Flow of Smart Deployment

The input for SALSA Service is a TOSCA document which includes a high level application topology, elasticity requirements and a deployment preference. SALSA processes the input via `Architecture refinement` and `Cloud resource configuration` modules, and produces a deployment configuration and sends it to rSYBL which forwards it to the CELAR Manager.

The high level application topology describes components in a more abstract way. For instance, cloud resources can be represented by generic terms 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. The elasticity requirements for deployment are high level demands described in application topology at deployment time and affect the deployment configurations. These requirements are provided in TOSCA with the application topology. For each deployment, the CELAR user can provide a deployment preference which guides SALSA in using the strategies to process requirements (e.g., optimize cost, maximize resource usage, or maximize latency).

The architecture refinement aims to enrich the application topology with deployment components in which the performance and cost of the application can be optimized. SALSA maintains a repository of existing components for the refining process. The output of this step is a full application topology with the particular components.

In the case the full application topology lacks of cloud resources description, SALSA Service does the placement of the needed components. The analysis process takes available cloud information and profiling information and generates the best configuration. The CELAR user can guide which deployment algorithm is used by defining his/her deployment preference.

5.2.5.1 SALSA Service API

The SALSA Service is located on GitHub on https://github.com/tuwiendsg/SALSA. The SALSA exposes a set of APIs to retrieve deployment-related information from rSYBL, which receives them from the CELAR Manager.

Table 8: SALSA API

Nr.	RESTful API: /salsa-center-services/rest/				Description
	Type	Resource URL	Consumes	Produces	
1	PUT	/submitDescription	application /xml	-	Submit new application structure which is a high-level TOSCA description.
2	GET	/getDeploymentDescription		application /xml	Get the full description of cloud application deployment.
3	GET	/getElasticityCapabilities		application /xml	Return the list of available elasticity capabilities for the current application.
4	PUT	/setCloudInformation	application /xml	-	Set the cloud information of specific cloud providers.
5	PUT	/setProfilingInformation	application /xml	-	Set the profiling information for analyzing.

5.2.6 Interactions among Decision Module Services

5.2.6.1 rSYBL-MELA Interaction

The interaction between rSYBL and MELA is shown through the sequence diagram in Figure 24. At initialization of the Elasticity Provisioning Platform MELA receives from rSYBL through the RESTful API of the MELA-AnalysisService the structure of the application to be controlled, metrics composition rules and elasticity requirements. The application structure and metric composition rules are forwarded to the MELA-DataService, which uses them to structure and enrich data collected from JCatascopia monitoring system implementation (see D4.1 [D4.1]) using the JCatascopia pool and push metric collection mechanisms. The elasticity requirements are used internally by the MELA-AnalysisService to determine the elasticity space and pathway of the.

rSYBL periodically requests from MELA-AnalysisService structured and enriched monitoring data, and updated elasticity space and pathway. Based on this, the behavior of the managed cloud application is analyzed, and, if necessary, elasticity actions are generated. After enforcing elasticity actions which change the application structure, rSYBL informs MELA about the updated application structure through the MELA-AnalysisService. Moreover, a modified or new set of metric composition rules and elasticity requirements can be submitted by rSYBL to MELA-AnalysisService, if this is needed.

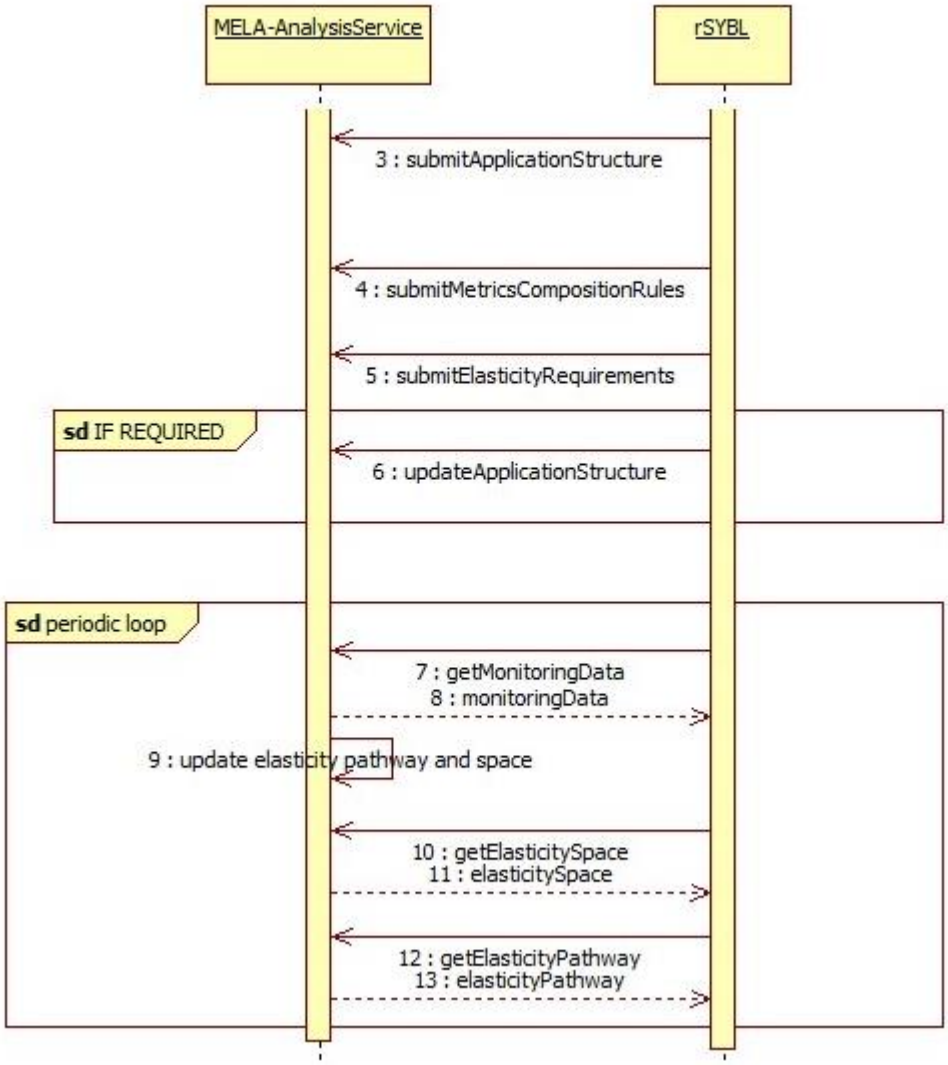

Figure 24: MELA-Analysis ServiceInteraction with rSYBL

5.2.6.1 rSYBL-SALSA Service Interaction

SALSA Service provides a service for generating complete description of TOSCA, with all the necessary information, including initial resources to be allocated, software to be installed, configuration of that software if it is not application specific, for the case the CELAR user does not know all the necessary details for deploying the application.

In that case, the application is described in terms of needed software (without its dependencies), and of the elasticity requirements, it is profiled to check how the application would behave under different resources configuration, and all this information is analyzed by the SALSA Service in order to generate a full TOSCA deployment description. Figure 25 shows the described interaction in steps 2-5.

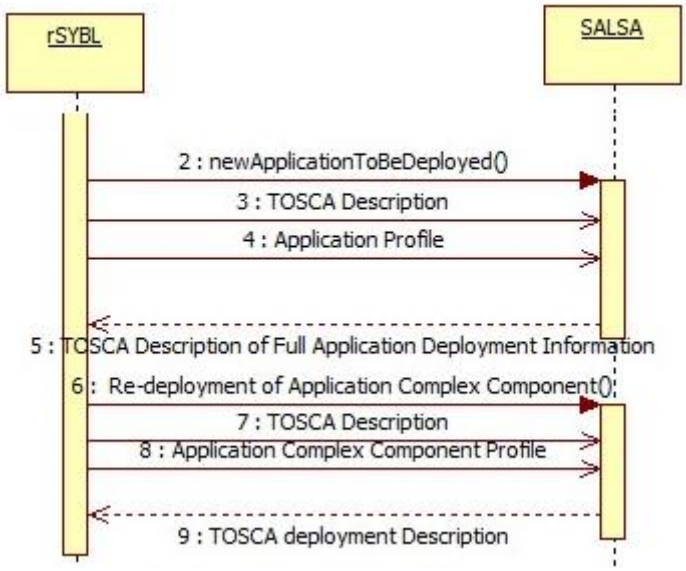

Figure 25: SALSA Interaction with rSYBL

For the case the rSYBL, the core part of the Decision Module, needs to re-deploy parts of application (e.g., for error cases), SALSA Service is called to give a deployment configuration. Figure 25 shows this case, in steps 6-9, in a similar fashion with full application deployment, with the exception that in this case the composite component needs to be configured in order for it to connect to the existing running part of the application.

5.3 Decision Module Evaluation

For evaluating how the Decision Module V1 can control applications at the moment, we use the Machine-to-Machine (M2M) DaaS Application, also used in Section 4.2 and 5.2.4, 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 composite components, an `Event Processing Service Topology` and a `Data End Service Topology`. 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 Service Topology`, and stored/retrieved from the `Data End Service Topology`.

When focusing on the `Event Processing Service Topology`metrics, number of clients, cost and response time, we can see that the cost is affected linearly in this case by the control actions enforced, since it is computed solely depending on the number of virtual machines.

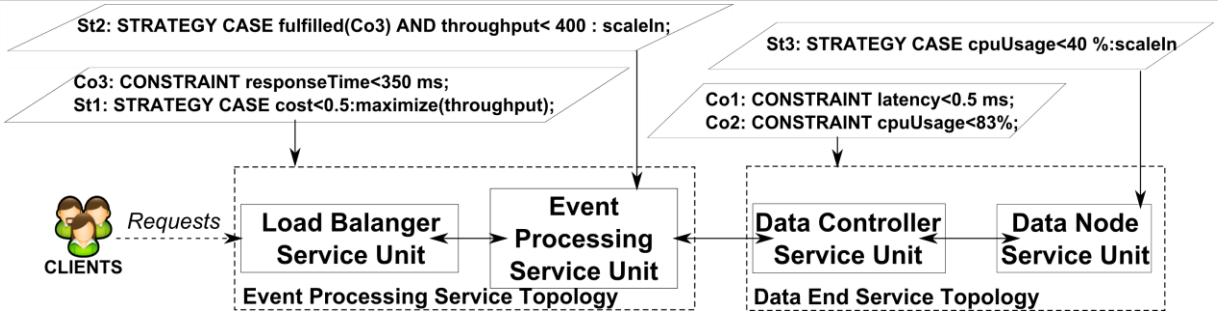

Figure 26: Application Used for Evaluation

In Figure 27, we can see that the number of VMs which is affected by the elasticity control, and varies with the number of clients although nowhere in the requirements is mentioned such a demand. Due to these scaling actions, the response time is able to stay within the required boundaries, at a relatively stable value without increasing more than acceptable for a too high period.

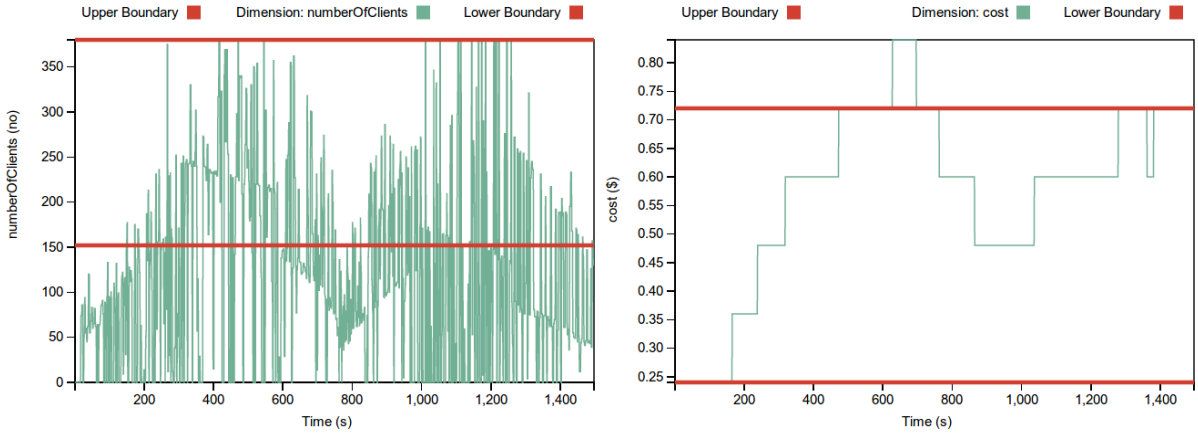

Figure 27: Elasticity Control shown at Event Processing Service Topology

As at current moment's implementation we are introducing action effects manually from observations, there is still place for improvement in the quality of our decisions, reasons for which we will integrate in our next version of the prototype research presented in Sections 3 and 4. Moreover, we aim at improving our decision mechanisms through taking not only on a reactive basis as it is the case now, but also predictive actions.

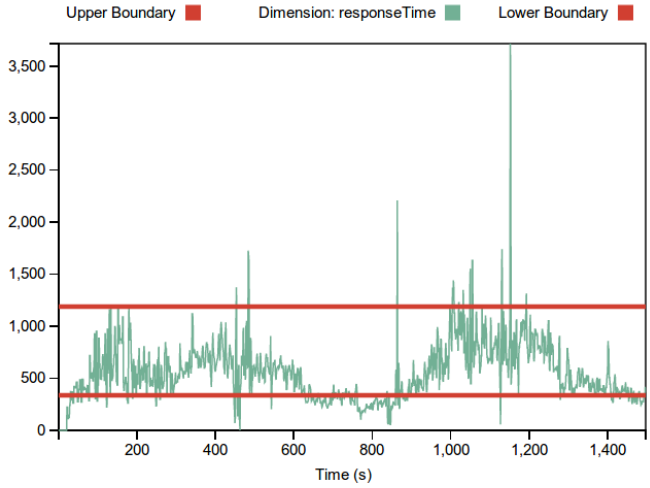

Figure 28: Response Time for Event Processing Service Topology

5.4 Next Steps towards Decision Making Module V2

In what follows, we will integrate ongoing research from Section 3 and 4 to improve the current Decision Module prototype.

The current version of the Decision Module implementation does not contain a fully implemented and tested learning engine on the basis of which decisions are taken while considering a clear effect (currently, we manually introduce the effects of each control action). For this, the implementation of Section 4.2 research, together with further research on elasticity metrics dependencies on MELA-Analysis Service will be integrated into Decision Module V2.

Moreover, the SALSA smart deployment service will be improved to provide decisions taking into account the expected elasticity of the application (e.g., taking into account what actions one can take on a provider or other), or the budget which is being allocated.

6. Conclusions

In this document we have provided a detailed description of ongoing research involved in the Decision Module of CELAR Platform, and of the progress made with the implementation of the Decision Module prototype. The research described in Section 3 and 4 will be integrated within the Decision Module implementation, which will be measured through milestones MS12 and MS13, and reported in our next deliverable D5.3.

References

[Ananthanarayanan, 2011] G. Ananthanarayanan et al. Scarlett: "Coping with Skewed Content Popularity in MapReduce Clusters". In *Proceedings of the sixth conference on Computer systems* (EuroSys '11). ACM, New York, NY, USA, 287-300.

[Aspnes, 2004] J. Aspnes, J. Kirsch, A. Krishnamurthy. „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.

[AutoScaling] Amazon AutoScaling, http://aws.amazon.com/autoscaling/

[AzureWatch]Paraleap AzureWatch, https://www.paraleap.com/AzureWatch/

[Bharambe, 2004] A. R. Bharambe, M. Agrawal, S. Seshan. "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.

[Cassandra, 2013] http://wiki.apache.org/cassandra/Operations#Load_balancing

[CloudSuite] "CloudSuite: A benchmark suite for emerging scale-out applications", http://parsa.epfl.ch/cloudsuite/cloudsuite.html

[Copil, 2013a] G. Copil, D. Moldovan, H.-L. Truong, and S. Dustdar. "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.http://doi.ieeecomputersociety.org/10.1109/CCGrid.2013.42

[Copil, 2013b] G. Copil, D. Moldovan, H.-L. Truong, S. Dustdar, "SYBL+MELA: Specifying, Monitoring, and Controlling Elasticity of Cloud Services", the 11th International Conference on Service Oriented Computing. Berlin, Germany, on 2-5 December, 2013.http://dx.doi.org/10.1007/978-3-642-45005-1

[Copil, 2014a] G. Copil, D. Moldovan, H.-L. Truong, S. Dustdar, I. Giannakopoulos, N. Papailiou, D. Tsoumakos, D. Trihinas, G. Pallis, M. Dikaiakos, "Elasticity behavior of cloud services", March 2014, under submission.

[Copil, 2014b] G. Copil,D. Moldovan, H.-L. Truong, S. Dustdar, "Multi-level Elasticity Control of Cloud Services with SYBL", March 2014, under submission.

[D2.1] Application Description Tool V1, Deliverable, CELAR Project, https://wiki.celarcloud.eu/lib/exe/fetch.php?media=CELAR_Project:Deliverables:Deliverables_2.X:CELAR_D2.1_finalrelease.pdf

[D3.2] Elasticity Provisioning Platform V1, Deliverable, CELAR Project, https://wiki.celarcloud.eu/doku.php?id=CELAR_Project:Deliverables:Deliverables_3.X:D3.2

[D4.1] Cloud Monitoring Tool V1, Deliverable, CELAR Project https://wiki.celarcloud.eu/lib/exe/fetch.php?media=CELAR_Project:Deliverables:Deliverables_4.X:CELAR_D4.1_finalrelease.pdf

[D5.1] Decision Process for On-demand Elasticity Report, Deliverable, CELAR Project, https://wiki.celarcloud.eu/lib/exe/fetch.php?media=CELAR_Project:Deliverables:Deliverables_5.X:celar_d5.1_v1.1.pdf

[D7.1] Cloud Policy Game Design Document, Deliverable, CELAR Project https://wiki.celarcloud.eu/lib/exe/fetch.php?media=CELAR_Project:Deliverables:Deliverables_7.X:celar_d7.1_finalrelease_v1.docx

[D8.1]Translational Cancer Detection Pipeline Design, Deliverable, CELAR Project, https://wiki.celarcloud.eu/lib/exe/fetch.php?media=CELAR_Project:Deliverables:Deliverables_8.X:celar_d8.1_draft_v1.0.docx

[DeCandia, 2007] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, W. Vogels. 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

[Fehling, 2012] C. Fehling, T. Ewald, F. Leymann, M. Pauly, J. Rutschlin, D. Schumm, "Capturing Cloud Computing Knowledge and Experience in Patterns," *International Conference on Cloud Computing (CLOUD), 2012 IEEE 5th*, vol., no., pp.726,733, 24-29 June 2012 http://dx.doi.org/10.1109/CLOUD.2012.124

[Folkerts, 2012], E. Folkerts, A. Alexandrov, K.Sachs, A. Iosup, V. Markl and C. Tosum, "Benchmarking in the Cloud: What it Should, Can, and Cannot Be", In 4th TPC Technology Conference on Performance Evaluation & Benchmarking (TPCTC 2012), held in conjunction with VLDB, Istanbul, Turkey, Aug 2012.

[Gambi, 2013] A. Gambi, D. Moldovan, G. Copil, H.-L. Truong, S. Dustdar. "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

[Ganesan, 2004] Prasanna Ganesan, Mayank Bawa, and Hector Garcia-Molina. "Online Balancing of Range-Partitioned Data with Applications to Peer-to-Peer Systems". In *Proceedings of the thirtieth International Conference on Very Large Data Bases* - Volume 30, VLDB '04, pages 444–455, Toronto, Canada, 2004. VLDB Endowment.

[Herbst, 2013] N. R. Herbst, S. Kounev, and R. Reussner, "Elasticity in Cloud Computing: What It Is, and What It Is Not," in *Presented as part of the 10th International Conference on Autonomic Computing*, 2013, pp. 23–27

[Islam, 2012] S. Islam, K. Lee, A. Fekete, and A. Liu, "How a consumer can measure elasticity for cloud platforms," in *ICPE*, 2012, pp. 85–96.

[Karger, 1997] D. Karger, E. Lehman, T. Leighton, R. Panigrahy, M. Levine, D. Lewin. 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, 2004] D. Karger, M. Ruhl, „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, 2011] I.Konstantinou, D. Tsoumakos, I. Mytilinis, N. Koziris. „Fast and cost-effective online load-balancing in distributed range-queriable system". *IEEE Transactions on Parallel and Distributed Systems*, 22(8), 1350-1364.

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

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

[MalStone] "MalGen and MalStone ", Open Cloud Consortium, https://code.google.com/p/malgen/

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

[Moldovan, 2014] D. Moldovan, G. Copil, H.-L. Truong, S. Dustdar, "Monitoring and Analysing Elasticity Space of Cloud Services", February 2014, under submission.

[MS11] Milestone 11: Decision Making Component V1, Milestone, CELAR Project, https://wiki.celarcloud.eu/doku.php?id=CELAR_Project:Milestones:WP5_milestones:MS11

[NoSQL Cassandra] http://blog.octo.com/en/nosql-lets-play-with-cassandra-part-13/

[Olio] "Olio Web2.0 Benchmark suite", http://incubator.apache.org/olio/

[OSGCloud 2012], SPEC Open Systems Group – Cloud Computing Work Group, "Report on Cloud Computing to the OSG Steering Committee", 2012.

[RightScale] RightScale, http://www.rightscale.com/

[SPEC OSGCloud] SPEC Open Systems Group (OSG) Cloud Subcommittee, https://www.spec.org/osgcloud/

[Sutter, 2012] Herb Sutter, "Welcome to the Parallel Jungle!", http://www.drdobbs.com/parallel/welcome-to-the-parallel-jungle/232400273

[SYBL+MELA, 2013] Copil, G., Moldovan, D., Truong, H.-L., Dustdar, S.: SYBL+MELA: Specifying, Monitoring, and Controlling Elasticity of Cloud Services. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) ICSOC 2013. LNCS, vol. 8274, pp. 429–436. Springer, Heidelberg (2013)

[TOSCA Policies] "Policies in TOSCA", http://docs.oasis-open.org/tosca/TOSCA/v1.0/cs01/TOSCA-v1.0-cs01.html

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

[YCSB] "Yahoo Cloud Benchmarking Service", http://research.yahoo.com/Web_Information_Management/YCSB/

[Verma, 2010] A. Verma, G. Kumar, R. Koller. „The cost of reconfiguration in a cloud". In *Proceedings of the 11th International Middleware Conference Industrial track* (Middleware Industrial Track '10). ACM, New York, NY, USA, 11-16. DOI=10.1145/1891719.1891721 http://doi.acm.org/10.1145/1891719.1891721

[Wang, 2013] W. Wang, B. Li, B. Liang, „To Reserve or Not to Reserve: Optimal Online Multi-Instance Acquisition in IaaS Clouds", 10th International Conference on Autonomic Computing, 2013

[Zhang, 2013] L. Zhang, X. Meng, S. Meng, J. Tan, „K-Scope: Online Performance Tracking for Dynamic Cloud Applications", 10th International Conference on Autonomic Computing, 2013