Georgiana Copil*, Daniel Moldovan, Hong-Linh Truong, and Schahram Dustdar

Continuous elasticity: Design and operation of elastic systems

DOI 10.1515/itit-2016-0020
Received May 2, 2016; revised August 24, 2016; accepted September 2, 2016

Abstract: Advancements in the areas of Cloud Computing, Internet of Things (IoT), and hybrid Human-Computer systems have made feasible the creation of a highly integrated human-machine world. The concept of elasticity plays a crucial role in fulfilling this vision, enabling systems to address various requirements reflecting performance, security, and business concerns. However, elastic systems are still in their inception, and numerous challenges need to be addressed in their development and management.

In this article we present an overview of our experience on elastic systems, with a focus on elastic cloud systems. In the quest for designing and managing elastic systems, several challenges need to be addressed, such as: (i) enabling the systems to fulfill different requirements from multiple involved stakeholders, (ii) designing elastic systems considering various degrees of elasticity capabilities provided by different technologies and environments, (iii) understanding behavioral relationships in elastic systems, and their effects on stakeholder requirements, (iv) monitoring costs and analyzing cost efficiency of elastic systems, (v) controlling the elasticity of such systems at runtime in order to fulfill stakeholders' requirements, and (vi) supporting system elasticity through operations management. We present the techniques we have adopted in order to tackle the above challenges. We introduce our solution for creating elastic systems, following their complete lifecycle, from design-time to operations management.

Keywords: Elasticity, design, operation, cloud computing, elastic systems.

***Corresponding author: Georgiana Copil,** TU Wien, Distributed Systems Group, Information Systems Institute, 1040 Vienna, Austria, e-mail: e.copil@dsg.tuwien.ac.at
Daniel Moldovan, Hong-Linh Truong, Schahram Dustdar: TU Wien, Distributed Systems Group, Information Systems Institute, 1040 Vienna, Austria

ACM CCS: Computer systems organization → Architectures → Distributed architectures → Cloud computing, Software and its engineering → Software creation and management → Designing software → Requirements analysis, Software and its engineering → Software creation and management → Designing software → Software design engineering, Computing methodologies → Artificial intelligence → Planning and scheduling → Planning with abstraction and generalization, Computing methodologies → Artificial intelligence → Planning and scheduling → Planning for deterministic actions

1 Introduction

Current technologies and trends facilitate the rapid development and operation of systems that are more and more integrated in our daily lives. Today's scenery is quite complex: devices are connected through the Internet, able to serve our needs; we are able to use software, and computing resources from the clouds; and we collaborate with computing systems, changing the roles we were accustomed with (e.g., Jennifer voice picking system [31]). Systems belonging to each of these domains, as well as cross-domain systems, are aiming towards a high degree of automation, and high adaptation rate to changes in the environment. Elasticity in software systems means that these systems are able to adapt and trade-off between their resources, cost, and quality.

Elasticity is consists of inherent dynamism, replaceability, heterogeneity, and compliance to business requirements. In order to develop elastic systems, elasticity needs to be kept in mind from design-time to operation-time. Modifying the system's behavior during operation means that the system provides interfaces with the outside world, and allows such changes. For instance, decreasing the amount of resources in a data cluster requires software interface enabling to copy the data from the resources to be removed/deallocated. This point holds as well for managing groups of workers in crowdsourcing marketplaces, where using APIs, the employer, or software programs on behalf of the employer, can decide who to add/remove or replace from its group of workers. Next, for understanding how the system operates, proper metrics and collection

mechanisms need to be designed. During system operation, the *complexity* of the system's structure, of resources used, of relationships and correlations existent in the system and in its environment, make the behavior assessment and control quite challenging.

In this paper, we introduce *a methodology for developing elastic systems*. We show how we address the issues described above, through tackling the following design and operation challenges: (i) designing cloud-native elastic systems profiting from cloud services heterogeneity, (ii) expressing requirements concerning desired systems behavior, (iii) understanding system behavior, including behavioral limits in relation to requirements, relationships among measured metrics, and expected behavior (iv) controlling the elastic system, based on all the knowledge accumulated through addressing challenges i–iv, and adapting operations management to the needs of elastic systems. We present our solutions addressing each of these challenges, and discuss the possibilities of developing cross-domain elastic systems, perfectly integrated into our lives.

This paper presents a holistic view on designing and managing elastic systems. Based on our work done in recent years [5–8, 24–27], we show how it all fits together in a complete approach for supporting the lifecycle of elastic systems. Building on previous work, in this paper we are proposing a novel methodology for developing elastic systems, and centered around the concepts of continuous design and operation for elastic systems.

The structure of the paper is as follows. Section 2 presents the overall methodology employed. Section 3 presents the scenario we will use in this paper. Section 4 presents challenges addressed and techniques used for designing elastic systems, Section 5 presents techniques for understanding and assessing elastic systems' behavior, and Section 6 details techniques used for managing elastic systems. Section 8 discusses envisioned challenges for developing and operating elastic systems built of cloud services, humans, and cyber-physical devices. Section 9 concludes the paper.

2 Methodology

For building elastic systems, stakeholders can use heterogeneous units of functionality offered as a service, or as third party components. Normally, the heterogeneous units of functionality can be anything from cloud services (e.g., virtual resources, platforms, or services) to connected devices (i.e., cyber-physical systems) and to employees part of crowdsourcing teams, that offer their services through well-defined interfaces. They can have diverse control capabilities, expose different metrics, and exhibit distinctive behavior. A new type of systems that can be built on top of them, which can profit from the possibility to understand and control the underlying units. We call these new type of systems *Elastic Systems*. In this paper we focus on cloud-based elastic systems, and discuss methods necessary for managing them from design to operation time.

The aspects of *dynamism*, *replaceability*, and *business requirements* should be considered both at design and operation time. Figure 1 shows the two main phases we see as essential for elasticity. The part on the top shows methods used for designing the system's elasticity. After developing the system artifacts, mechanisms for enforcing system's elasticity capabilities must be developed, along with mechanisms for collecting system's metrics. Further, these should be exposed through well-defined interfaces, to be used by automated or semi-automated controllers and/or operations managers. On the basis of this, the next step in the system design is to define high-level system requirements specified by various stakeholders (detailed in Section 4). System operation steps are shown at the bottom of Figure 1. For achieving elasticity, the system has to be monitored and understood through (detailed in

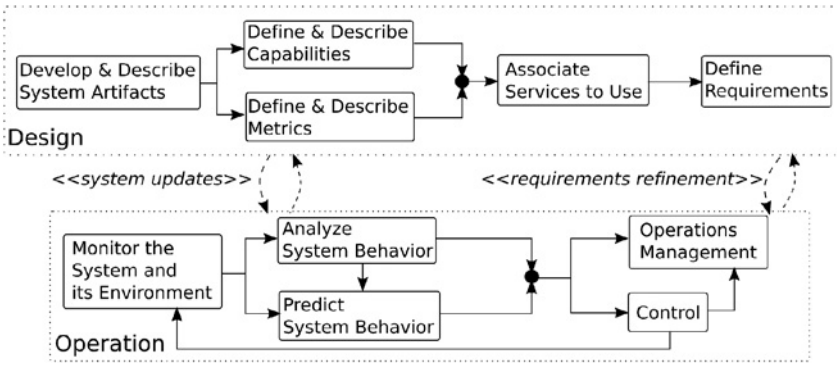

Figure 1: Steps in the design and operation of elastic systems.

Section 5): (i) collecting metrics concerning it, (ii) analyzing its behavior, (iii) predicting its behavior. Based on this, automated or semi-automated controllers can manage the behavior of systems and turn them into elastic systems (detailed in Section 6). Having this knowledge, it is very important to revisit the system design decisions (e.g., capabilities available for a unit, or configuration options), and refine requirements, introduce new capabilities, metrics, or even change the system architecture when bottlenecks are identified (e.g., a system unit that is not scalable may reduce system's performance, irrespectively of other units' elasticity). This continuous refinement (detailed in Section 8) through system updates or requirements refinement connects the two main phases of elasticity depicted in Figure 1.

We identify the following research questions that will be answered in the next sections:

1. What are the necessary methods for designing cloud-based elastic systems?
2. What aspects of systems' behavior need to be monitored, and how do we understand elastic systems behavior?
3. How should such an elastic system be managed, based on the obtained information (i.e., design, needs/requirements, knowledge on behavior)?

3 Model and scenario

3.1 Elastic systems model

For any system to become elastic, its structure, what can be monitored, how can it be changed, which are the virtual resources or cloud services it is using, and what are their characteristics are very important. Figure 2 shows the three types of information that we model: (i) structural information, (ii) infrastructure system information, and (iii) elasticity information. We consider that the system has as basic blocks system units, which can be any software components. System topologies are groups of units that are semantically connected (e.g., database clusters). At runtime, these structural units have associated infrastructure information, composed of virtual resources, and relationships among them (e.g., containers, artifacts, virtual machine). Both virtual resources and system units have associated elasticity information, which is necessary for achieving the desired elasticity for the system. Elasticity information is composed of capabilities (i.e., how the system units and virtual resources can change at runtime), metrics (i.e., measures of the system's behavior), and requirements (i.e., desirable behavior specified by stakeholders).

3.2 Scenario

Let us consider an IoT system, which collects data from devices, and serves interested users querying requests. We have implemented such a system[1], to use in our developed approaches, from design-time to run-time. The system has a part of its components running in the cloud, and several components running in mini-clouds or gateways, located in the proximity of IoT devices. As the load of such a system

1 https://github.com/tuwiendsg/DaaSM2M/wiki/IoT-M2M-DaaS

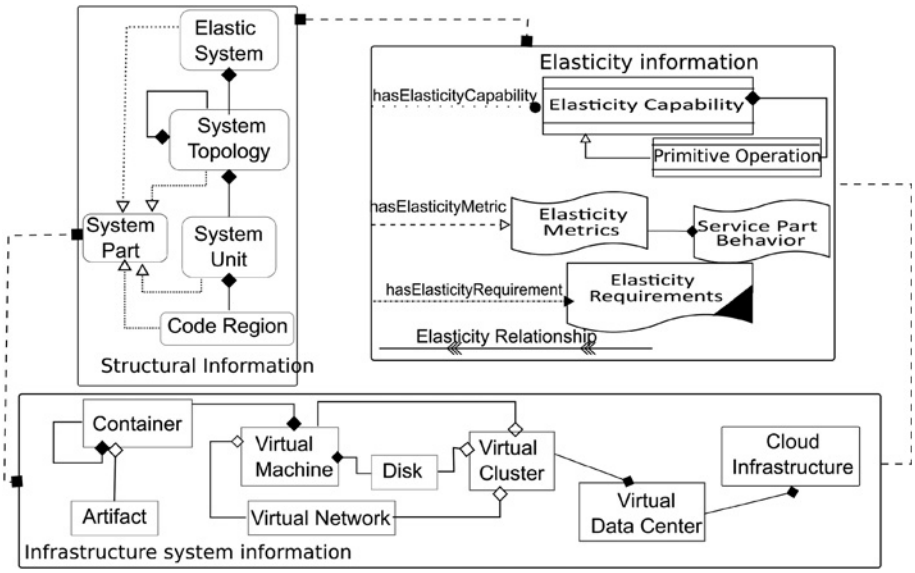

Figure 2: Model for representing elastic cloud systems.

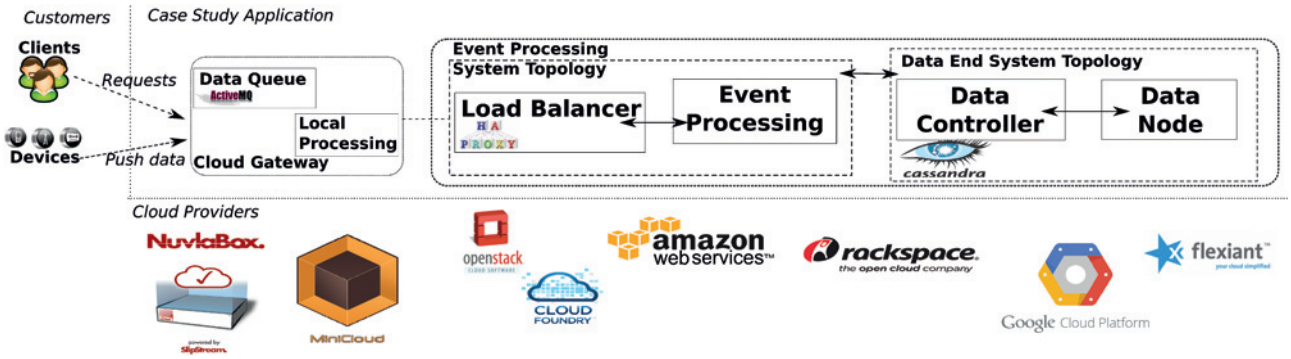

Figure 3: Elastic IoT system.

varies, both the components meant to be located in public clouds, and the local ones, have to continually adapt the used resources and configurations in order to support varying load, varying amount of processing that may be needed, or of data that needs to be stored. Stakeholders would want all of this happening automatically, without the need to train employees for the operation of cloud-based systems, and without the need to allocate extra-effort for this.

Figure 3 shows the system we use as motivating scenario, with two topologies: (i) one system topology local to the devices, which will be deployed in gateways, composed of a data queue and a local processing unit, and (ii) another part which can be deployed in public clouds, composed of a processing topology (i.e., one load balancer, and one event processing unit), and a data end topology (i.e., data controller and data node). We can see a mapping of concepts from Figure 2 to the actual application: system units such as Data Queue and Event Processing, topologies such as Cloud Gateway and Event Processing topology, or relationships among the Event Processing and Data End system topologies. Elasticity capabilities provide the ability to control the system during run-time (e.g., add/remove instances for local processing, event processing and Cassandra[2] data node units, changing the balancing type for the HAProxy[3] Load Balancer, location or access balancing on data cluster).

During runtime, elasticity capabilities should be automatically employed in order to fulfill quality and cost requirements, on different areas of the system (e.g., adding virtual machines to the Event Processing Unit, changing the load distribution from round-robin to priority queuing, re-balancing the Data End System Topology cluster). For this, system's behavior needs to be thoroughly understood, and this kind of information should be the basis of any type of control. Relationships among the throughput with which can be sent in the cloud from the local processing unit, the total cost of hosting the system in the cloud, and the skewness of the Data End cluster, can be used in order to determine the units on which the control should focus. This complex automated control has to interface with operations processes, be them manual or semi-automated. Each of the phases, from design, to analysis, control, and operation produce information that can be used in the rest of the phases, refining their quality (see Section 8).

4 Design

For achieving elasticity at run-time, we need to design the systems with elasticity in mind, from the way we choose the cloud services which we will use, to the requirements that stakeholders have for the system.

4.1 Designing cloud-based elastic systems

The appropriate cloud services must be determined and used for deploying elastic systems in cloud environments. The used cloud services must both provide support for the required system elasticity, and ensure the needed performance and cost. To support the design of elastic cloud systems, we use our QUELLE [25] framework, for quantifying the elasticity of cloud services and determine system deployment configurations using cloud services providing the necessary elasticity capabilities, and which fulfill resources, quality, and cost requirements.

2 http://cassandra.apache.org/
3 http://www.haproxy.org/

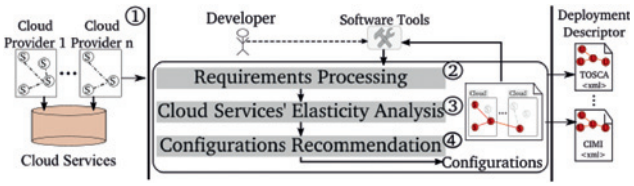

Figure 4: Designing cloud-based elastic systems.

For designing cloud-based elastic system we define a process having the following phases (Figure 4) :

- The elasticity capabilities of cloud services from different cloud providers are captured and modeled ①
- The system requirements in terms of performance, resources, and cost are captured ②
- We define an elasticity quantification function for evaluating cloud services' support for the envisioned system elasticity ③
- We apply algorithms for recommending system configurations from cloud services ④

Elasticity capabilities of a cloud service define how its cost, quality, and resources can be configured during its lifecycle. Concretely, an elasticity capability defines: (i) what resource, cost, quality or associations among cloud services can be created, (ii) when the cloud service can be reconfigured, (e.g., instantiation or run time), and (iii) how often (e.g., hourly, monthly). We introduce in Figure 5 a model for capturing detailed information on elasticity capabilities and elasticity dependencies. As one capability might indicate multiple configuration possibilities, it has a set of elasticity dependencies associated. An elasticity dependency specifies to which cost, quality, resource, or service a cloud service can be associated. Volatility is the most important dependency property, defining its minimum usage time, determining the frequency at which the dependency can be allocated/deallocated for the service.

Our approach recommends system configurations by quantifying the elasticity support of cloud services. To this end we define a series of coefficients for imposing a numerical value over the elasticity capabilities of particular systems, which a user must customize. An

Figure 5: Representing elasticity capabilities of cloud services.

`ElPhaseQ` coefficient is used for quantifying the importance of `Instantiation-Time` and `Run-Time` elasticity. An `ElDepQ` quantifies the elasticity dependencies between cloud services through user-defined values representing the importance of `Optional` and `Mandatory` dependencies. A custom `VolatilityQ` quantifies the volatility of an elasticity capability.

To obtain cloud service recommendations for a system, a user first chooses the cloud provider and describes its services. Then, the user defines the variables for the above coefficients. The user describes requirements over the system units, and the important elasticity dimensions. As a user might not initially know the complete system requirements, we apply an iterative approach. The recommended configurations are analyzed, and the requirements are refined and resubmitted. Let's consider for the elastic system from Figure 3 that its `Data End` requires an IaaS cloud service with 10 GB of RAM, I/O Performance of at least 1000 IOps, and at least a Moderate network performance. After several requirements refinement iterations, QUELLE produces a software-interpretable deployment description visualized in Figure 6 using Winery [16]. The recommendation contains a *m1.xlarge* IaaS service fulfilling the resource and network requirements. A IaaS *EBS* is recommended due to I/O performance requirements, with associated *High I/O Performance* and *High I/O Performance Cost*, and a *Monitoring Service* with *Standard Monitoring Frequency* and associated cost option.

4.2 Expressing requirements

The elasticity requirements are demands formulated by stakeholders, expressing the necessary behavior for the system to be elastic. Our solution for specifying requirements, the SYBL language [6], enables elastic specifications at different granularities, depending on the user's demands and perspective: system, system topology, system unit, and code region.

As opposed to usual languages for scalability, where rules are specified at a per-virtual resource level, SYBL provides greater granularity. At the highest level, the system level, global system characteristics can be described. At the system unit level stakeholders can express requirements for black box system units or requirements focusing at a level lower than system but higher than code, while the code region level enables the user to specify requirements inside their codebase. By providing these finer-grained specification levels, we support stakeholders to choose the level of abstraction they are interested in

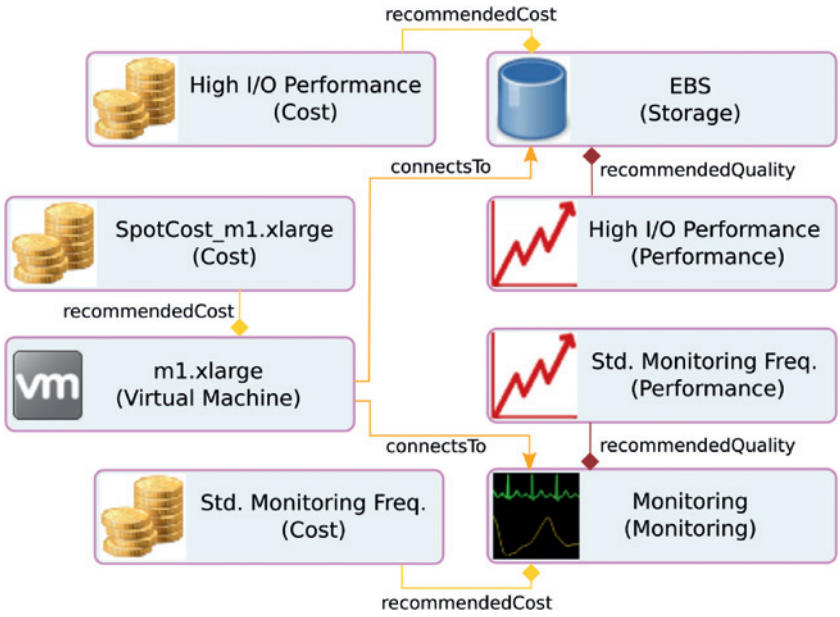

Figure 6: Design recommendation for Data End.

(e.g., system owners on system level, developers on unit level) for expressing requirements.

SYBL is based on three main constructs: monitoring, constraints, and strategies. Monitoring requirements specify what needs to be monitored, and under what conditions. Constraints specify desired system states, while strategies specify desired system behavior.

Listing 1 shows in BNF the constructs of the SYBL language. The monitoring requirements start with a `MONITORING` keyword and specify new variables to be monitored. A constraint defines elasticity requirements for the cloud system state, defining the limits of cloud system behavior. A *strategy* specifies requirements on the behavior of the system. It specifies both control strategies to be enforced under specific conditions, and `WAIT`, `STOP` or `RESUME` actions for the controller, which can be paused/stopped/resumed when specified conditions hold. Therefore, with these two constructs at the center of the SYBL language, constraints and strategies, depending on the stakeholder's or developer's knowledge about the system, we enable various elasticity state and behavior specification mechanisms. Controllers interpret the language, detailed in Section 6, being in charge with determining the specific control mechanisms which enable such system states or behavior.

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

Listing 1. SYBL in Backus Naur Form (BNF).

SYBL hides the complexity of enforcing a variety of complex calls, to different APIs (e.g., cloud provider APIs, or bash configurations) with the help of elasticity capabilities defined in the model in Section 3. It facilitates the system stakeholders or developers to focus more on the elasticity requirements that would help their application to behave as desired.

Current SYBL implementations allow specifying requirements as Java Annotations, using our custom XML format, and as policies in the TOSCA description. An example of a SYBL requirement is shown in Listing 2.

```
<tosca:ServiceTemplate
name="CaseStudyApplication">
<tosca:Policy name="St1"
policyType="SYBLStrategy" >
St1:STRATEGY minimize(Cost) WHEN
high(overallQuality)
</tosca:Policy>...
```

Listing 2. SYBL requirement as TOSCA Policy

5 Assessment

After designing the system, during run-time, its behavior needs to be constantly analyzed in order to provide support both for automated controllers, and for stakeholders wanting to better understand their system.

5.1 Monitoring elastic cloud systems

Elastic systems can re-configure individual components/units at run-time, due to various requirements through: (i) vertical scaling, adding/removing resources to existing components; or (ii) horizontal scaling, duplicating components/instantiating more instances of the same cloud service (e.g., virtual machine, cloud storage). This generates an important monitoring problem. As cloud services are allocated/deallocated dynamically at run-time, if monitoring information is associated only with each service, it will be lost during scale-in operations. Addressing the elasticity requires a different approach in managing collected monitoring information, which we achieve through MELA [26], a framework for monitoring and analyzing elastic systems.

To analyze elastic systems we first categorize monitoring data in three dimensions: Cost, Quality, and Resource (Figure 7). These categories are sufficient for capturing data about any monitored element (e.g., system topology

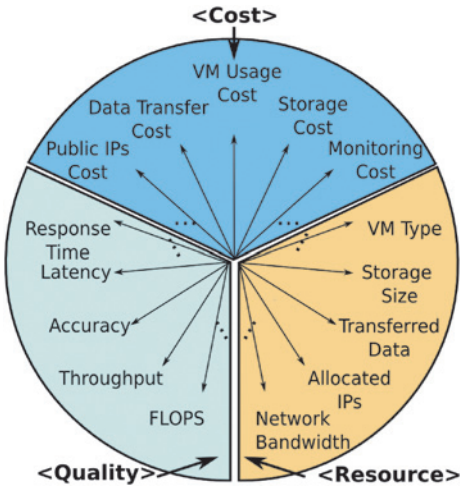

Figure 7: Elasticity dimensions.

or unit) within a cloud system, and can be used for understanding the behavior of that system. For an elastic cloud system, the Quality dimension would capture metrics characterizing the system's quality, such as response time or throughput. The Cost dimension would in turn capture all metrics influencing cost, such as cost of using the virtual machine (e.g., hourly or monthly cost), cost of data transferred over the network, or separate cost of using storage (e.g., cost per each 10 GB of stored data). The Resource dimension metrics capture resource usage and allocation information, such as the amount of data transferred over the network.

To obtain a complete view over the cloud system behavior we compose, aggregate, and enrich monitoring snapshots collected from existing monitoring systems. Depending on the type of metrics, a valid composition of two metrics might involve different operations, depending on the metric type and the information which must be obtained. For example, one would average response time for a system unit to get an indicator of the system's performance, but sum up its throughput to get overall number of performed operations. Thus, we introduce a domain-specific language describing metric composition rules as a cascading sequence of operations which apply one or more operators over one or more operands (Listing 3). For each rule there are at least one *reference metric*, one *resulting metric*, and several *operations*. The reference metric is used as a base for computing the composite metric, and it is searched in the metrics of the target monitored element children having the Target Monitored Element Level. The resulting metric defines the name and unit of the composite metric being created. Defining the composition rules requires domain specific knowledge, knowing which operation is appropriate for which metric.

```
rule := operation "=>" metric
operation:= operator "(" operand { "," 
operand } ")"
operator := "+"|"-"|"*"|"/"|"AVG"|
"SUM"|"MAX"|"MIN"|"SET"|"KEEP"
operand := metric | number | string
metric := name, measurementUnit,
[monitoredElementID],
monitoredElementLevel
```

Listing 3. Metric composition rules grammar

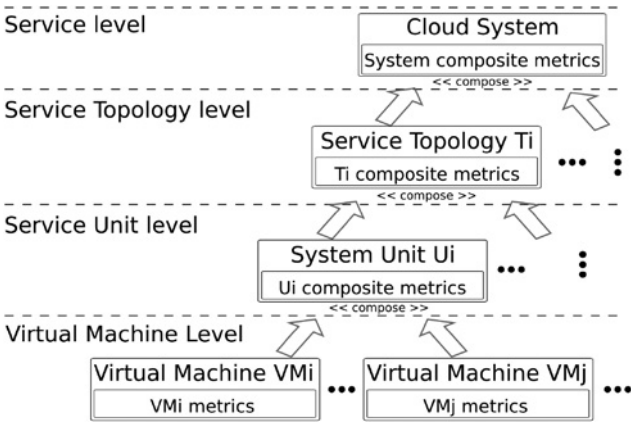

Figure 8: Multi-level metric composition process.

We compose lower level metrics into higher level ones, offering information better suited for elasticity control (Figure 8). The metrics collected from the VM level are aggregated and a System Unit monitoring snapshot is created, giving information about the overall unit behavior. This is further aggregated in a topology snapshot, up to the overall system level snapshot.

5.2 Analyzing elasticity space of cloud systems

The run-time control of elastic systems must ensure that systems fulfill their requirements. However, the whole set of requirements over all system units might not be known, due to system complexity, limited user knowledge over the overall system, or other factors. Considering a two tier application, one might know required system response time, but not what data latency is required from the data end to ensure that response time. Thus, we investigate new concepts that can be used to characterize the cloud system's elastic behavior based on multi-dimensional monitoring data, with which we extend our MELA [26] prototype.

We address this issue by focusing on understanding boundaries over the metric's values in which user-defined elasticity requirements are fulfilled. The known system requirements are expressed as user-defined elasticity boundaries over the system's cost, quality and resources. Based on them, we evaluate collected monitoring information and determine elasticity boundaries for all monitored elements of a cloud system. Thus, from a set of supplied requirements we determine the requirements for all cloud system monitored elements. Based on the determined and supplied boundaries we define the system *elasticity space*, capturing all runtime system metrics and their boundaries. Figure 9 shows system's evolution in time, i.e., elasticity pathway, through the elasticity space.

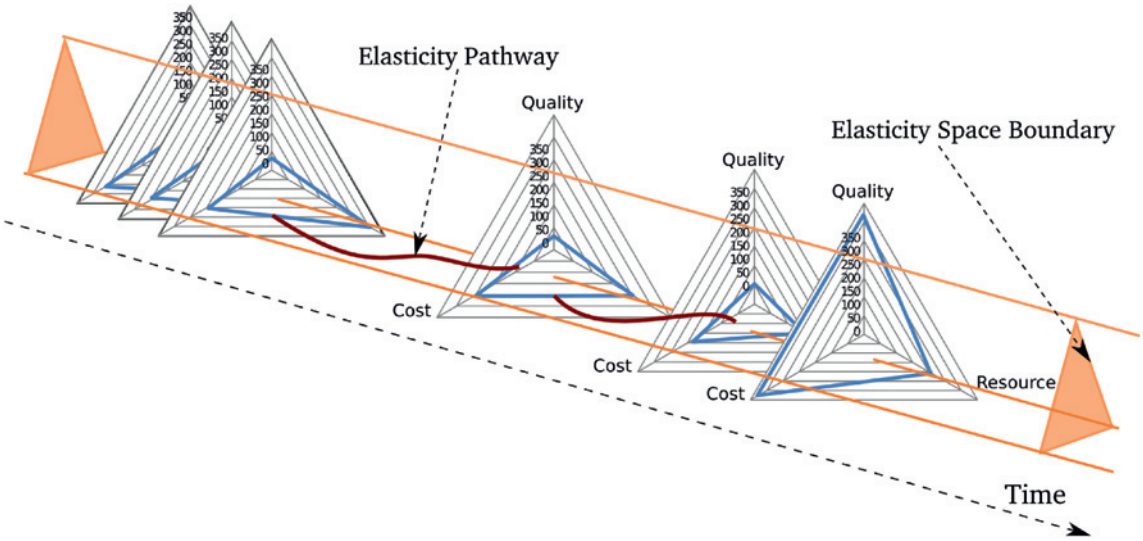

Figure 9: Elasticity space and pathway concepts.

To stay within its elasticity boundaries, an elastic system must scale out/in its cost, quality and resources at run-time. By allocating/deallocating cloud services, the system copes with variations in pricing, quality and load. The elastic behavior is the behavior of a system which is dynamically reconfigured at run-time by software controllers. Formally, let $f_{elSpace}$ be an elasticity space function, and $MS = \{ms_i\}$ be the set of monitoring snapshots. Then an elasticity space *elSpace* can be defined as: $elSpace = f_{elSpace}(MS)$. A $f_{elSpace}$ has to perform two steps: (i) detect when an elastic behavior starts and stops, and (ii) extract monitoring data describing the system's behavior while respecting the user-defined elasticity boundaries. In principle, there could be several elasticity space functions, which can be developed for and applied to different types of monitored elements, e.g., specific types of system units, topologies, or the whole system.

An elasticity space function is designed to extract useful information about the overall behavior of the cloud system when elasticity requirements are fulfilled. For example, given a user-defined elasticity requirement over cost/client/h, an elasticity space might contain only the throughput and cost/VM/h metrics from which the cost/client/h targeted by requirements can be determined. It would not include metrics that have no impact on the cost/client/h. Thus, using the elasticity space, one can determine the elasticity boundaries to be enforced on the metrics that influence the user-defined elasticity requirements. Moreover, one can analyze if the behavior of an elastic cloud system is within expected user-defined elasticity boundaries by checking the elasticity boundaries of its elasticity space. For example, the upper elasticity boundary of the cost/client/h from the determined elasticity space could have a different value than expected by the user.

The elasticity space enables analysis over the system's elastic behavior. Thus, based on the analysis, one can define an *elasticity pathway*, characterizes the relationships among system metrics in time. An elasticity pathway $elPtw$ is determined by a function f_{elPtw} which takes as input an elasticity space *elSpace* and a set of metrics M, and returns another function describing behavioral patterns or characteristics of the monitored element: $elPtw = g(M) = f_{elPtw}(elSpace, V)$. Various functions can be defined over the elasticity space, enabling space analysis from multiple perspectives.

In the current MELA prototype, we implement an *elasticity space function* which determines as space boundaries the maximum and minimum encountered metric values when the user-defined elasticity requirements are respected. The function uses as input the user-defined elasticity requirements for the whole cloud system, and is applied for all system topologies, units, and unit instances. Let's consider the Event Processing Topology from Figure 3, with requirements over certain response time. Our elasticity space function determines the boundaries for the other collected metrics (depicted in Figure 10), determining the maximum and minimum CPU usage and throughput the Event Processing Topology must fulfill. The determined boundaries show the resource congestion and throughput limits any elasticity controller should monitor and enforce to ensure the system fulfills initial requirements.

5.3 Analyzing elasticity relationships in elastic systems

Cloud system are becoming more and more complex. In elastic cloud systems, individual system units are typically not behaving independently. Instead, due to communication or run-time control dependencies, there exist different relationships between system units, influencing their run-time behavior. In the following we describe our approach for determining the elasticity relationships influencing the behavior of elastic systems [24].

An elasticity relationship between one elastic element and a set of other elements describes the change in the behavior of the first element w.r.t. its elasticity boundaries, triggered by a change in the behavior of the other elements. Particular relationships can be of interest for particular stakeholders. However, analyzing such relationships is challenging. Due to the potential complexity of the system's software stack, each software layer can introduce different relationships. Second, the relationships can vary between different cloud providers. To this end, we focus on determining relationships between any of the system's performance, cost, and resource usage.

The most important for a relationship is determining the change function describing how much the value of the elasticity metrics of one element change. Elasticity of cloud systems is evaluated based on boundaries over the system's metrics defined by elasticity requirements. For determining the relationship we abstract from concrete monitored values, the behavior of elastic systems with respect to their requirements (i.e., elasticity boundaries). To this end we define the concept of *Elasticity Work* of a cloud system as the current load on the system with respect to its elasticity boundaries. We define the difference between the current elasticity work and the elasticity boundaries as the *Elasticity Energy*. Using these concepts illustrated for a single metric in Figure 11, we determine relationships

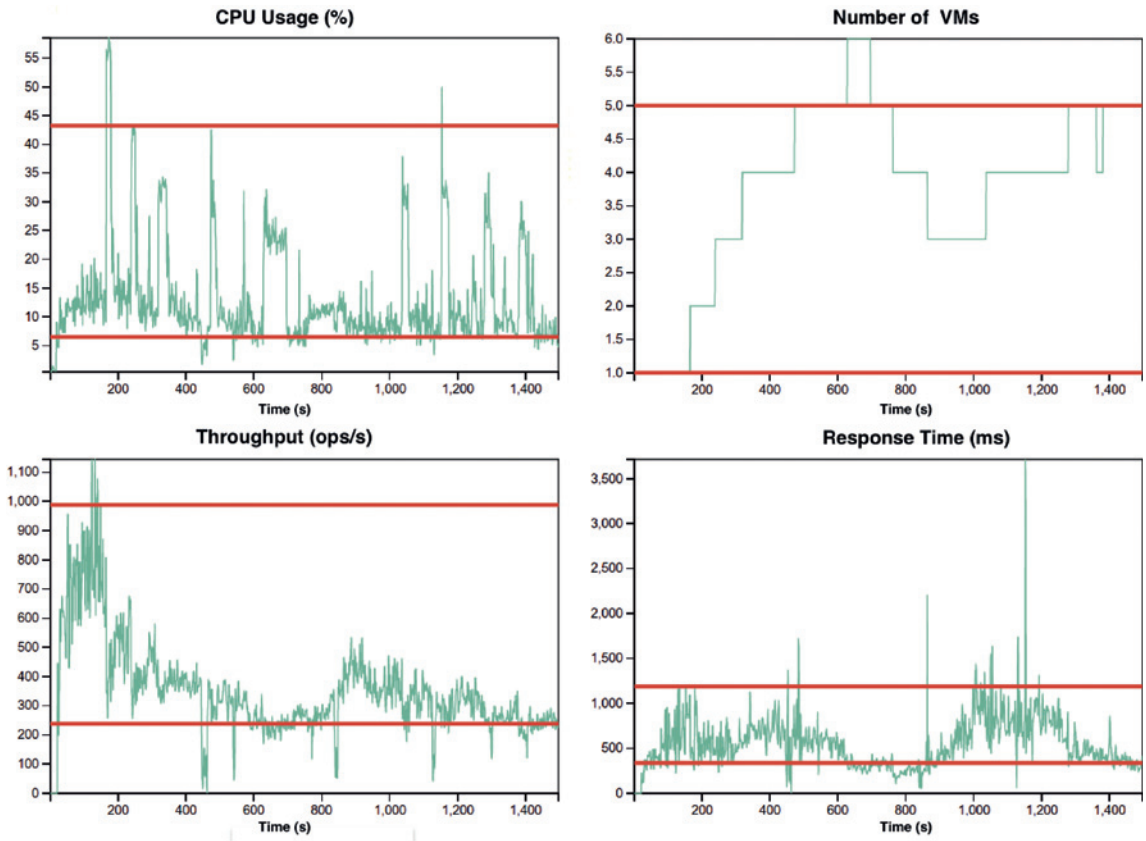

Figure 10: Event processing topology elasticity space and boundaries.

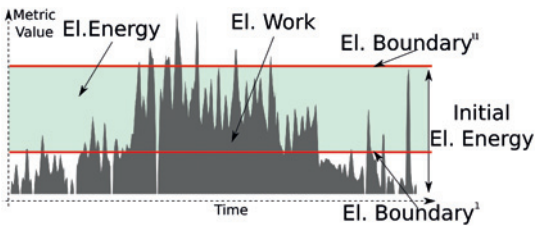

Figure 11: Elasticity boundary, work and energy concepts.

between the elasticity energy of two systems, and not individual metric values.

Let us consider the case study application. At runtime, an elasticity controller scales the system, according to its requirements. From the system structure (Figure 3) we can assume that relationships should exist between units which communicate directly, such as Load Balancer and Event Processing. However, other relationships might not be so obvious, being generated by indirect communication (e.g., a relationship between Load Balancer load influencing the performance of the Data Node).

We extend MELA with a mechanism for determining relationships based on R[4]. We use a linear regression approach, computing the linear correlations between two energy time series. For each determined coefficient of the linear relationship, we check if the estimation error is one order of magnitude smaller than the coefficient, and if not, we discard the relationship as inaccurate. Finally, we obtain the change function, with the associated coefficient of determination, an indicator on how well the extracted relationship fits the original data.

Applying our approach on the case study application (Figure 3) we determine a relationships graph between system metrics (Figure 12). We focus on the relationship between the `cpuUsage` of the Data End Unit and `throughput` of the Event Processing. The determined relationship (Figure 13) is a linear equation in which the elasticity energy of `cpuUsage` at time `t` can be estimated by multiplying the `throughput`'s energy value at time `t` with 0.29, and adding 2.86 (i.e., the free term in the linear relation in Figure 13) to the result. Converting the abstract

4 http://www.r-project.org/

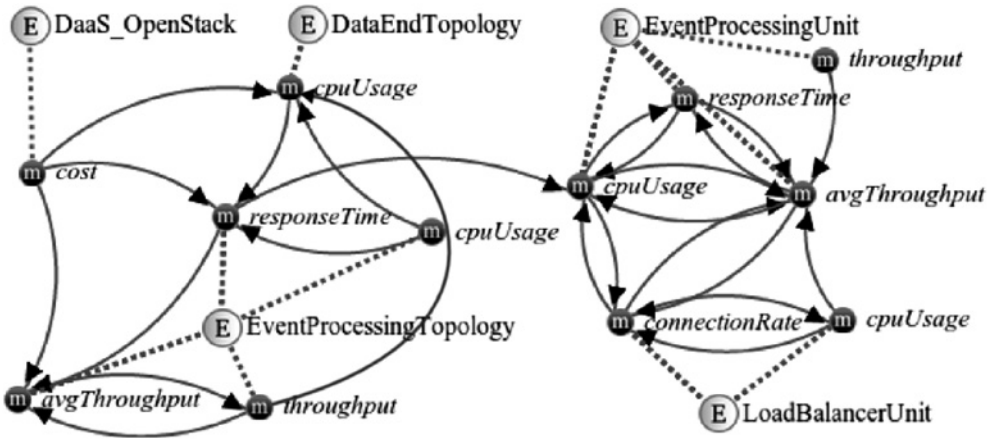

Figure 12: Case study application on private cloud – determined elasticity relationships graph.

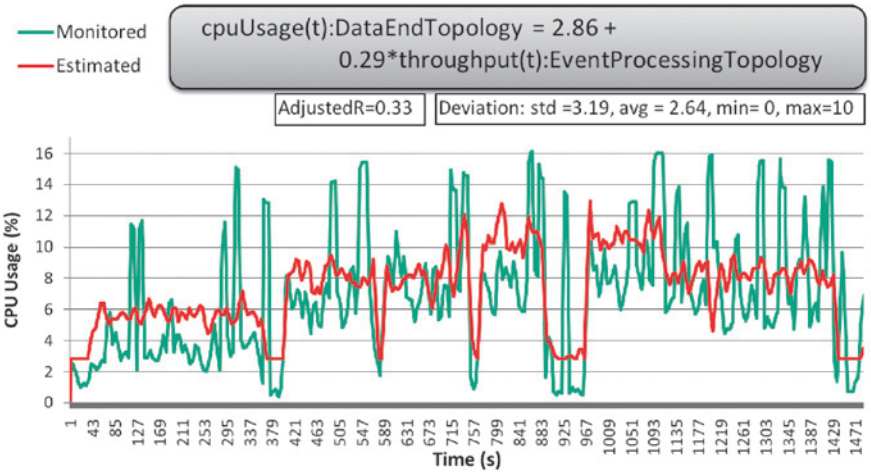

Figure 13: Case study application on private cloud – determined quality relationship.

energy to concrete values with respect to the current elasticity boundaries of the system, we can estimate `cpuUsage` based on the `throughput`'s monitored values. From the relationship, we estimate that, using this deployment structure, with maximum accepted CPU usage (from elasticity requirements) of around 90%, the maximum achievable throughput is $(90 - 2.86)/0.29 \approx 300$ sensors per second.

Thus, using our approach, one learns that when more than 300 sensors are connecting to the case study application, the data end should be scaled out. Thus, we enable users to understand how system units influence each other at run-time, crucial in designing appropriate elasticity control mechanisms.

5.4 Estimating elastic systems behavior

In controlling cloud systems, a common approach employed by many elasticity controllers [1] [34], is to allocate/de-allocate virtual resources when threshold rules are violated. This approach may be sufficient for simple cloud systems, but for large-scale distributed cloud systems with complex inter-dependencies among components, we need a deeper understanding of their behavior for a better control. Therefore, we need to be able to understand which control processes are most appropriate for a particular situation, and what are their effects, in time. Moreover, we are interested in this not only on the system part which is directly involved in the control process but also on the rest of the system.

For estimating this behavior we propose using a three-phase approach [7] (see Figure 14): (i) mapping past behavior related with control processes to multi-dimensional points for keeping the relationship among consecutive metric values, and (ii) using a clustering based approach to form clusters from these multi-dimensional behavior points, showing common behavior patterns, and (iii) mapping current behavior in multi-dimensional behavior points, and computing behavioral clusters that are closest to current behavior, for obtaining the expected behavior.

Figure 15 shows, on the left side, the current behavior of the data node. Based on this information, and on historical information, our approach is able to estimate future

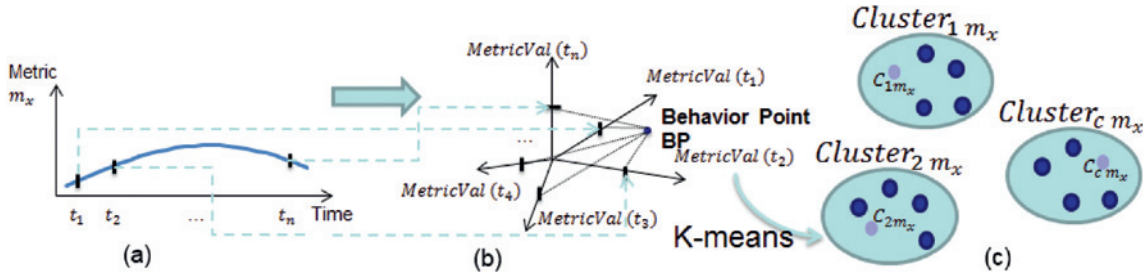

Figure 14: Estimating elastic systems behavior.

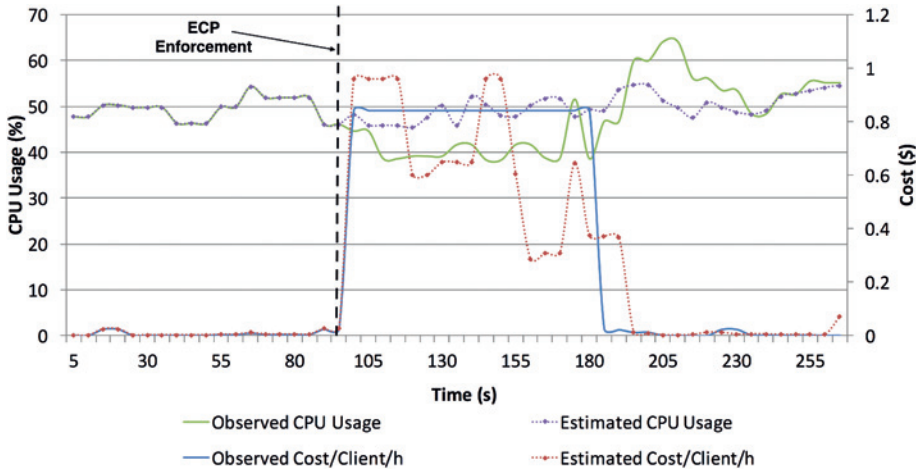

Figure 15: Effect of scale-in data node on the use-case application.

behavior. In the right side, we show with continuous line the estimated behavior, and dotted line the observed behavior. Our approach is able to estimate with a high degree of accuracy the future behavior for the case of enforcing a scale-in process consisting of decomissioning the unit, and removing virtual resources.

6 Management

Based on the assessment techniques presented above, we design a control mechanism able to automatically adapt the system to the load, considering the requirements and the complex cost schema of cloud providers. Based on this, we further provide support for stakeholders for operations management, focusing on the amount of information they receive, the information type, and the way they can interact with the running system.

6.1 Multi-level elastic cloud systems control

As there are various types of stakeholders interested in cloud-hosted systems (e.g., cloud systems developers and cloud systems providers), they have different preferences at the various abstraction levels. They have coarse or fine grained knowledge about their cloud systems, with regard to various matters (e.g., the provider knows how much s/he is willing to pay for the entire system to be hosted on the cloud, while the developer knows quality indicators at different layers of the system). However, most of existing mechanisms are limited to assuming the level of trade-off necessary between cost and quality [2, 29, 32]. Moreover, we need to manage both the static description of the cloud system, and its runtime behavior, which depends on the virtual infrastructure on which it runs.

Our approach [5] uses multiple types of information, such as requirements specified by stakeholders at multiple abstraction levels, associated to the system model, and monitoring information. We implemented this technique in the rSYBL framework[5], having connectors for multiple monitoring frameworks and cloud providers.

Based on the model described in Section 3, at runtime we construct a dependency graph, in which nodes are instantiations of concepts from the model (e.g., system units, system topologies), and edges are given by relationships. Based on it, we enable elasticity control simultaneously

5 https://github.com/tuwiendsg/rSYBL

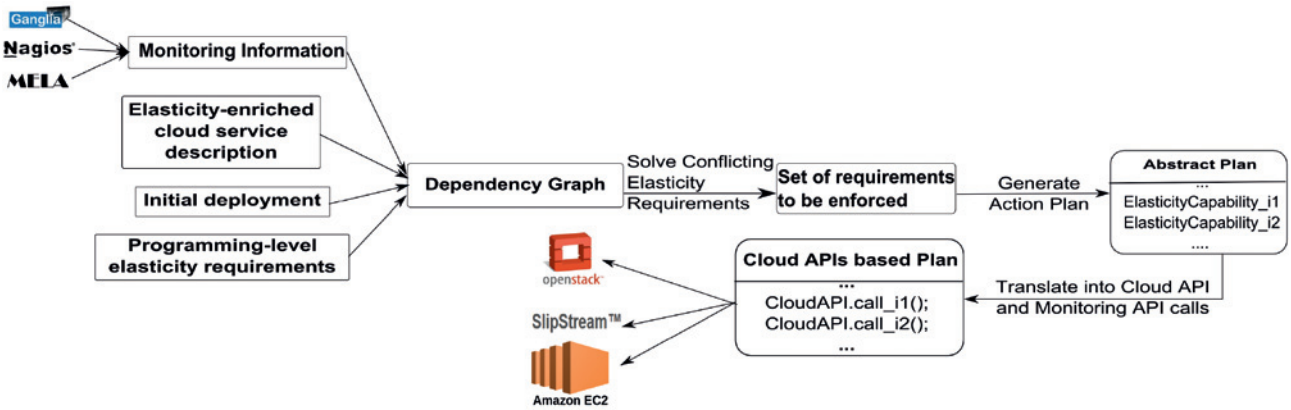

Figure 16: Elasticity control – from requirements to enforced plans.

for each of the described nodes, resulting in a multi-level elasticity control of the described cloud system, based on the flow shown in Figure 16. The system stakeholders describe their system using TOSCA [4] or other description standards. The initial deployment configuration is specified either by the automatic deployment tool used or by developers if a manual deployment approach is chosen. The elasticity requirements are evaluated and conflicts that may appear among them are resolved (for details on conflict resolution please refer to [5]). After that, an action plan is generated, consisting of elasticity capabilities that enable the fulfillment of specified elasticity requirements. The action plan is composed from elasticity capabilities that have associated a series of cloud provider API calls, configurations, or bash/scripts executions.

For generating an action plan for cloud system elasticity control, we formulate the planning problem as a *maximum coverage problem*: we need the minimum set of capabilities that help fulfilling the maximum set of requirements. Since maximum coverage problem is an NP-hard problem, and our research does not target finding the optimal solution for it, we choose the greedy approach. The main step of the greedy approach consists of finding each time the elasticity capability EC_i fulfilling the most constraints and improving the most strategies. For calculating the effects, the elasticity behavior estimation technique presented in the previous section is used.

The results of running a sinusoidal workload with a high amount of bursts are shown in Figure 17. For this case, we assume the cost of a VM 0.12 $ per hour. The Event Processing System Topology is being controlled, in time, for fulfilling requirements targeting the cost and throughput.

6.2 Cost-aware control of elastic systems

Run-time costs evaluation is required for understanding and controlling elastic systems running in public clouds. In what follows, we capture cloud providers pricing schemes, evaluate the costs and cost efficiency of elastic systems, and use them to define an approach for cost-aware scale-in of elastic cloud systems [27].

Systems deployed in public clouds can be built from a proprietary software and public cloud services. While systems are scaled-out due to performance requirements, cost is the main driver for system scale-in [22]. However, cost of elastic systems is complex. Certain costs can be static, such as costs for reserving a cloud service. Other costs can be dynamic, such as modifying the price of using a cloud service depending on its usage. Cost can also depend on particular service combinations.

Towards costs analysis, we define a cost representation model detailing the *Cost Element* concept (Figure 18). Each cost element that is associated with a cloud service can be either reservation-based or per usage. Next, it can be defined per billing cycle, over its reservation time or usage. Additionally, cost can be specified in intervals, specifying intervals of measured units over the billing metric. Finally, a cost element might be applicable if the service is used in a particular configuration, including concrete resource and quality properties, or other cloud services the service should have associated.

We enrich MELA with the previous model and use it to compute and visualize the costs of our use-case in Figure 19 focusing on the Event Processing topology. The system costs are composed of the costs of its topologies, composed from the costs of system units. The cost of individual units consists of the costs of the used instances of *VM* services, and the costs of the *Cloud Storage*, reported both in price per disk size, and per size of read/written

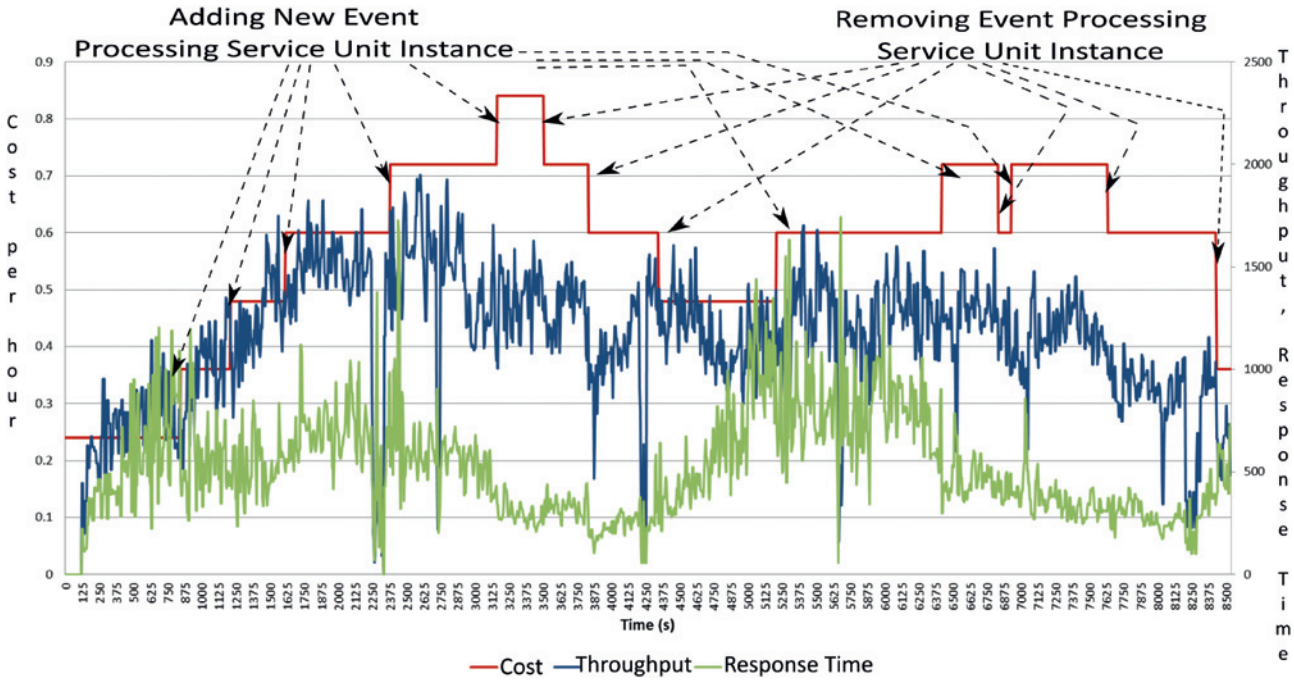

Figure 17: Controlling the case study application: throughput vs. cost on event processing system topology.

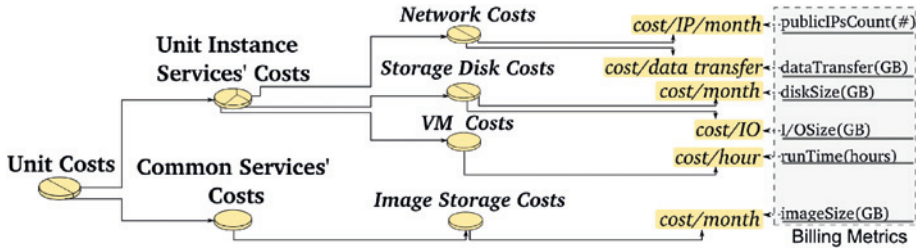

Figure 18: Example of scalable unit costs complexity.

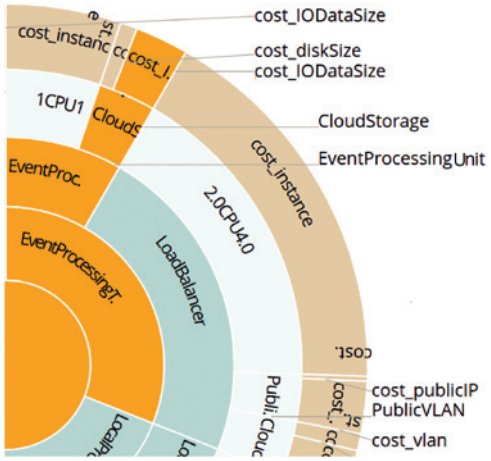

Figure 19: System costs composition for event processing topology.

data. We also have the costs of additional used services, such as the *Public VLAN*. From the cost proportion of each cost element, the developer can understand which units have a low cost impact, and which are costly. Using this information, the developer can reduce costs by redesigning the system, or changing the used cloud services or cloud provider.

After multiple scale in/out actions, the billing cycles of different instances of the cloud services used by the system units can become desynchronized, i.e., billing occurs at different points in time, depending on when each instance was allocated, and on how much it was used (Figure 20). When scaling a cloud system, a cost-aware controller should deallocate a unit instance with a usage of 100% over all its cost elements, w.r.t., the billing cycle of each element, e.g., if a unit uses cloud services billed per hour and per GB of data, it is cost efficient to deallocate it when it has run for an integer number of hours, and has generated an integer number of GBs.

To evaluate the benefit of considering cost efficiency we have implemented a scalability controller supporting two cost-agnostic and two cost-aware scaling strategies. The cost-agnostic strategies deallocate the *Last* and *First* allocated unit instance. The first cost-aware strategy

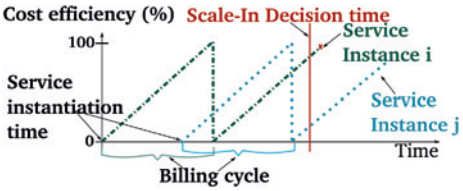

Figure 20: Service instance cost efficiency w.r.t. billing cycle

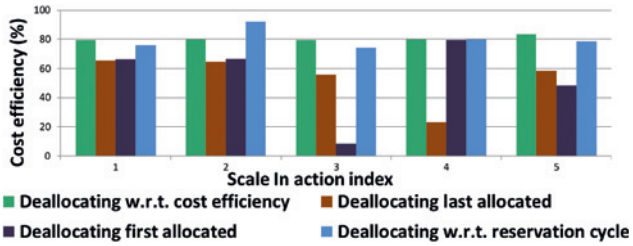

Figure 21: Scale-in cost efficiency under different strategies.

deallocates a unit instance and its cloud services when its VM has run over 90% of its reservation billing cycle. The second cost-aware strategy deallocates a unit instance when its cost efficiency is over 80%.

From Figure 21, we can notice that except one action, the strategy deallocating units when their cost efficiency is over 80% obtains better cost efficiency even compared to the one deallocating units at over 90% of their reservation billing cycle. This highlights that all cost elements need to be considered when scaling elastic systems.

In highly fluctuating load, controllers can use the cost efficiency in order to reduce the costs, keeping services that are already payed for even if they are not necessarily needed. As the load fluctuates, supplementary services might be needed soon, thereby increasing the control stability. As shown in Figure 22, under the cost-agnostic strategy, the controller deallocates unit instances as soon as requested. In contrast, the cost-aware controller waits until a unit instance has reached 80% cost efficiency before deallocating it. If a scale-out is requested, the cost-aware controller will verify if it is waiting to scale-in, and cancel the pending action.

Employing cost-aware scalability controllers in public clouds can also increase the performance and stability of cloud applications, while reducing their costs. As shown in Figure 22, the cost-aware strategy avoided unnecessary scale-ins, while the cost-agnostic one deallocated and allocated back unit instances.

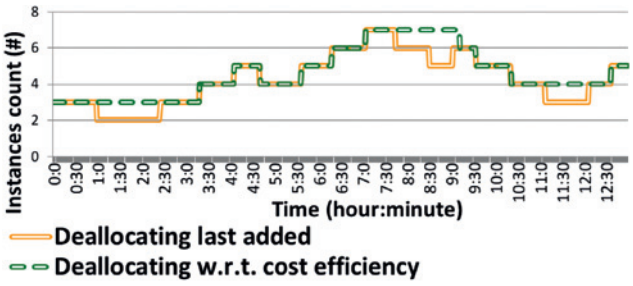

Figure 22: Number of event processing instances under cost-aware and cost-agnostic scalability.

6.3 Elastic cloud systems operations management

A system deployed in the cloud can use various resources and services offered by cloud providers, and can be very dynamic at run-time. The cloud is one of the most dynamic environments. Providers can change their cost schemas, and the offered service characteristics, from one day to another. Although in this environment automated controllers such as the ones shown above are necessary, given this high dynamism, it might be necessary for system stakeholders to re-examine the desired behavior of a cloud system, and their interactions with other stakeholders (e.g., cloud providers, or data providers). In this context, the responsibilities and processes of operation management teams need to be revisited, in order to accommodate characteristics of elastic systems. Even more, the control needed for these kind of complex systems should be semi-automated [30], still allowing the operation roles to intervene where needed. The approach that we choose for our solution is supervisory control (i.e., level of automation 9 from Endsley's levels of automation [11]), for allowing, from the operation roles side, both the supervision of the controller and the system, and the intervention in the control process (e.g., through refinement of requirements).

Our solution [8] involves adding *roles* (i.e., employees) as first class entities in cloud system elasticity control loops. Based on the roles, we define necessary interaction protocols for system's elasticity-driven operation management. We focus on interactions between roles and elasticity controllers, but we also support interactions among employees, for notifying each other of updates or for delegating responsibility related to various events. We extend SYBL to support roles and role-based communication between stakeholders. Based on this, we develop an elasticity Operations Management Platform (eOMP) for cloud systems, and we validate its usefulness showing various events encountered for a complex system. eOMP can support different organization structures,

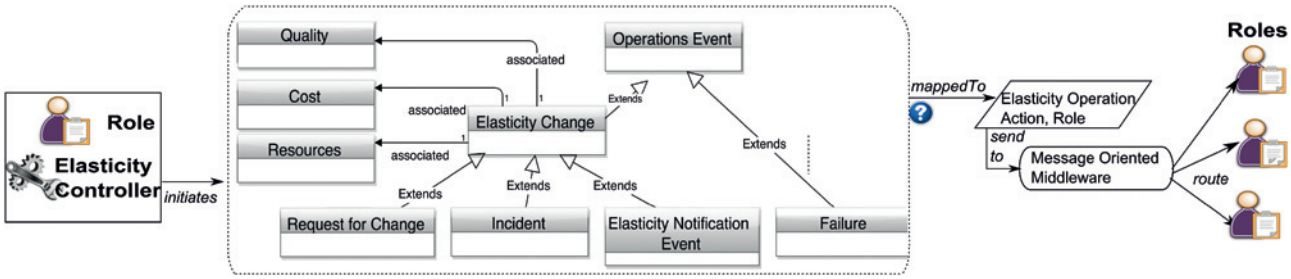

Figure 23: From elasticity changes to responsible roles.

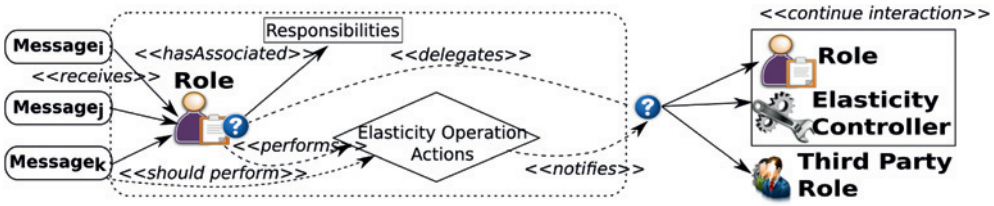

Figure 24: From roles to elasticity operation actions.

and enables employees to interact easily with the elasticity controller, for obtaining a more complex elasticity control. eOMPs supports managing unexpected situations, by facilitating the collaboration between elasticity controllers and stakeholders, identifying various types of events occurring during operation, and providing mechanisms for solving them.

In order to support operations management for elastic cloud systems, the roles and elasticity controller need to collaborate in order to manage system operation during runtime. As shown in Figure 23, roles should be notified by other roles or by the elasticity controller concerning the operation events that occur in relation to the elasticity of current system. From the operation events, we focus on any type of elasticity changes. System stakeholders are usually interested in failure or quality-related events, in order to be able to learn from system behavior and environment changes that produce failures. As shown in Figure 24, a role can receive a multitude of messages from other roles or from elasticity controllers. Analyzing them, the role decides whether it can perform needed actions, or if it should delegate to other roles. After analyzing the interaction flow, the next section is focused on analyzing the roles and their responsibilities and interests in elasticity operations management.

In eOMP, for ensuring that the correct role receives messages according to its responsibilities, and its authority, we define filtering and aggregation methods for directing messages to the correct roles. Figure 25 shows the amount of messages received by each of the roles in the organizational structure, for our use-case scenario based on the case study application.

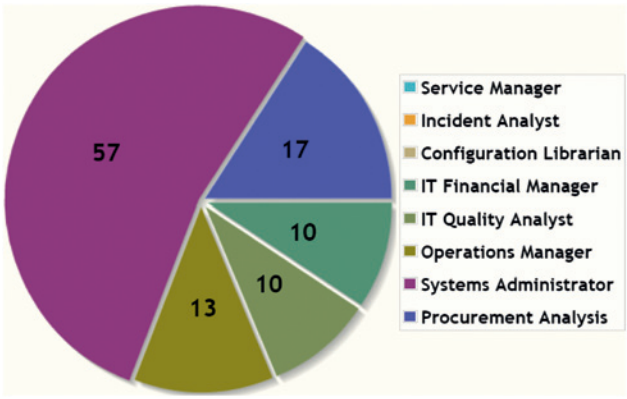

Figure 25: Total role interactions.

This way, we start bridging the gap between the ever more automated world of elastic systems, and the operations management world, with clearly defined processes and responsibilities. Although the automation of elastic systems control does take over some of traditional responsibilities from system operation management, it should also involve an adaptation of responsibilities, and a change of focus in order to meet the dynamism and needs of elastic systems.

7 Related work

In requirements languages, multiple solutions have been proposed in recent years. Kouki et al. [17] propose an extension of CSLA (Cloud Service Level Agreement) [3], focusing on QoS degradation and penalty models for an easier and clearer interaction between the cloud customer and the

cloud provider. Li et al. [20] propose PSLA (PaaS level SLA) language for the description of SLA, focusing on workload elasticity and PaaS-specific properties. CloudMF [13] is a language for cloud infrastructure resources management, with focus on uniformly describing applications for reducing vendor lock-ins. Based on CloudML [14], Kritikos et al. [18] propose SRL, a policy language for scaling multi-cloud applications. Zabolotnyi et al. [35] propose SPEEDL, a declarative language for event-based scaling of cloud applications. The major difference between existing work and our approach is that our work tackles elasticity requirements from more than one perspective (resource, quality, cost) and at different levels of granularity, thus assigning the user the capacity of specifying when the application should scale throughout its execution and, most importantly how.

An elastic monitoring framework for cloud infrastructures is presented in Konig et al. [15], based on a peer-to-peer architecture enabling the authors to monitor a diverse set of entities and metrics, spanning across all layers of a cloud stack. Also dealing with dynamic infrastructures, Trihinas et al. [33] introduce JCatascopia, a tool for monitoring elastic systems, employing dynamic probe addition and removal to cope with infrastructure dynamicity. Moreover, to provide support for monitoring elasticity, monitoring probes can be activated/deactivated dynamically during system run-time, if required by elasticity controllers. Moving further into system-level monitoring, Leitner et al. [19] apply complex event processing techniques to extract system-specific performance information from system-level metrics. To this end, monitored data is expressed as event streams, the authors determining the system state using rules targeting sequences of detected events. Highlighting that in cloud environments one cannot assume the existence of online distributed monitoring nodes and reliable inter-node communication, Shicong et al. [23] present an adaptive cloud monitoring system providing information about monitoring message delay and loss. Further, the authors provide estimations on monitoring accuracy, and capture uncertainties introduced by messaging problems. Cloud monitoring is also the focus of many industry tools such as Nagios[6], Zabbix[7], OpenNMS[8], or Hyperic[9]. Such tools mostly focus on gathering data from the physical and virtual infrastructure, and distributing it, without correlating it with the systems running on it. We differ as we do not focus on monitoring. Instead, we rely on data from existing monitoring solutions, aggregate and enrich it according to the system's structure.

In controlling systems, Tolosana-Calasanz et al. [32] propose controlling resources necessary for data streams, using a shared token bucket approach. Dupont et al. [10] propose the notion of software scalability, both horizontal and vertical, by drawing inspiration from infrastructure scalability. For SaaS providers, horizontal (i.e., adding/removing more software units), or vertical (i.e., increasing/decreasing offerings for the service), can be a good customization opportunity to profit from elasticity at IaaS level. Aragna et al. [2] define metrics and rules for elasticity control, and study various scenarios (e.g., infrastructure only, or database only control). Nakos et al. [29] propose an approach for elasticity control based on dynamic instantiated Markov decision processes, using probabilistic model checking. Ferry et al. [12] propose an approach for the continuous management of scalability in multi-cloud systems, based on CloudMF [13]. Compared to the above-mentioned work, we control elasticity not just in terms of resources but also in terms of quality and cost and use application structure for proposing an accurate multiple level control of elasticity of cloud services. Furthermore, they lack user-customized elasticity control.

Several commercial solutions enable cloud infrastructure management support, but do not support service operations management at cloud customer's service level (e.g., VMWare vCloud Suite[10], Oracle Enterprise Manager[11]). Liu et al. [21] propose an incident diagnosis approach based on incident relationships, using co-occurring and re-occurring incidents for performing root cause analysis. Munteanu et al. [28] propose an architectural approach for incident management in the cloud, from service monitoring perspective, including incident lifecycle management, event and incident detection, incident classification and recovery and root cause analysis. In contrast with above presented work, we focus on the elasticity aspect of service operations management in the cloud, characterizing the relevant properties and interactions. Moreover, we emphasize the importance of supervisory control for the cloud, and introduce service provider employees as first-class entities in the control loops.

6 http://www.nagios.org/
7 http://www.zabbix.com/
8 http://www.opennms.org/
9 http://www.hyperic.com/

10 http://www.vmware.com/products/vcloud-suite
11 http://www.oracle.com/technetwork/oem/enterprise-manager

8 Continuous elastic systems management

As emphasized in Section 2, design and operation of elastic systems are not consecutive or atomic phases (see Figure 1). In each of the operation steps, gathered information can be useful for design refinement: possible issues could be fixed by developers, new metrics and capabilities could be introduced, other services can be used for the initial deployment, and requirements could be refined. This merges design and operations processes, following the DevOps culture [9], in order to provide continuously best quality and flexibility in business requirements.

First, when monitoring and analyzing the system at runtime, one can detect a series of patterns in behavior, possible bottlenecks, and possible flaws in the system design, as detailed in Section 5. When reported to operation managers, this information needs to be fed back in a design refinement, in order to develop new features, introduce new units (e.g., introduce load balancers where needed), or adapt requirements considering system's elasticity space. Moreover, estimations on the effect capabilities have (e.g., adding virtual resources for a unit produces little effect), enables developers to look into possible issues with the respective unit, or possible issues with the actual capability. Next, while the system is running, stakeholders should be able to refine requirements based on the observed control outcome produced by initial requirements. This would be done either by operation managers, or by operation managers with the support of the development team. Furthermore, a complete support for elastic systems at run-time means also operations management support, as shown in Section 6. We are supporting roles of various types in receiving the relevant information, and in interacting with semi-automated controllers for managing operation processes for the running systems.

All of the above can and should be applied, on any elastic systems, not only on cloud-based systems, as it was the case for our proposed solutions. The driving properties for our solutions were the systems' dynamism, heterogeneity, replaceability of composing units, and business perspective. These properties are defining for any types of elastic systems, be them cloud systems, human computing based systems, or IoT systems. Therefore, our solutions can be extended to support even more complex scenarios, integrating people, things, and computing in open dynamic ecosystems.

9 Conclusions

We have presented our solution for supporting elastic systems development and operation. We support designing native cloud systems with elasticity in mind, for specifying requirements, and assessing and managing their behavior at run-time.

Elasticity is not a defining property only for cloud systems, but of systems in general. In a world where service-based systems are evermore used, and where physical things become part of our digital world, elasticity should be one of the main properties of every system. The techniques that we have introduced for building elastic cloud-based systems can be extended to other domains, where the targeted characteristics still hold: heterogeneity, dynamism, business orientation, and replaceability of composing units.

References

1. A. Al-Shishtawy and V. Vlassov. Elastman: Autonomic elasticity manager for cloud-based key-value stores. In *International Symposium on High-performance Parallel and Distributed Computing (HPDC)*, pages 115–116, New York, NY, USA, 2013. ACM.
2. C. Ardagna, E. Damiani, F. Frati, G. Montalbano, D. Rebeccani, M. Ughetti, et al. A competitive scalability approach for cloud architectures. In *International Conference o Cloud Computing (CLOUD)*, pages 610–617. IEEE, 2014.
3. EU Commission. Cloud Service Level Agreement. http://ec.europa.eu/digital-agenda/en/news/cloud-service-level-agreement-standardisation-guidelines.
4. O. T. Committee. Oasis topology and orchestration specification for cloud applications. Technical report, OASIS, 2010.
5. G. Copil, D. Moldovan, H.-L. Truong, and S. Dustdar. Multi-level Elasticity Control of Cloud Services. In S. Basu, C. Pautasso, L. Zhang, and X. Fu, editors, *Service-Oriented Computing*, volume 8274 of *Lecture Notes in Computer Science*, pages 429–436. Springer Berlin Heidelberg, 2013.
6. G. Copil, D. Moldovan, H.-L. Truong, and S. Dustdar. Sybl: an extensible language for controlling elasticity in cloud applications. In *International Symposium on Cluster Computing and the Grid (CCGRID)*, 2013.
7. G. Copil, D. Trihinas, H.-L. Truong, D. Moldovan, G. Pallis, S. Dustdar, and M. Dikaiakos. ADVISE – a Framework for Evaluating Cloud Service Elasticity Behavior. In X. Franch, A. Ghose, G. Lewis, and S. Bhiri, editors, *Service-Oriented Computing*, volume 8274 of *Lecture Notes in Computer Science*. Springer Berlin Heidelberg, 2014.
8. G. Copil, H.-L. Truong, and S. Dustdar. Supporting Cloud Service Operation Management for Elasticity. In A. Barros, D. Grigori, N. C. Narendra, and H. K. Dam, editors, *Service-Oriented Computing*, number 9435 in Lecture Notes in Computer Science, pages 123–138. Springer Berlin Heidelberg, Nov. 2015. DOI:10.1007/978-3-662-48616-0_8.

9. P. Debois. Devops: A software revolution in the making. *Journal of Information Technology Management*, 24(8):3–39, 2011.
10. S. Dupont, J. Lejeune, F. Alvares, and T. Ledoux. Experimental Analysis on Autonomic Strategies for Cloud Elasticity. In *International Conference on Cloud and Autonomic Computing (ICCAC)*, pages 81–92. IEEE, Sept. 2015.
11. M. R. Endsley and D. B. Kaber. Level of automation effects on performance, situation awareness and workload in a dynamic control task. *Ergonomics*, 42:462–492, 1999.
12. N. Ferry, G. Brataas, A. Rossini, F. Chauvel, and A. Solberg. Towards bridging the gap between scalability and elasticity. In *International Conference on Cloud Computing and Services Science-Special Session on Multi-Clouds (CLOSER)*, pages 746–751, 2014.
13. N. Ferry, H. Song, A. Rossini, F. Chauvel, and A. Solberg. Cloudmf: Applying mde to tame the complexity of managing multi-cloud applications. In *International Conference on Utility and Cloud Computing (UCC)*, pages 269–277. IEEE Computer Society, IEEE/ACM, 2014.
14. G. Goncalves, P. Endo, M. Santos, D. Sadok, J. Kelner, B. Melander, and J.-E. Mangs. Cloudml: An integrated language for resource, service and request description for d-clouds. In *International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 399–406. IEEE, Nov 2011.
15. B. Konig, J. Alcaraz Calero, and J. Kirschnick. Elastic monitoring framework for cloud infrastructures. *IET Communications*, 6(10):1306–1315, 2012.
16. O. Kopp, T. Binz, U. Breitenbücher, and F. Leymann. Winery - a modeling tool for tosca-based cloud applications. In S. Basu, C. Pautasso, L. Zhang, and X. Fu, editors, *International Conference on Service Oriented Computing (ICSOC)*, volume 8274, pages 700–704. Springer Berlin Heidelberg, 2013.
17. Y. Kouki, F. de Oliveira, S. Dupont, and T. Ledoux. A language support for cloud elasticity management. In *International Symposium o Cluster, Cloud and Grid Computing (CCGrid)*, pages 206–215. IEEE/ACM, May 2014.
18. K. Kritikos, J. Domaschka, and A. Rossini. Srl: A scalability rule language for multi-cloud environments. In *International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 1–9. IEEE Computer Society, 2014.
19. P. Leitner, C. Inzinger, W. Hummer, B. Satzger, and S. Dustdar. Application-level performance monitoring of cloud services based on the complex event processing paradigm. In *International Conference on Service-Oriented Computing and Applications (SOCA)*, pages 1–8, Dec 2012.
20. G. Li, F. Pourraz, and P. Moreaux. Psla: A paas level sla description language. In *International Conference on Cloud Engineering (IC2E)*, pages 452–457. IEEE, March 2014.
21. R. Liu and J. Lee. It incident management by analyzing incident relations. In C. Liu, H. Ludwig, F. Toumani, and Q. Yu, editors, *Service-Oriented Computing*, volume 7636 of *Lecture Notes in Computer Science*, pages 631–638. Springer, 2012.
22. T. Lorido-Botran, J. Miguel-Alonso, and J. Lozano. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of Grid Computing*, 12(4):559–592, 2014.
23. S. Meng, A. K. Iyengar, I. Rouvellou, L. Liu, K. Lee, B. Palanisamy, and Y. Tang. Reliable state monitoring in cloud datacenters. In *International Conference on Cloud Computing Technology and Science (CLOUD)*, pages 951–958. IEEE, 2012.
24. D. Moldovan, G. Copil, H.-L. Truong, and S. Dustdar. On analyzing elasticity relationships of cloud services. In *International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 447–454, Dec 2014.
25. D. Moldovan, G. Copil, H.-L. Truong, and S. Dustdar. Quelle - a framework for accelerating the development of elastic systems. In M. Villari, W. Zimmermann, and K.-K. Lau, editors, *Service-Oriented and Cloud Computing*, volume 8745 of *Lecture Notes in Computer Science*, pages 93–107. Springer Berlin Heidelberg, 2014.
26. D. Moldovan, G. Copil, H.-L. Truong, and S. Dustdar. Mela: elasticity analytics for cloud services. *International Journal of Big Data Intelligence*, 2(1):45–62, 2015.
27. D. Moldovan, H.-L. Truong, and S. Dustdar. Cost-aware scalability of applications in public clouds. In *International Conference on Cloud Engineering (IC2E)*. IEEE, April 2016 accepted.
28. V. Munteanu, A. Edmonds, T. Bohnert, and T.-F. Fortis. Cloud Incident Management, Challenges, Research Directions, and Architectural Approach. In *International Conference on Utility and Cloud Computing*, pages 786–791. IEEE/ACM, 2014.
29. A. Naskos, E. Stachtiari, A. Gounaris, P. Katsaros, D. Tsoumakos, I. Konstantinou, and S. Sioutas. Dependable horizontal scaling based on probabilistic model checking. In *International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*. IEEE/ACM, 2015.
30. T. B. Sheridan. *Telerobotics, automation, and human supervisory control*. MIT press, 1992.
31. L. Systems. Jennifer voice picking system. http://www.. lucasware.com/jennifer-voice-picking/.
32. R. Tolosana-Calasanz, J. A. Banares, C. Pham, and O. F. Rana. Resource management for bursty streams on multi-tenancy cloud environments. *Future Generation Computer Systems*, 2015.
33. D. Trihinas, G. Pallis, and M. Dikaiakos. Jcatascopia: Monitoring elastically adaptive applications in the cloud. In *International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pages 226–235. IEEE/ACM, May 2014.
34. W. Wang, B. Li, and B. Liang. To reserve or not to reserve: Optimal online multi-instance acquisition in iaas clouds. In *International Conference on Autonomic Computing (ICAC)*, pages 13–22, Berkeley, CA, 2013. USENIX.
35. R. Zabolotnyi, P. Leitner, S. Schulte, and S. Dustdar. SPEEDL – A declarative event-based language to define the scaling behavior of cloud applications. In *World Congress on Services (SERVICES)*, pages 71–78, 2015.

Bionotes

Dr. Georgiana Copil
TU Wien, Distributed Systems Group,
Information Systems Institute,
1040 Vienna, Austria
e.copil@dsg.tuwien.ac.at

Georgiana Copil is a university assistant at the Distributed Systems Group at TU Wien. Her research interests include cloud computing, elastic computing, and Internet of Things.

Dr. Daniel Moldovan
TU Wien, Distributed Systems Group,
Information Systems Institute,
1040 Vienna, Austria
d.moldovan@dsg.tuwien.ac.at

Daniel Moldovan is a project assistant at the Distributed Systems Groups at TU Wien. His research interests include self-adaptive distributed systems, cloud computing, elastic computing, and enterprise and large scale software architectures.

Priv.-Doz. Dr. Hong-Linh Truong
TU Wien, Distributed Systems Group,
Information Systems Institute,
1040 Vienna, Austria
truong@dsg.tuwien.ac.at

Hong-Linh Truong is an assistant professor in the Distributed Systems Group at TU Wien. His work focuses on service engineering analytics, cloud computing, service-oriented architectures and computing, distributed and parallel computing, Internet of Things, complex and elastic distributed systems, and context-aware computing.

Univ. Prof. Dr. Schahram Dustdar
TU Wien, Distributed Systems Group,
Information Systems Institute,
1040 Vienna, Austria
dustdar@dsg.tuwien.ac.at

Schahram Dustdar is a full professor of computer science (informatics) and he heads the Distributed Systems Group at TU Wien, Austria. His work focuses on Distributed Systems. Prof. Dustdar is an IEEE Fellow, a member of the Academia Europaea, an ACM Distinguished Scientist, and recipient of the IBM Faculty Award 2012.