MELA: elasticity analytics for cloud services

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

Distributed Systems Group,
Vienna University of Technology,
Argentinierstrasse 8/184-1, A-1040 Vienna, Austria
Email: d.moldovan@dsg.tuwien.ac.at
Email: e.copil@dsg.tuwien.ac.at
Email: truong@dsg.tuwien.ac.at
Email: dustdar@dsg.tuwien.ac.at
*Corresponding author

Abstract: While cloud computing has enabled applications to be designed as elastic cloud services, there is a lack of tools and techniques for monitoring and analysing their elasticity at multiple levels, from the service level to the underlying virtual infrastructure. In this paper, we focus on monitoring and evaluating elasticity of cloud services, crucial for supporting users and automatic elasticity controllers, to understand the services' behaviour, and to develop smarter mechanisms for controlling their elasticity. We define novel concepts, namely *elasticity space* for describing the elastic behaviour of cloud services, and *elasticity pathway* for characterising the service's evolution through the elasticity space. We introduce techniques for enriching monitoring information and determining the elasticity space and pathway. Based on the above, we introduce MELA, an elasticity analytics as a service, providing features for monitoring and analysing the elasticity of cloud services in multi-cloud environments. To illustrate our approach, we conduct several experiments on an elastic data-as-a-service for a cloud-based machine-to-machine (M2M) platform.

Keywords: elastic computing; cloud service; elasticity analytics; monitoring.

Reference to this paper should be made as follows: Moldovan, D., Copil, G., Truong, H-L. and Dustdar, S. (2015) 'MELA: elasticity analytics for cloud services', *Int. J. Big Data Intelligence*, Vol. 2, No. 1, pp.45–62.

Biographical notes: Daniel Moldovan is a Research Assistant at the Distributed Systems Group, Institute of Information Systems, Vienna University of Technology, where he is working towards his PhD in the Context of Monitoring and Analysing Elastic Cloud Services since 2013. He received his Master in Distributed Systems and Computer Networks, and Bachelor in Computer Science at Technical University of Cluj Napoca, Romania. His research interests include distributed, green and elastic computing.

Georgiana Copil is a University Assistant at the Distributed Systems Group, Institute of Information Systems, Vienna University of Technology, where she is working towards her PhD in the context of controlling elastic cloud services since 2013. She received her Master in Computer Vision and Artificial Intelligence, and Bachelor in Computer Science at Technical University of Cluj Napoca, Romania. Her research interests include artificial intelligence, distributed and cloud computing.

Hong-Linh Truong is an Assistant Professor for Service Engineering Analytics at the Distributed Systems Group, Institute of Information Systems, Vienna University of Technology. He received his Habilitation in Practical Computer Science, from Vienna University of Technology, and a PhD in Computer Science from the same university. His research interests focus on understanding of performance, context, and data quality metrics associated with distributed and parallel applications, services and systems through monitoring and analysis, and on utilising these metrics for the design, adaptation and optimisation of these applications, services and systems.

Schahram Dustdar is a Full Professor of Computer Science and Head of the Distributed Systems Group, Institute of Information Systems, at the Vienna University of Technology. He is an ACM Distinguished Scientist and IBM Faculty Award recipient. He received his Habilitation degree in Computer Science at Vienna University of Technology, and received his MSc and PhD in Business Informatics from the University of Linz, Austria. His research interests include service-oriented architectures and computing, cloud and elastic computing, and complex and adaptive systems.

This paper is a revised and expanded version of a paper entitled 'MELA: monitoring and analyzing elasticity of cloud services' presented at 5th International Conference on Cloud Computing, CloudCom, Bristol, UK, 2–5 December 2013.

1 Introduction

With the rising cloud popularity, the number of applications and systems born in or migrated to cloud environments has substantially increased. In this context, effort has been paid for the development of emerging *elastic cloud services*, which scale up/out when the workload is high, and scale back in/down when possible, reducing cost while maintaining performance and quality. Going beyond the traditional 'elastic scalability', which concentrates on scaling in/out resources to achieve performance, such as in Mao and Humphrey (2013) or Han et al. (2012), in general, elasticity has three main dimensions: 'resource elasticity', 'cost elasticity', and 'quality elasticity', described in Dustdar et al. (2011). Thus, an *elastic service* can cope with changing external factors by providing means of reconfiguring its cost, quality and resources.

Supporting such multi-dimensional elasticity is challenging, especially how to monitor and evaluate cloud service's elastic behaviour, determine the proper cost and quality indicators and their boundaries, and utilise them for optimising and controlling the services' *elasticity*. While existing monitoring and analysis tools can present metrics related to performance, cost, or resource usage of the whole cloud service as in Singh et al. (2010), or Trihinas et al. (2014), or from the underlying virtual infrastructure as in Wang et al. (2011) and Meng et al. (2012), they do not provide a cross-layered, multi-level service elasticity behaviour picture, thus hindering the discovery of the cause for service requirements violations. Moreover, currently, deciding cost and quality indicators and their boundaries is a difficult task. Managing *elasticity* of cloud services would benefit from a multi-level monitoring and analysis view, which connects the service level behaviour with the virtual infrastructure behaviour and provides means for reasoning about the service behaviour at multiple levels. In particular, we argue that in order to understand elastic cloud services, we need to investigate new concepts that can be used to characterise the cloud service's elastic behaviour based on multi-dimensional monitoring data.

In this paper, we introduce the concepts of *elasticity space* and *elasticity pathway*, and apply them in evaluating elasticity of cloud services. First, the elasticity space is used in capturing the elastic behaviour of cloud services. Second, the elasticity pathway characterises the service's evolution through the elasticity space, and can be used to predict the service's behaviour. We further introduce a mechanism for constructing multi-level service monitoring snapshots, over which we apply our techniques for determining the elasticity space and pathway. The introduced concepts and techniques are implemented in MELA, an 'elasticity analytics as a service'. MELA allows cloud service developers, providers and automatic controllers to analyse their service behaviour from the whole service level to the underlying virtual infrastructure, extracting characteristics and providing crucial insights in their elasticity. The main contributions of our paper are:

- A model, domain-specific language and customisable mechanism for constructing multi-level monitoring snapshots for elastic services.
- Novel concepts of *elasticity space* and *elasticity pathway* for analysing elasticity of cloud services at multiple levels.
- Customisable mechanisms for extracting runtime boundaries of cloud service's elasticity that fulfil user-defined elasticity requirements.

This paper has substantially revised and expanded our initial work in Moldovan et al. (2013). We have extended the paper with a discussion on elastic cloud services (Section 2). We have substantially revised our concept of elasticity space and pathway (Section 3), improved the service representation model, and introduced an XML format for representing elastic cloud services. We have further added an XML format for our metric composition language, and defined the processes of creating new metrics and composing cost. Moreover, while the initial paper contained a single more abstract algorithm, new algorithms have been introduced, for cross-layer metric composition, and evaluating the elasticity space and pathway. We extended the description of MELA's components and its RESTful API (Section 5), and expanded our experiments to cover the analysis of elasticity space and pathway for all major units of an elastic cloud service (Section 6).

The rest of this paper is structured as follows. Section 2 presents the motivation and research problems. Section 3 presents our concepts of elasticity space and pathway for describing elasticity of cloud services. Section 4 presents our approach to monitoring and analysing elasticity of cloud services. In Section 5, we describe the MELA framework. Section 6 presents our prototype and experiments. We discuss related work in Section 7. Section 8 concludes this paper and outlines the future work.

2 Motivation and research problems

Let us consider a realistic data-as-a-service (DaaS) for an machine-to-machine (M2M) cloud platform, for which we have elasticity requirements defined by the cloud service provider, developer, and elasticity controller, w.r.t service run-time performance and cost. The DaaS provides data storage and exchange services for M2M platforms, such as smart cities or vehicle fleets, which use their M2M gateways for sending data to the DaaS. The DaaS is

composed of a message-oriented middleware, a data storage for gathering/storing M2M data, and event processing services which interact with the stored data.

Considering a non-elastic implementation of the DaaS, it would contain a powerful `Event Processing` instance interacting with a `Data Node`. Let is assume that the service could process up to 1,000 clients per second, after which its performance decreases. Thus, when the number of clients is low, the service's virtual cloud resources would be underutilised, but still paid in full, leading to unnecessary high cost. On the other hand, when the number of clients' increases beyond 1,000, the DaaS performance would decrease, and even if the service owner might want to pay more to increase performance, the only solution would be to instantiate the service on larger virtual resources.

In contrast, an elastic DaaS (Figure 1) would be designed with elasticity capabilities, i.e., run-time reconfiguration options such as ability to add/remove service units behind a load balancer. Thus, the DaaS has two service topologies (logical groupings of service units), `Data End` and `Event Processing`, having as elasticity capabilities addition and removal of service unit instances running on VMs. For designing such capabilities, the DaaS has a `Load Balancer` enabling addition/removal of `Event Processing` instances. Similarly, the `Data End` is distributed, having a `Data Controller` acting as data access load balancer, enabling addition/removal of `Data Node` instances.

Thus, the above-mentioned elastic DaaS can start with a lighter initial configuration, and lower cost. When the number of clients increases at T_1, another `Event Processing` instance can be added to cope with the load, and thus, increase the cost of running the DaaS. At time T_2, after the number of clients decreases, the additional `Event Processing` instance can be terminated, reducing cost. If required, at T_3, due to rising number of clients, multiple `Event Processing` and `Data Node` instances can be added/removed.

In general, we have three main views from which cloud services are described (Figure 2):

1 design-time, where we see the whole service dependency model (cloud service, service topology, and service unit) and user-defined requirements

2 run-time, where instances of service units are deployed and executed in virtual machines

3 the virtual infrastructure, where several virtual machines, possibly grouped in virtual clusters, are used.

While using various monitoring techniques, such as Trihinas et al. (2014) or Katsaros et al. (2012), we can capture monitoring data from the whole service level or the VM level, however, they do not answer the following crucial questions:

- What should be the behaviour of the service topologies and units when fulfilling user-defined elasticity requirements?
- When is the service's behaviour elastic, i.e., adapting and fulfilling user-defined elasticity requirements?
- How does the service's elastic behaviour evolve in time, i.e., what are the correlations and patterns in its behaviour?

Figure 1 Elastic cloud service control (see online version for colours)

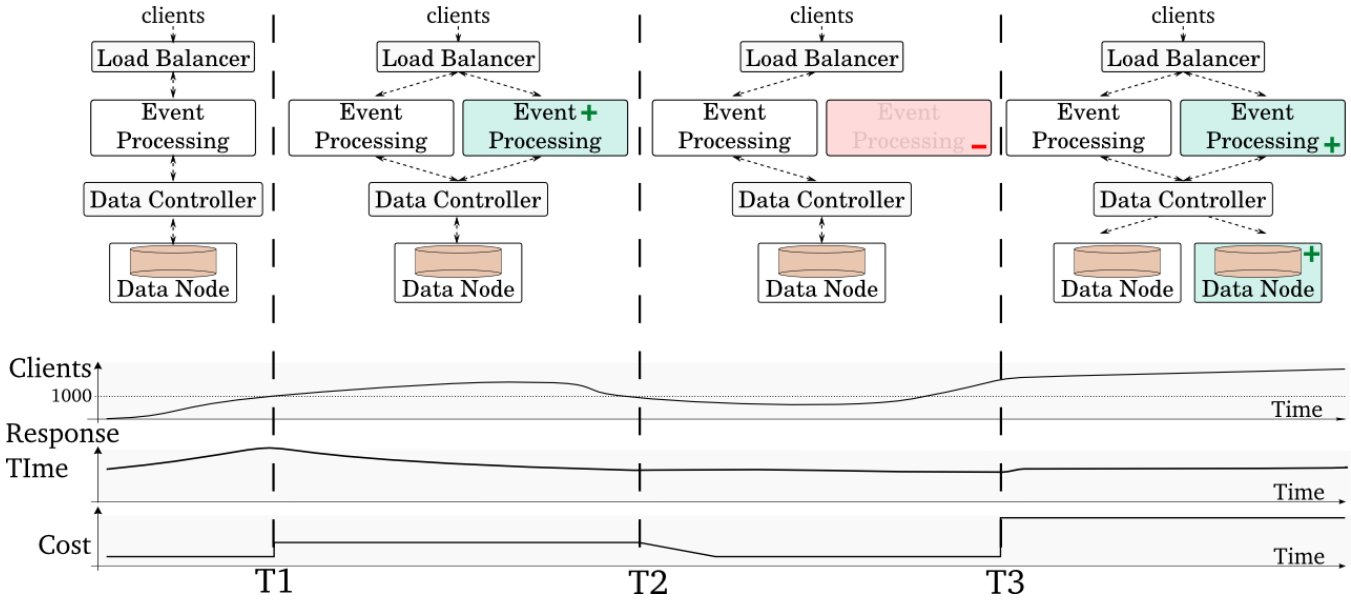
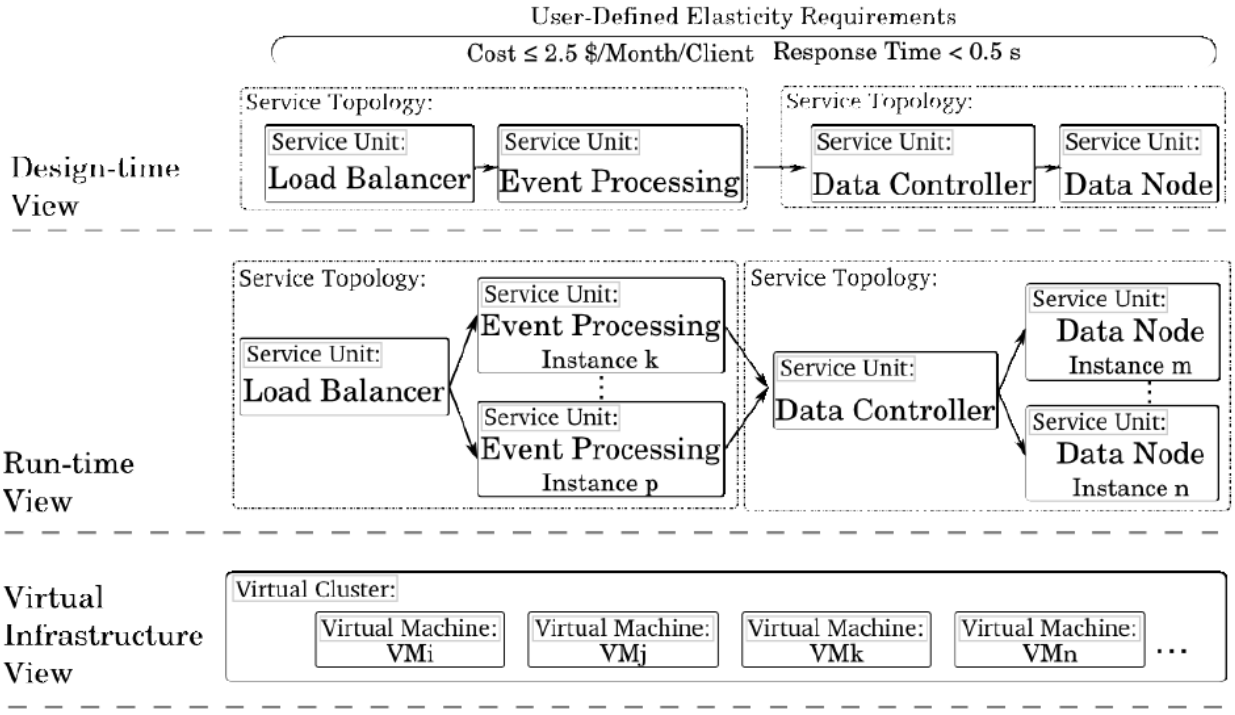


Figure 2 Elastic cloud service views



Capturing, describing and analysing the elastic behaviour of cloud services are crucial not only for developers who build and optimise them, but also for software controllers that change the services' topology at run-time, enforcing user-defined elasticity requirements. Thus, elasticity controllers, such as Copil et al. (2013) or Mao and Humphrey (2013), need a mechanism for extracting elasticity characteristics, used to refine user-defined elasticity requirements or predict the service behaviour, leading to better service control and quality.

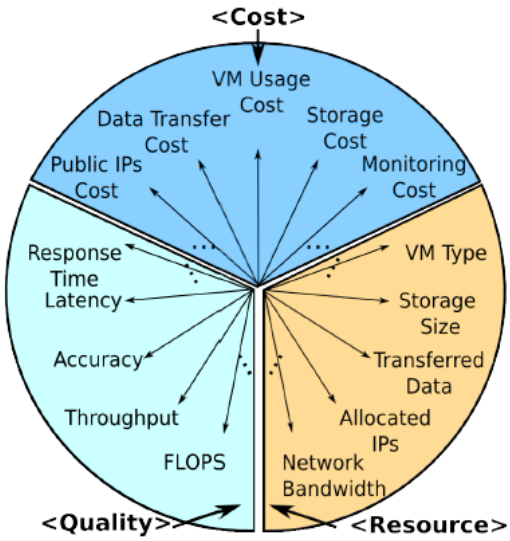
To analyse the behaviour of elastic cloud services, we must first collect monitoring information, map it to the service structure, and extract from it higher level information used to characterise the service's elasticity. While determining elasticity of a monitored element (e.g., load balancer and data node) is already challenging, we cannot just deal with single instances. We also need to analyse the elasticity of the whole cloud service by examining dependencies among different monitored elements and the virtual infrastructure hosting them. Furthermore, we need to map the service's elasticity behaviour to user-defined elasticity requirements, to analyse how and when the requirements are fulfilled.

This motivates us to investigate the following issues:

- Which concepts can be used to capture the elastic behaviour of cloud services?
- How to extract characteristics that describe the service's elastic behaviour to support both reactive and predictive control of elastic services?
- How to analyse the cloud service's behaviour, detecting the source of user-defined elasticity requirements violations?

This paper focuses on capturing the properties of elastic services at multiple levels, providing support for analysing their behaviour from multiple views, and characterising the elastic behaviour of each cloud service based on user-defined elasticity requirements.

Figure 3 Elasticity dimensions (see online version for colours)



3 Elasticity space and pathway of cloud services

In this section, we introduce the concepts of elasticity space and elasticity pathway for analysing and characterising the elastic behaviour of cloud services.

3.1 Multi-dimensional elasticity

In order to support and analyse the multi-dimensional elasticity of cloud services defined in Dustdar et al. (2011), we categorise monitoring data in three dimensions: *cost*, *quality*, and *resource* (Figure 3). These categories are sufficient for capturing data about any monitored element (e.g., service topology or service unit) within a cloud service, and can be used for understanding the elasticity of that service. For an elastic cloud service, the *quality* dimension would capture metrics characterising the service'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 would capture resource usage and allocation information, such as the amount of data transferred over the network.

Conceptually, to capture monitoring data associated with a monitored element at a specific time t, we define the monitoring snapshot, *ms*, containing monitoring data about *cost*, *quality* and *resource* elasticity dimensions [equation (1)].

$$ms = \left\{ \left(\langle c_i \rangle, \langle q_j \rangle, \langle r_k \rangle, t \right) \middle| c_i \in Cost, \; q_j \in Quality, r_k \in Resource \right\} \tag{1}$$

3.2 Elasticity boundary

Monitoring snapshots capture metrics, but do not provide information about boundaries over the metric's values in which user-defined elasticity requirements are fulfilled. Therefore, in order to analyse the elastic behaviour of a monitored element, we represent metric boundaries using the *elasticity boundary* concept:

Definition 1: An *elasticity boundary* describes the upper and lower bound over a set of metrics for a monitored element.

Conceptually, an elasticity boundary, *elBoundary*, is defined as follows:

$$elBoundary = \left(\langle c_i^u, c_i^l \rangle, \langle q_j^u, q_j^l \rangle, \langle r_k^u, r_k^l \rangle \right) \tag{2}$$

where c_i^u and c_i^l denote the upper bound and the lower bound of metric $c_i \in Cost$, respectively, q_j^u and q_j^l for $q_j \in Quality$, and r_k^u and r_k^l for $r_i \in Resource$.

We use the elasticity boundary to capture both user-defined elasticity requirements (*user-defined elasticity boundary*), and detected/evaluated elasticity requirements (*evaluated elasticity boundary*). Using the user-defined elasticity boundary we represent requirements over the user's elastic service's cost, quality and resources. From the user-defined requirements expressed as user-defined elasticity boundaries indicating parameters c_i, g_j, r_k under which the cloud service should behave, we evaluate collected monitoring information and determine elasticity boundaries for all monitored elements of a cloud service. Thus, from a set of supplied requirements for a particular monitored element (e.g., cloud service and service unit), we determine the requirements for all cloud service monitored elements, providing information to control the elasticity of each element of the cloud service.

3.3 Elasticity space

Given a set of monitoring snapshots and user-defined elasticity boundaries, for supporting run-time control of service's elasticity, we need to understand when a monitored element is in elastic behaviour, if its behaviour violates the user-defined elasticity boundaries, and if we can characterise the service behaviour using some specific 'pathways'. Naturally, we expect that the meaning of 'elasticity' will depend on the types of monitored elements, their runtime settings and requirements. To this end, we define the concept of elasticity space to determine and evaluate when a monitored element is in elastic behaviour:

Definition 2: An elasticity space captures all runtime metrics described in the user-defined elasticity boundary and all other metrics influencing the user-defined elasticity boundary, when a monitored element is in elastic behaviour, which is determined via an *elasticity space function.*

To respect its elasticity boundaries, an elastic service must scale out/in its cost, quality and resources at run-time, by allocating/deallocating cloud services, to cope with variations in pricing, quality and load. We refer to the behaviour of a service which is dynamically reconfigured at run-time by software controllers as elastic behaviour. Formally, let $f_{elSpace}$ be an elasticity space function, $MS = \{ms_i\}$ 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:

1 detect when an elastic behaviour starts and stops

2 extract monitoring data describing the service behaviour 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, such as specific types of service units, topologies, or the whole service. Furthermore, these functions are applied on the metrics from user-defined elasticity boundaries.

An elasticity space function is designed to extract useful information about the overall behaviour of the cloud service when elasticity requirements are fulfilled. For example, given a user-defined elasticity requirement over *serviceCost*/*client*/*h*, an elasticity space might contain only the *throughput* and *cost*/*VM*/*h* metrics from which the *serviceCost*/*client*/*h* targeted by requirements can be determined, not including metrics that have no impact on it. 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 analyse if the behaviour of an elastic cloud service is within expected user-defined elasticity boundaries by checking the elasticity boundaries of its elasticity space. For example, the upper elasticity boundary of the *serviceCost*/*client*/*h* from the determined elasticity space could have a different value than expected by the user.

3.4 Elasticity pathway

While the elasticity space enables cloud service elasticity analysis, it does not provide insight into relationships and dependencies between metrics influencing the elastic behaviour over time, e.g., *throughput* and *cost*/*VM*/*h* might or might not follow a linear relationship. In order to characterise the elastic behaviour from specific views/perspectives over a cloud service, we define the concept of *elasticity pathway*.

Definition 3: Given a specific view on metrics $V = \{m_1, m_2, \cdots, m_n\}$, an elasticity pathway for V characterises the elasticity relationship among mi over the time.

A view over a set of metrics is a subset of metrics chosen for analysis, which potentially influence the user-defined requirements. For example, given a user-defined elasticity requirement over *serviceCost*/*client*/*h*, a view could contain the *throughput* and *cost*/*VM*/*h* metrics from which the *serviceCost*/*client*/*h* can be derived. An elasticity pathway function is designed to perform a complex evaluation of the cloud service behaviour, determining characteristics that can be used to predict the service's behaviour.

Formally, an elasticity pathway e_{lPtw} is determined by a function f_{elPtw} which takes as input an elasticity space *elSpace* and a view V over the space's metrics, and returns another function describing behavioural patterns or characteristics of the monitored element: $elPtw = g(V) = f_{elPtw}(elSpace, V)$. Various elasticity pathway functions can be defined over the elasticity space, enabling space analysis from multiple perspectives. As the elasticity pathway function is applied over an elasticity space, the quality of the determined elasticity pathway is heavily influenced by the data in the elasticity space.

4 Monitoring and analysing elasticity of cloud services

In order to monitor and evaluate the elasticity space and pathways of a cloud service, we need to:

1 build the cloud service dependency model

2 build the monitoring snapshot for all monitored elements

3 apply particular elasticity space and pathway functions over the monitoring snapshots.

4.1 Elastic cloud service dependency model

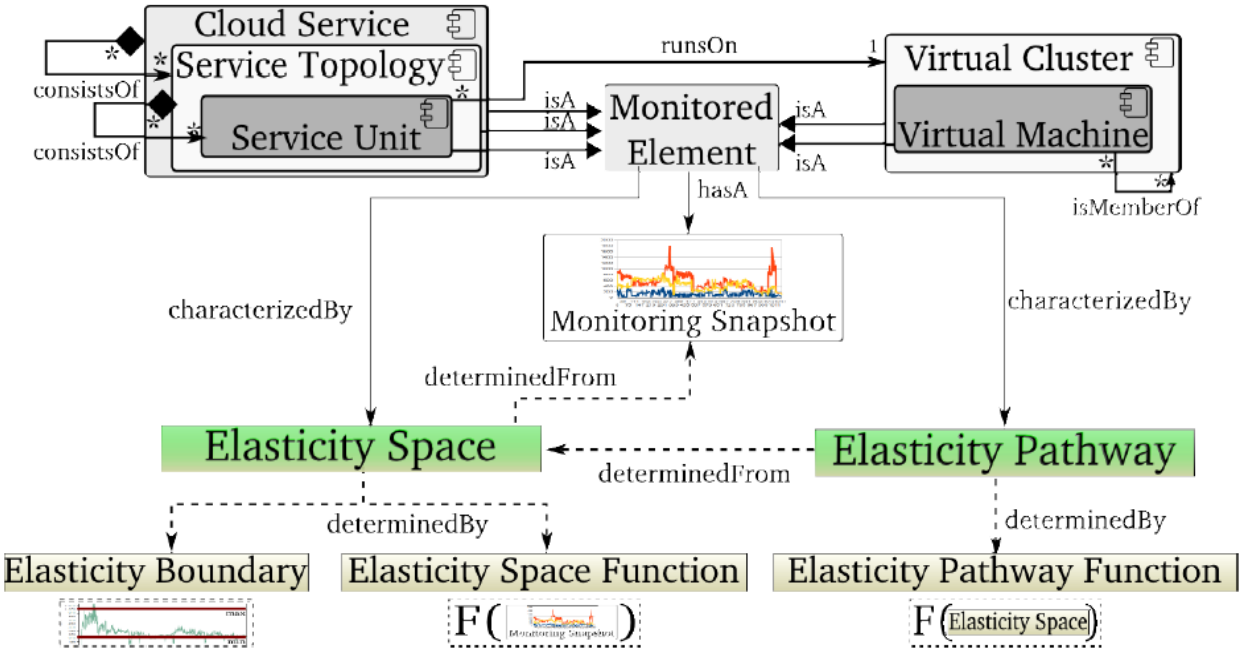
For describing elastic services from the whole service level to the underlying virtual infrastructure, we need an abstract representation model that enables the description of cloud services and their elasticity requirements at multiple levels. To support multi-level analysis of service behaviour, we represent a cloud service composed of service topologies, each topology containing service units, deployed on virtual machines belonging to virtual clusters (Figure 4). A *service unit* is a functional element of a cloud service that runs inside a virtual machine, either standalone or along other service units. A *service topology* does not have an equivalent in the virtual cloud infrastructure, instead it logically groups related service units, e.g., a `Data End`. Instances of units run on *virtual machines* belonging to *virtual clusters* (e.g., network clusters, or different cloud providers). Thus, our model can represent cloud services which are composed of other services (service topologies), deployed in federated cloud environments or in different availability zones (virtual clusters).

The dependency model of the cloud service is extracted or defined by users/controllers, and used for structuring monitoring information and analysing the service's behaviour. For easy integration with other tools, we use an XML-based representation (Listing 1). In general, each monitored element is defined with the `MonitoredElement` tag, and contains two mandatory properties, id, and level, and an optional name property. The `level` indicates if the monitored element is one of VM, `VIRTUAL_CLUSTER`, `SERVICE_UNIT`, `SERVICE_TOPOLOGY`, or `SERVICE`, and each monitored element can contain zero or more monitored elements with a lower level. For the `VM` level, the `id` is the IP of each virtual machine. At run-time, this description would be managed by a controller, and updated with the virtual machines that run service units instances, after each scaling action.

4.2 Cross-layered metric composition

Existing tools for monitoring cloud services such as Ganglia (http://ganglia.sourceforge.net/) or Trihinas et al. (2014) are agnostic of the service's logical structure and associate monitoring information with the individual virtual machines running service units' instances. As one unit can have multiple instances distributed among different VMs, associating information with VMs does not give any indicator about the overall behaviour of the service units, information crucial for elasticity controllers. Therefore, to obtain a complete view over the cloud service behaviour, from low level metrics to higher ones, we develop techniques for cross-layer composition of monitoring snapshots, elasticity spaces and elasticity pathways, following the previously presented cloud service model (Figure 4).

Figure 4 Associations between elasticity space/pathway and monitored elements (see online version for colours)



Listing 1 XML-based dependency model

```
1  <MonitoredElement id = ""level="SERVICE">
2      <MonitoredElement id =
       ""level="SERVICE_TOPOLOGY">
3          <MonitoredElement id =
           ""level="SERVICE_UNIT">
4              <MonitoredElement id = "10.x.x.x" level =
               "VM" />
5      </MonitoredElement>
6  …
```

Table 1 Metric composition operations

Operation	*Description*
+, –	Adds/Subtracts a value (metric or static) to another value (from another operation or metric)
*, /	Multiplies/Divides a value (from another operation or metric) with another value (metric or static)
AVG	Computes the average value of a sequence of metric values
SUM	Computes the sum of a sequence of metric values
MAX, MIN	Extracts the maximum/minimum value from a sequence of metric values
SET	Assigns a static value to a metric
KEEP	Returns unchanged a metric value or the result of another operation

4.2.1 Metric composition language

Monitoring snapshot composition introduces the problem of combining/aggregating metrics. Depending on the type of metrics, a valid composition of two metrics might involve different operations. We define an XML-based domain-specific language for describing metric composition rules as a cascading sequence of operations which apply one or more operators over one or more operands. An operand can be a static value or a metric reference, and the set of available operations is presented in Table 1. The language grammar is shown in Listing 2, and the XML format for specifying rules is shown in 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 *TargetMonitoredElementLevel*. The *resulting metric* defines the name and unit of the composite metric being created. Defining the composition rules requires domain specific knowledge, such as knowing which operation is appropriate for which metric (e.g., SUM for cost, AVG for response time).

4.2.2 Monitoring information structuring and enrichment

Monitoring snapshots capture metrics, but not the same as elasticity requirements defined by the user, or at the same level. For example, a monitoring snapshot might include `throughput` per VM, while the elasticity requirements might target `numberOfClients` over the whole service. Using the above metric composition language, metrics collected from the VM level can be associated to the upper service levels, and new metrics can be created, according to individual service requirements, applying composition operations, as follows. First, the service level for which the new metric is created should be specified using the `TargetMonitoredElementLevel`, and can be one of `VM`, `VIRTUAL_CLUSTER`, `SERVICE_UNIT`,

`SERVICE_TOPOLOGY`, or `SERVICE`, according to the service dependency model. Optionally, if the rule should be applied on a specific monitored element, (a unit, a topology or the whole service), its `ID` can be specified with the `TargetMonitoredElementID` tag. Information about the new metric to be created must be specified using the `ResultingMetric` tag, including name, measurement unit, and type (one of elasticity dimensions). Then, a cascading list of operations can be defined, each operation having a `type`, defined in Table 1, and a `ReferenceMetric`, indicating the metric over which the operation should be applied. If the operation should be applied over metrics from a specific monitored element, its ID can be specified with the `SourceMonitoredElementID` tag.

Listing 2 Metric composition rules grammar

```
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 XML-based metric composition rule format

```
1  <CompositionRule TargetMonitoredElementLevel = "LEVEL">
2     <TargetMonitoredElementID>id</TargetMonitoredElementID>
3     <ResultingMetric type = "METRIC_TYPE" measurementUnit = "text" name = "text" />
4     <Operation MetricSourceMonitoredElementLevel = "LEVEL" type = "OPERATION">
5        <ReferenceMetric type = "METRIC_TYPE" measurementUnit = " text " name = " text " />
6        <SourceMonitoredElementID>ID</SourceMonitoredElementID>
7     </Operation>
8  </CompositionRule>
```

Listing 4 Example of cost composition

```
1   <CompositionRule TargetMonitoredElementLevel = "VM">
2      <ResultingMetric type = "RESOURCE" name = "numberOfVMs" measurementUnit = "count" />
3      <Operation value = "1" type = "SET_VALUE" />
4   </CompositionRule>
5
6   <CompositionRule TargetMonitoredElementLevel = "SERVICE_TOPOLOGY">
7      <ResultingMetric type = "COST" measurementUnit = "$/ client /h" name = "cost/client/h" />
8      <TargetMonitoredElementID>TOPOLGY_ID</TargetMonitoredElementID>
9      <Operation type = "DIV">
10        <Operation type = "MUL" value = "VM_COST">
11           <Operation MetricSourceMonitoredElementLevel = "VM" type = "SUM">
12              <ReferenceMetric type = "RESOURCE" name = "numberOfVMs" />
13           </Operation>
14        </Operation>
15
16        <Operation MetricSourceMonitoredElementLevel = "SERVICE_UNIT" type = "KEEP">
17           <ReferenceMetric type = "RESOURCE" measurementUnit = "clients/s" name = " clients " />
18           <SourceMonitoredElementID>LoadBalancer</SourceMonitoredElementID>
19        </Operation>
20     </Operation>
21  </CompositionRule>
```

Algorithm 1 Cross-layered metric composition algorithm

```
Input: service, compositionRules, monitoringData
Output: snapshot – multi layer monitoring snapshot
1:  function BuildMonitoringSnapshot(service, compositionRules, monitoringData)
2:      snapshot = createInitialSnapshot(monitoringData)
3:      for all elementLevel in [VM,ServiceUnit, ServiceTopology, Service] do
4:          levelElements = service.getElementsOnLevel(elementLevel)
5:          levelRules = compositionRules.getRulesOnLevel(elementLevel)
6:          for all rule in levelRules do
7:              targetElements = getRuleTargets(rule, levelElements)
8:              for all element in targetElements do
9:                  compositionValue= ApplyCompositionRule(rule, element, snapshot)
10:                 snapshot.addNewMetric(rule.metric,compositionValue, element)
11:             end for
12:         end for
13:     end for
14: return snapshot
15: end function
16:
17: function ApplyCompositionRule(rule, monitoredElement, snapshot)
18:     sourceElements = getSourceElements(rule.metric, monitoredElement)
19:     values = []
20:     for all element in sourceElements do
21:         values.add(snapshot.getMetricValue(element,rule.metric))
22:     end for
23:     for all operation in rule.operations do
24:         values = operation.apply(values)
25:     end for
        return values
26: end function
```

4.2.3 Cost composition

Monitoring and analysing cost is crucial in enabling run-time monitoring, analysis and control of service cost by elasticity controllers. Cost usually cannot be monitored directly, due to a lack of cloud provider APIs providing run-time fine-grained cost monitoring. Instead, using metric composition rules, we can associate cost information usually collected statically from cloud providers, with monitoring information collected by monitoring tools, obtaining real-time cost monitoring. Using our composition language, in Listing 4, we show a template for composing the `cost/client/h` (line 7) for a certain topology (line 8), by enriching monitoring information with cost information and new metrics. For computing the virtual infrastructure cost, a new `numberOfVMs` metric is created with value 1 for each VM belonging to this topology (lines 1–4). Then, the values of `clients` metric obtained from all topology's VMs are summed up (lines 11–13), and multiplied (lines 10–14) with the cost per VM, `VM_COST`. Lines 16–19 define an operation retrieving the `clients` metric from a specific `LoadBalancer` unit, used to divide (lines 9–20) the cost of running the virtual infrastructure obtained above. Starting from this template, other composite cost metrics can be defined according to specific service requirements.

4.2.4 Metric composition process

Algorithm 1 describes our process of constructing multi-level monitoring snapshots by applying custom metric composition rules which structure monitoring information, compute cost, or enrich monitoring information depending on service specific requirements. The *BuildMonitoringSnapshot* function applies metric composition rules bottom-up on the service's monitored elements, first at the *virtual machine* level, then at *service unit*, *service topology* and *service levels*, creating the cross-layer monitored snapshot. Each composition rule is applied over its target monitored elements belonging to that specific level (lines 6–12). The *ApplyCompositionRule* function handles the application of metric composition

rules. First, if a rule specifies a specific monitored element from which the metric values are to be collected, it is considered as a single monitored values data source. Otherwise, monitored values are extracted from the element itself, or from all children of the element which belong to the rule's metric *source level* (line 19). From all monitored element data sources, the source metric is searched, and its values collected (lines 20–22). Over the collected values, the rule's composition operations are applied sequentially, the output of a previous operation acting as input for the next (lines 24–26). The output of the last operation is returned as result of the metric composition process.

4.3 Multi-dimensional elasticity space and pathway analysis

Based on the metric composition, we provide an analysis mechanism for determining the elasticity space and pathway at different dependency model levels, providing a multi-level decomposition of the cloud service behaviour. Such a decomposition is beneficial to elasticity controllers, which can use our approach to detect which monitored element from which service dependency model level violates service requirements, and reason in terms of metrics located at that particular level. Providing a complete view over the behaviour of cloud services enables service providers/developers and software controllers to reason on the same service using separate perspectives.

4.3.1 Elasticity space analysis

Algorithm 2 describes the process of evaluating the elasticity space, in which custom elasticity space functions can be used to update the elasticity boundaries (`updateElasticityBoundaries` function). The service structure, elasticity requirements, metric composition rules, and collected monitoring data are received as input. For each new collected monitoring snapshot, the cross-layered enriched monitoring snapshot is computed by applying metric composition rules in line 2. For each service monitored element, the elasticity space is evaluated by updating the elasticity boundaries. Lines 7–11 check if all elasticity requirements are fulfilled, and if yes, the upper and lower elasticity boundaries are updated. If not, the monitoring snapshot is stored for future reference.

We understand that analysing and controlling elasticity of cloud services can be a continuous process in which elasticity requirements or even the service structure are refined during run-time. We define the following changes that have impact on the process of determining the elasticity space and boundaries:

1 metric composition rules

2 elasticity requirements

3 service structure.

In the first case, we support addition and removal of metrics by altering the supplied composition rules, the elasticity space evaluation continuing by enriching the space determined so far using the metrics available in each new monitoring snapshot. In the latter 2 cases, the whole elasticity space is recomputed based on historical monitoring information, updating all elasticity boundaries.

In the current prototype, we implement an *elasticity space function* which, starting from user-defined elasticity requirements for the whole cloud service, determines as space boundaries for all service topology and service unit instances, their maximum and minimum encountered metric values when the user-defined elasticity requirements are respected.

Algorithm 2 Evaluating elasticity space

Input: *service*, *requirements*, *metricCompositionRules*, *monitoringData*

```
1:  function EvaluateElasticitySpace(service, requirements, metricCompositionRules, monitoringData)
2:      snapshot = BuildMonitoringSnapshot(service, metricCompositionRules, monitoringData)
3:      elasticitySpace = getElasticitySpaceLearnedSoFar()
4:      for all snapshot in monSnapshot.elements do
5:          elementElSpace = elasticitySpace.get(snapshot.element)
6:          elementRequirements = serviceRequirements.get(snapshot.element)
7:          if snapshot.fulfillsAll(elementRequirements) then
8:              elementElSpace = elementElSpace.updateElasticityBoundaries(snapshot)
9:          else
10:             elementElSpace = elementElSpace.store(snapshot)
11:         end if
12:     end for
13: return elasticitySpace
14: end function
```

Algorithm 3 Evaluating elasticity pathway

```
Input: elasticitySpace, serviceElement
1:  function EvaluateElasticityPathways(elasticitySpace, serviceElement)
2:      elPathwayFunctions = getPathwayFunctionsFor(serviceElement)
3:      elPathways = []
4:      for all elPathwayFunction in elPathwayFunctions do
5:          elPathways.push(elPathwayFunction(elasticitySpace))
6:      end for
7:  return elPathways
8:  end function
```

4.3.2 Elasticity pathway analysis

Based on an elasticity space, *elasticity pathway* functions can be defined for each service unit, topology, or whole service, enabling custom analysis of service behaviour. Based on the determined elasticity space, for the supplied monitored element (`service element`), we apply its elasticity pathway functions, analysing its behaviour and extracting behavioural characteristics (Algorithm 3).

For the current prototype, we adapt as elasticity pathway function an unsupervised behaviour learning technique using self-organising maps (SOMs) presented by Dean et al. (2012), and classify monitoring snapshots by their encounter rate in DOMINANT, NON-DOMINANT, and RARE. Such a pathway is important for understanding if the regular behaviour of the service respects user-defined elasticity requirements. As SOMs are unsupervised neural networks that map multi-dimensional spaces into low dimensional ones, they are suitable for classifying monitored snapshots, as snapshots can contain many different metrics. Each SOM's neuron value is derived from its snapshots. Each monitoring snapshot to be classified is mapped to the group from which it has the smallest distance. With each new snapshot, the group and its SOM neighbours 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 initialisation of the system is important. As we classify groups after the number of snapshots mapped to them, we do not initialise 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 initialise 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 neighbourhood function of 1 for the directly targeted group and of 1/*neighbourLevel*/*neighboursCount* for its neighbours. Updating the neighbours creates and update new groups, mapping the input data better. The discount learning factor is 1/*neighbourLevel*, the neighbourhood is 2 and the map size is 10 × 10. A filtering step merges groups with same value, consolidating the monitored snapshots, and the average absolute deviation (AAD) (the average of the absolute deviations) of the number of snapshots mapped per group is computed. Using the AAD, a group is RARE if its deviation is negative and its absolute 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 are enough to give insight in the elastic behaviour of cloud services.

5 MELA: elasticity analytics as a service

Based on our concepts in Section 3, we develop MELA, an elasticity analytics as a service (Figure 5). MELA contains a core *MELA Service*, and *Data Collector* nodes. A *Data Collector* node is a customisable component that gathers, from existing monitoring solutions, data associated with a dependency model level or monitored element (e.g., `responseTime` or `throughput` for the `Event Processing` service topology), and sends it for processing and analysis to the *MELA Service*.

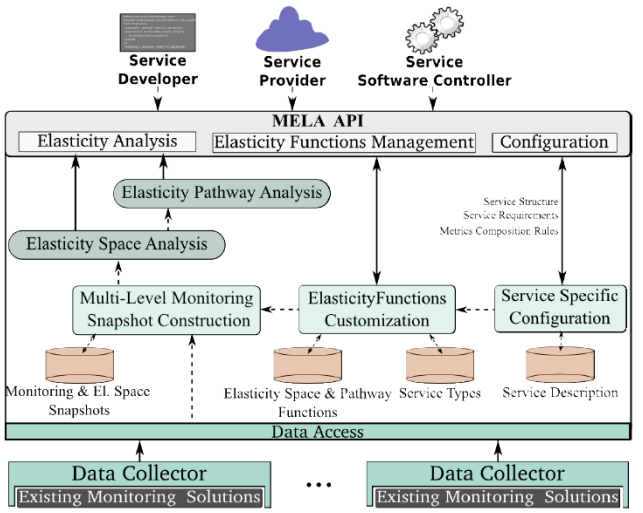
Monitoring data from existing monitoring systems is usually associated with a single level, e.g., virtual infrastructure, service topology or service unit. An important MELA feature is the linking of these levels, implying a configuration step using the *configuration API*, defining the service structure, the metrics composition rules to be applied for the monitored elements at each level, and the service requirements. The *multilevel monitoring snapshot construction* component uses the service specific configuration to provide composite monitoring snapshots from data collected from the *data collector* nodes. *Elasticity space* and pathway functions are tailored trough the *elasticity functions management API* and used to determine the elasticity space by the *elasticity space analysis* MELA component, from which the elasticity pathway is determined by the elasticity pathway analysis MELA component. Composite monitoring snapshots together with determined elasticity space are stored in a *Monitoring and elasticity space snapshots* repository. Using the *elasticity analysis API*, the MELA user can retrieve the elasticity space and pathway for the whole service, or specific monitored elements.

Table 2 MELA RESTful API

URI	*Operation*	*Functionality*
/servicedescription	PUT	Submitting the service structure
/servicedescription	POST	Updating the service structure with added or removed virtual machines
/metricscompositionrules	PUT	Submitting metric composition rules
/servicerequirements	PUT	Submitting service's elasticity requirements
/metrics	GET	Retrieving the metrics that can be collected from the service
/metriccompositionrules	GET	Retrieving the metric composition rules in JSON format
/monitoringdataJSON	GET	Retrieving the service's composite multi-level monitoring snapshot in JSON format
/monitoringdataXML	GET	Retrieving the service's composite multi-level monitoring snapshot in XML format
/elasticityspace	POST	Retrieving the elasticity space in JSON format of a supplied monitored element
/elasticitypathway	POST	Retrieving the elasticity pathway in JSON format of a supplied monitored element

As MELA is designed to analyse elastic services which can allocate/deallocate virtual machines depending on elasticity requirements, it provides two mechanisms for managing this volatile service structure. The default mechanism is to delegate this responsibility to the MELA user (cloud service developer/provider or controller), which, in this, case uses the MELA API to update the service structure after a new virtual machine running service units has been added or removed. The second behaviour is to let MELA detect automatically when a new virtual machine has been added or removed to which service units the machines belong. This behaviour is achieved by reporting the service units' ids hosted by a virtual machine using a virtual machine metric, and configuring MELA to use the metric to automatically update the service structure. MELA is released as an open-source project (http://tuwiendsg.github.io/MELA/), and MELA exposes its functionality through RESTful services, providing methods for configuring MELA and retrieving multi-level monitoring information and elastic service behaviour analysis (Table 2).

Figure 5 MELA overview (see online version for colours)



6 Experiments

We evaluate MELA in the context of COMOT, a platform-as-a-service for elasticity in the cloud (Truong et al., 2014), containing tools for describing, deploying, and controlling elastic cloud services. The realistic DaaS application for an M2M cloud mentioned in Section 2 is used as pilot service, were the DaaS uses Cassandra (http://cassandra.apache.org/) for its `Data End` units, and HAProxy (http://haproxy.1wt.eu/) for its `Load Balancer` unit. In this scenario, the M2M DaaS provider wants to implement a 2.5$ monthly subscription for each service client (sensor). UsingCOMOT, the provider describes the DaaS and its elasticity requirements, and deploys the service, which is then automatically controlled at run-time.

MELA has two important roles in this evaluation. First, it provides the human service provider the ability to monitor and analyse the service's elasticity behaviour (via metrics) at each service level, to understand the elasticity of the service, and verify if such a pricing scheme is sustainable. Secondly, it provides structured and enriched monitoring information used by COMOT's elasticity controller during service's run-time.

As elasticity is triggered by performance/cost requirements, we simulate a load from real sensor data, following a Gaussian distribution of M2M sensors connecting to the DaaS, starting from 50 sensors per second, increasing to 350, and then decreasing again, each request requiring between 1 and 10 operations. The service VMs were deployed on our OpenStack (http://www.openstack.org/) cloud. For these experiments MELA utilises Ganglia (http://ganglia.sourceforge.net/) for the MELA data collector node, retrieving generic OS level, and service specific monitoring data (e.g., clients/h, throughput, and response time) from custom Ganglia plug-ins we have developed.

6.1 Monitoring elastic cloud services

The first feature of MELA is multi-level elasticity monitoring. Using the cross-layered metric composition mechanism, the MELA user (service developer/provider) starts by defining a composition rule for extracting the cost

of running the service in cloud, customising the template provided in Section 3, assuming service cost to consist only of the cost of running each virtual machine, assumed 0.12$ per VM per hour. As the service provider is interested in the overall cost per client, additional metric composition rules are defined to first propagate the `activeConnections` metric from the virtual machines running the *LoadBalancer* service unit to the service unit level as a renamed `numberOfClients` metric, and then further propagate the `numberOfClients` to the service unit's parent *EventProcessing* service topology. Obtaining the `cost/client/h`, a *CloudService* level rule sums the `cost` of the children service topology instances, and divides it by the `numberOfClients` metric from the *EventProcessing* topology. Figure 6 shows with thick lines how, using MELA, such a complex `cost/client/h` metric is composed and evaluated by combining available metrics with additional cost information.

To monitor the performance of the DaaS, the MELA user extracts for the *DataController* service unit its `writeLatency` by averaging the *write_latency* reported by all its instances running in virtual machines. For the *EventProcessing* service unit the average `responseTime` and total `throughput` are extracted from its virtual machines, and propagated to the *EventProcessing* service topology. Additionally, a metric composition rule is defined to extract the average `cpuUsage` for all service unit instances. Figure 6 shows a snapshot of the DaaS, containing the simple and composite metrics deemed important for analysing the service's behaviour.

Providing higher-level composite metrics such as `cost/client/h`, structured after the service structure in topologies and units, MELA facilitates service behaviour analysis by the service controller, focusing on simple or composed metrics that are of interest.

Figure 6 MELA visualisation of multi-level monitoring data with complex cost composition (see online version for colours)

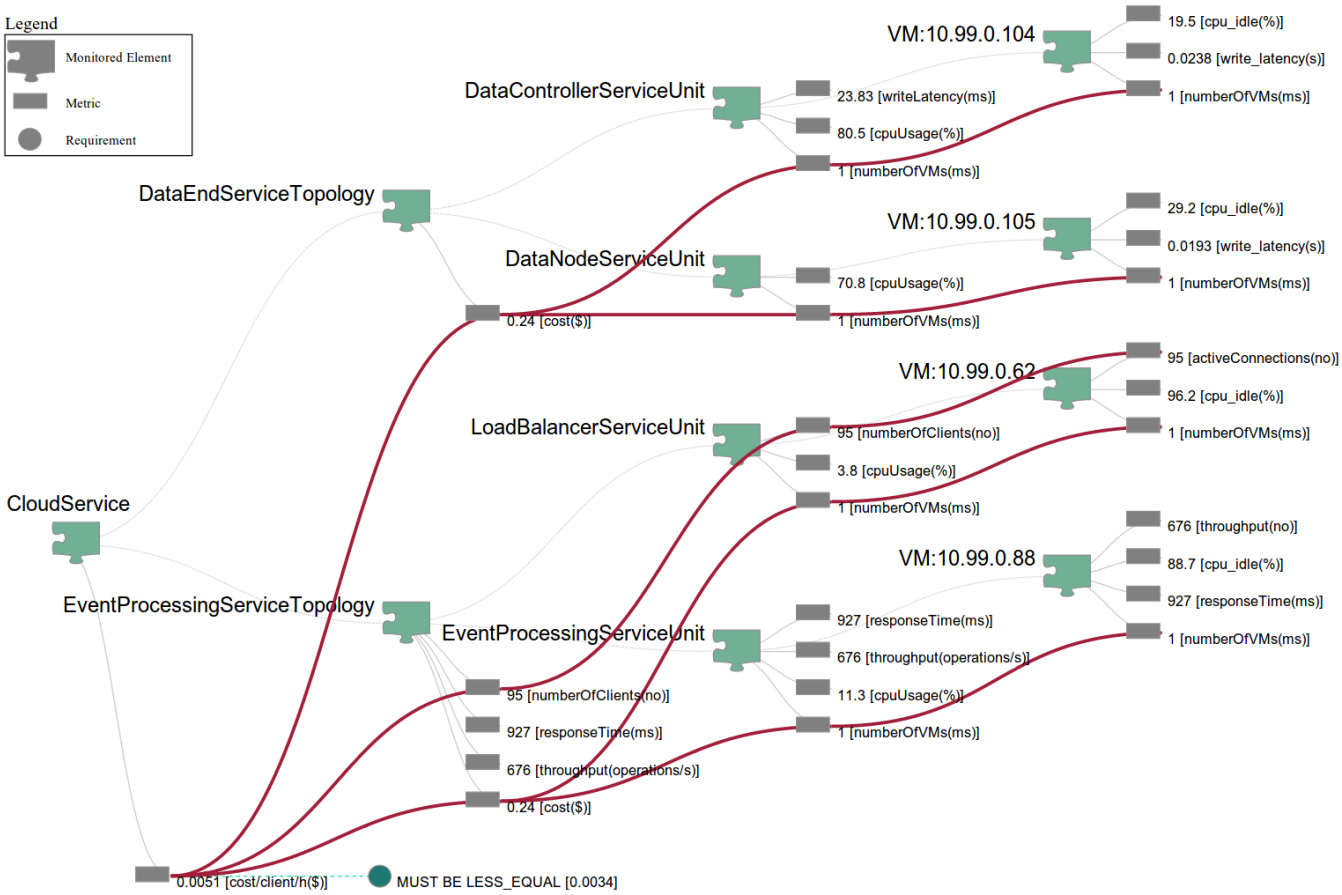
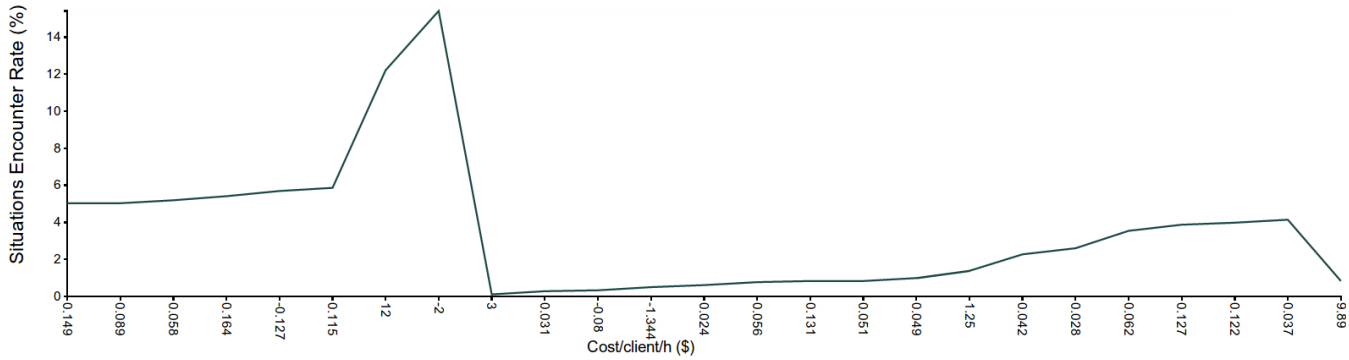


Figure 7 Cloud service elasticity pathway (see online version for colours)



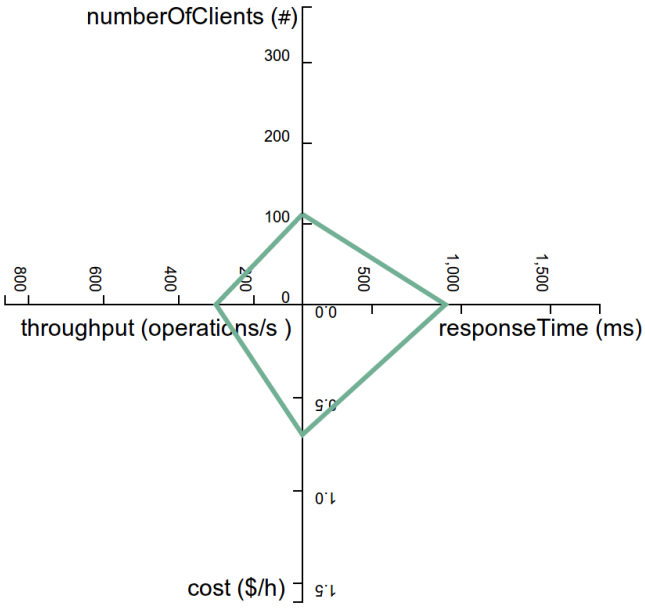
6.2 *Analysing elastic cloud services using elasticity space and pathway*

MELA's second feature is elasticity analysis. The elasticity space is used to determine what are the behavioural boundaries in which the service fulfils supplied requirements. The elasticity pathway function prototype groups combination of different metric values as DOMINANT, NON-DOMINANT, and RARE, according to their encounter rate in the monitoring data, determining the usual behaviour of the service and the correlations between the analysed metrics. In our scenario, assuming a month of 30 days, the elasticity requirement for the service is a maximum cost of 0.0034$ per served client per hour. Using MELA, the user determines what are the elasticity boundaries for various service topologies and units in which the cost requirement is respected, and the correlations between the analysed metrics. Using this information, the MELA user can validate and refine the elasticity requirements submitted to COMOT and used to control the elasticity of the service.

As the `cost` boundaries are set by the MELA user, the elasticity pathway of the service is inspected (Figure 7), revealing that the `cost/client/h` is less than 0.0034$ only in approximate 72% of the encountered situations, leading to the conclusion that a pricing scheme of 2.5$ per month per client is not fully sustainable. To find the reason for this situation, the provider uses the MELA multi-level analysis feature to focus on the event processing service topology.

Figure 8 presents a snapshot of the elasticity space for the event processing service topology containing the `numberOfClients/h`, `responseTime`, `throughput`, and `cost/h` metrics. The complete space is depicted in Figure 9 and thick lines mark the elasticity space boundaries, i.e., minimum and maximum acceptable values. For the `numberOfClients/h` elasticity space dimension (upper left corner), the determined lower elasticity space boundary is approximately 150, understandable given that the service uses at least 4 VMs at 0.12$/h (150 multiplied by 0.0034, gives a 0.51$/h). To learn more about the service topology behaviour and its boundaries, the user focuses on the `responseTime` dimension (lower right corner), for which MELA determines both a minimum and maximum boundary. This might seem weird at first, as the user would expect to always want minimum response time, but, in this case, this might be as result of under using the virtual machines, which is not cost effective. Similarly, a lower boundary was determined for the `throughout` dimension (lower left corner), and the points of low throughput are consistent with those of low `responseTime`, thus strengthening the previous conclusion. By investigating the elasticity pathway of the event processing service topology (Figure 10), by summing up the behaviour situations, the user can determine that in approximately 20% to 30% of the situations, there is high response time with low number of clients, which might indicate a potential bottleneck somewhere in the service.

Figure 8 Event processing service topology elasticity space snapshot (see online version for colours)



Thus, the MELA user focuses next on the service units belonging to the *DataEnd* topology, and examines the elasticity space of the *DataController* service unit by capturing its `writeLatency` and `cpuUsage` (Figure 11). For the `writeLatency` elasticity space dimension (left side), MELA determined a higher elasticity space boundary, indicating the maximum latency recorded in which the

service respected the cost requirement. For the `cpuUsage` dimension there is only a lower elasticity boundary. Investigating the elasticity pathway of the *DataController* service unit (Figure 12), the MELA user can determine that in approximately 30% of the encountered situations the `cpuUsage` was over 95%, and in around 50% over 90%, indicating a potential bottleneck. However, a further analysis is needed to understand the complete elasticity behaviour of the DaaS service, by investigating the elasticity space and pathway for the other service elements, selecting more metrics to be analysed, and/or refining the elasticity requirements for the COMOT elasticity controller.

Figure 9 Event processing service topology elasticity space (see online version for colours)

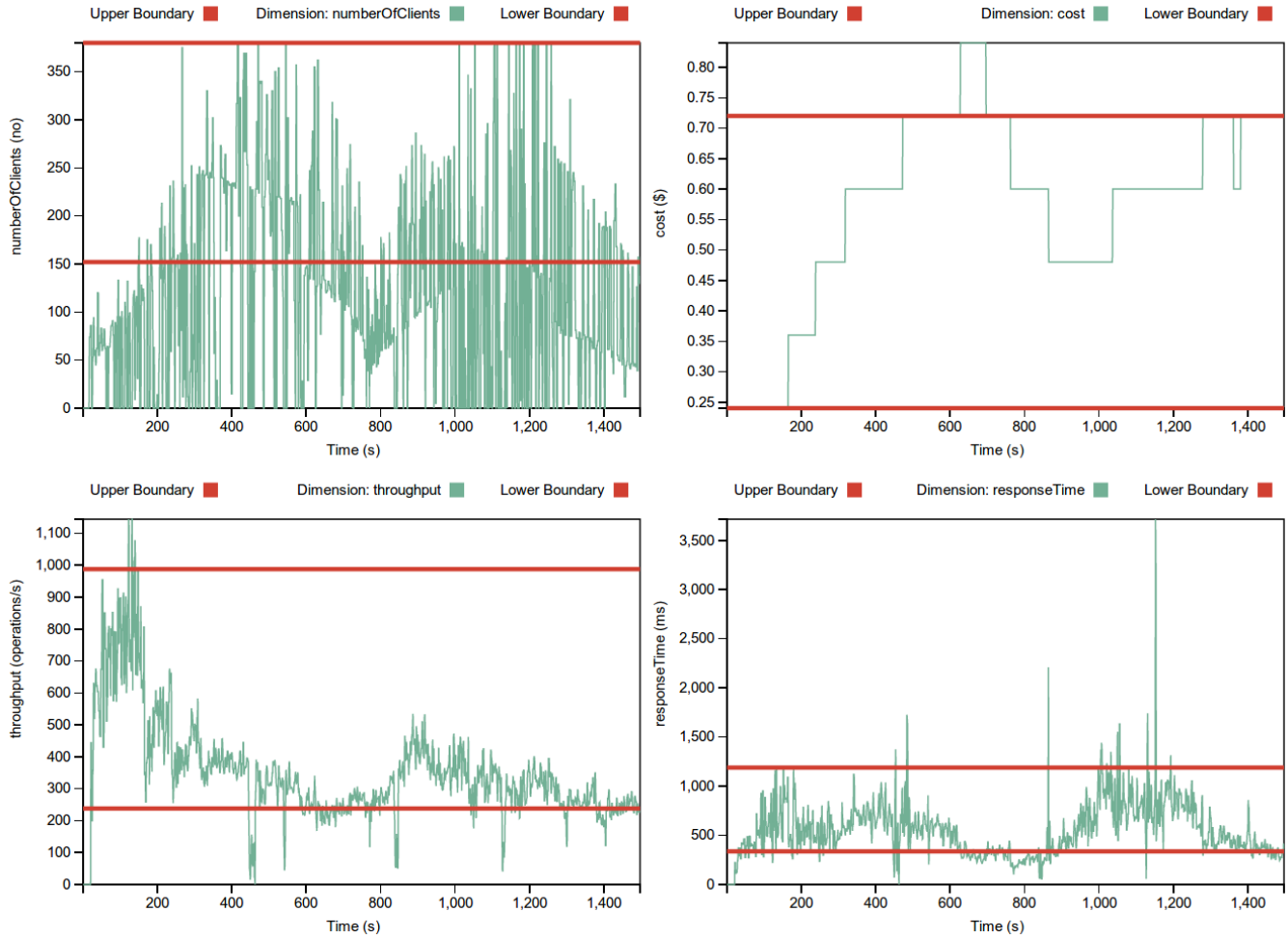


Figure 10 Event processing service topology elasticity pathway (see online version for colours)

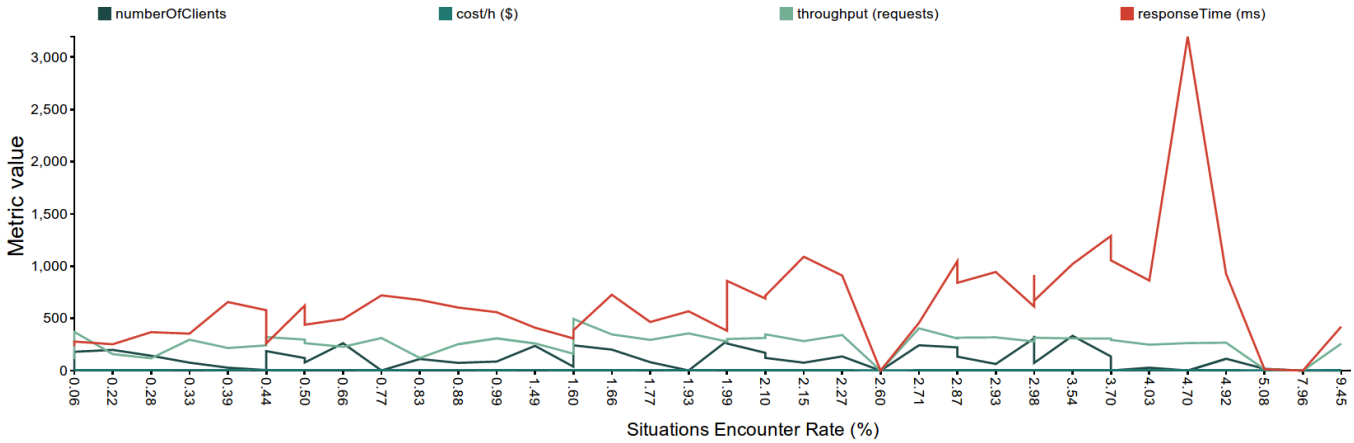


Figure 11 Data controller service unit elasticity space (see online version for colours)

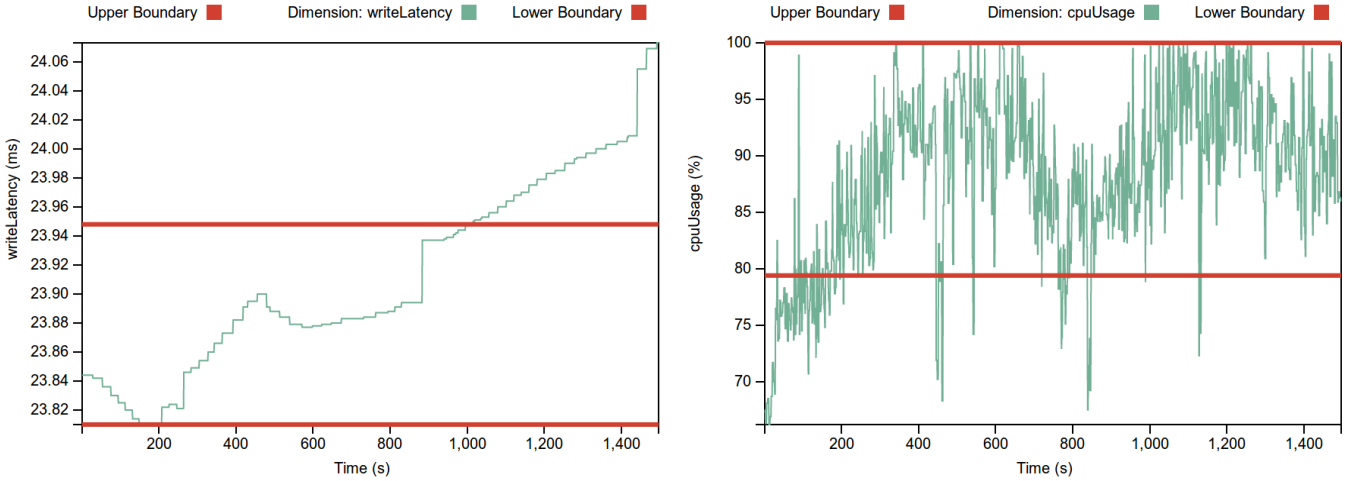
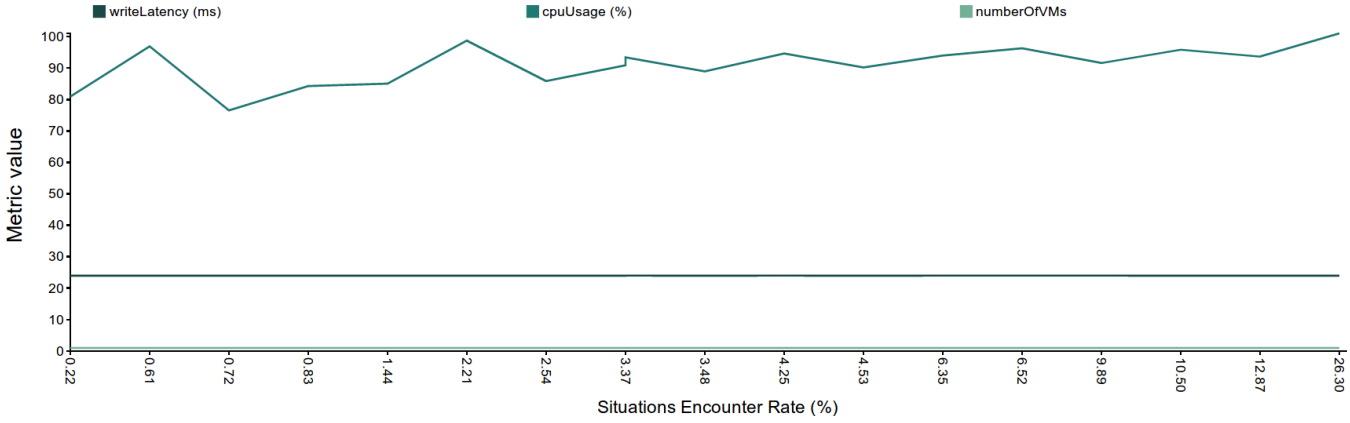


Figure 12 Data controller service unit elasticity pathway (see online version for colours)



The above-mentioned experiments have highlighted the importance and challenges of monitoring and analysing the behaviour of elastic services at multiple levels. The elasticity space is crucial in understanding the elasticity boundaries of cloud services. Given a control mechanism and workload, in these experiments, from the elasticity space analysis and a single high level user requirement, the user can obtain insights regarding what other boundaries the control mechanism must enforce at different levels in the cloud service, e.g., that it needs at least 148 clients. Applying the elasticity pathway function, the user can learn what is the overall behaviour of the cloud service, bringing insight in how does the service behave and what are the dependencies between the service's metrics, e.g., `responseTime`, `throughput` and `numberOfUsers`. With this insight, the user can refine the service's requirements and resume the analysis process, iteratively improving the service's elasticity.

7 Related work

A framework for data collection and aggregation is introduced by Wang et al. (2011). In Trihinas et al. (2014), the authors introduce JCatascopia, a tool for monitoring elastic applications relying on dynamic probe addition/removal. Leitner et al. (2012) aim to extract application-specific performance information from system-level metrics, relying on application event streams. Dhingra et al. (2012) monitor cloud resource usage from the infrastructure owner perspective. An adaptive cloud monitoring tool providing estimations on monitoring accuracy is described by Meng et al. (2012). An elastic monitoring framework for cloud infrastructures is presented by Konig et al. (2012), providing a powerful query mechanism for retrieving service level information, and Katsaros et al. (2012) present a monitoring system collecting both virtual infrastructure and service level information. He et al. (2013) propose a cloud services monitoring framework analysing monitoring information and detecting abnormal behaviour, while Venzano and Michiardi (2013) study traffic patterns on a private cloud, highlighting that relationships between metrics are influences by network, virtualisation layer, and VM collocation. Gullhav et al. (2013) apply an extended response time block method to monitor and approximate the response time of cloud services, considering the horizontal scalability of a single business tier, while Lloyd et al. (2012) correlate physical and virtual machine resource utilisation statistics to predict application performance across VMs.

Xiong et al. (2013) introduce vPerfGuard, a framework for service performance diagnosis in consolidated cloud environments, automatically discovering metrics which are most descriptive of service performance. In contrast, we adopt the cloud user perspective, and provide a customisable mechanism for mapping system level monitoring data to the running service structure, and enriching data obtained from various monitoring systems, providing a complete view over the monitored cloud service behaviour.

Cloud monitoring is also the focus of many industry tools such as Nagios (http://www.nagios.org/), Ganglia (http://ganglia.sourceforge.net/), Zabbix (http://www.zabbix.com/), OpenNMS (http://www.opennms.org/), or Hyperic (http://www.hyperic.com/). Such tools mostly focus on gathering data from the physical and virtual infrastructure, and distributing it, without correlating it with the services running on it. Such tools can act as data sources for our framework, as we do not focus data collection. We differ as we do not focus on monitoring, and rely on data from existing solutions, structure and enrich it, and use it to analyse the elasticity of cloud services.

Based on monitoring information, service analysis and control mechanisms can be built. An architecture for controlling cloud services using high-level rules is introduced by Vaquero et al. (2012). The cost of Amazon EC2 spot instances is analysed by Agmon Ben-Yehuda et al. (2011). Emeakaroha et al. (2012) present CASViD, an adaptive architecture for evaluating SLA violations using both system-level and application-level monitoring information. A mechanism for adapting cloud allocation is presented by Singh et al. (2010), using an aggregation that monitors the workload at each service tier. Demchenko et al. (2012) present a framework for multi-domain heterogeneous cloud services interoperability. Konstantinou et al. (2012) present a cloud-enabled framework for monitoring and adaptively resizing NoSQL clusters. Kolodner et al. (2011) introduce a scalable cloud environment for data-intensive storage services, scaling with respect to the resource usage cost and service performance. Warneke and Kao (2011) target elastic data processing in clouds by allocating virtual machines on demand, and Duong et al. (2011) introduce a framework for dynamic resource provisioning and adaptation in IaaS clouds. Sampaio and Mendonça (2011) propose Uni4Cloud, a framework for the deployment and management of multi-cloud services, while Miglierina et al. (2013) introduce a control theoretic approach for multi-cloud services, targeting resource level control. We differ as we provide insight in the elasticity of cloud services, and introduce concepts and techniques for extracting elasticity boundaries, with respect to the cost, quality and performance of cloud services, based on which control strategies for the service's elasticity can be refined.

8 Conclusions and future work

Elasticity analytics are crucial to cloud service developers and providers to understand the behaviour of their services at multiple levels, from individual units to the whole service, towards developing smarter mechanisms for controlling their elasticity. This paper introduced concepts and techniques for monitoring and analysing the elasticity of cloud services. A cross-level metric composition mechanism was introduced, for linking service level with system level monitoring information, and deriving higher level information from it. For characterising the behaviour of elastic cloud services, the concepts of elasticity space and elasticity pathway were introduced. Evaluating the elasticity space based on the cross-level metric composition mechanism and user requirements, our approach determines elasticity boundaries for all service's elements, bringing insight in the behaviour of the service, and how such a service should be controlled. Applying the elasticity pathway over the elasticity space, a mechanism for classifying the elasticity behaviour of cloud services was defined, providing insight in the service's behaviour evolution and a base for predicting it. We have introduced MELA, elasticity analytics as a service, implementing the concepts and techniques defined in this paper. MELA provides features for real-time multi-level analysis of elastic cloud services, and integrates different elasticity analysis functions to support the analysis of other complex elastic behaviour.

As future work we will investigate cloud infrastructure elasticity and elasticity of cloud offered services. We will further focus on determining patterns of elasticity behaviour of cloud services. Furthermore, we will work on automatic extraction of elasticity dependencies between elasticity space dimensions, providing deeper insight in cloud service's elasticity.

Acknowledgements

This work was partially supported by the European Commission in terms of the CELAR FP7 project (FP7-ICT-2011-8 #317790). This article is a revised and expanded version of the paper entitled 'MELA: monitoring and analysing elasticity of cloud services', presented at IEEE International Conference on Cloud Computing Technology and Science (CloudCom), Bristol, UK, December 2–5, 2013.

References

Agmon Ben-Yehuda, O., Ben-Yehuda, M., Schuster, A. and Tsafrir, D. (2011) 'Deconstructing Amazon EC2 spot instance pricing', in *International Conference on Cloud Computing Technology and Science, CloudCom*.

Copil, G., Moldovan, D., Truong, H-L. and Dustdar, S. (2013) 'Multi-level elasticity control of cloud services', in Basu, S., Pautasso, C., Zhang, L. and Fu, X. (Eds.): *Service-Oriented Computing, Lecture Notes in Computer Science*, Vol. 8274, pp.429–436.

Dean, D.J., Nguyen, H. and Gu, X. (2012) 'UBL: unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems', in *International Conference on Autonomic Computing, ICAC*, ACM, pp.191–200.

Demchenko, Y., Makkes, M., Strijkers, R. and de Laat, C. (2012) 'Intercloud architecture for interoperability and integration', in *International Conference on Cloud Computing Technology and Science, CloudCom*, IEEE, pp.666–674.

Dhingra, M., Lakshmi, J. and Nandy, S.K. (2012) 'Resource usage monitoring in clouds', in *International Conference on Grid Computing, GRID*, pp.184–191.

Duong, T.N.B., Li, X. and Goh, R. (2011) 'A framework for dynamic resource provisioning and adaptation in IaaS clouds', in *International Conference on Cloud Computing Technology and Science, CloudCom*, IEEE, pp.312–319.

Dustdar, S., Guo, Y., Satzger, B. and Truong, H.L. (2011) 'Principles of elastic processes', *IEEE Internet Computing*, September–October, Vol. 15, No. 5, pp.66–71, doi: 10.1109/MIC.2011.121.

Emeakaroha, V., Ferreto, T., Netto, M., Brandic, I. and De Rose, C. (2012) 'Casvid: application level monitoring for SLA violation detection in clouds', in *Computer Software and Applications Conference, COMPSAC*, IEEE, pp.499–508.

Gullhav, A., Nygreen, B. and Heegaard, P. (2013) 'Approximating the response time distribution of fault-tolerant multi-tier cloud services', in *IEEE/ACM International Conference on Utility and Cloud Computing (UCC)*, pp.287–291.

Han, R., Guo, L., Ghanem, M. and Guo, Y. (2012) 'Lightweight resource scaling for cloud applications', in *IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp.644–651.

He, S., Ghanem, M., Guo, L. and Guo, Y. (2013) 'Cloud resource monitoring for intrusion detection', in *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, Vol. 2, pp.281–284.

Katsaros, G., Kousiouris, G., Gogouvitis, S.V., Kyriazis, D., Menychtas, A. and Varvarigou, T. (2012) 'A self-adaptive hierarchical monitoring mechanism for clouds', *Journal of Systems and Software*, Vol. 85, No. 5, pp.1029–1041.

Kolodner, E., Tal, S., Kyriazis, D., Naor, D., Allalouf, M., Bonelli, L., Brand, P., Eckert, A., Elmroth, E., Gogouvitis, S., Harnik, D., Hernandez, F., Jaeger, M., Lakew, E., Lopez, J., Lorenz, M., Messina, A., Shulman-Peleg, A., Talyansky, R., Voulodimos, A. and Wolfsthal, Y. (2011) 'A cloud environment for data-intensive storage services', in *International Conference on Cloud Computing Technology and Science, CloudCom*, IEEE, pp.357–366.

Konig, B., Alcaraz Calero, J. and Kirschnick, J. (2012) 'Elastic monitoring framework for cloud infrastructures', *IET Communications*, Vol. 6, No. 10, pp.1306–1315.

Konstantinou, I., Angelou, E., Tsoumakos, D., Boumpouka, C., Koziris, N. and Sioutas, S. (2012) 'Tiramola: elastic NoSQL provisioning through a cloud management platform', in *International Conference on Management of Data, SIGMOD*, ACM, pp.725–728.

Leitner, P., Inzinger, C., Hummer, W., Satzger, B. and Dustdar, S. (2012) 'Application-level performance monitoring of cloud services based on the complex event processing paradigm', in *Service-Oriented Computing and Applications, SOCA*, pp.1–8.

Lloyd, W., Pallickara, S., David, O., Lyon, J., Arabi, M. and Rojas, K. (2012) 'Performance modeling to support multi-tier application deployment to infrastructure-as-a-service clouds', in *IEEE International Conference on Utility and Cloud Computing (UCC)*, pp.73–80.

Mao, M. and Humphrey, M. (2013) 'Scaling and scheduling to maximize application performance within budget constraints in cloud workflows', in *IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, pp.67–78.

Meng, S., Iyengar, A.K., Rouvellou, I., Liu, L., Lee, K., Palanisamy, B. and Tang, Y. (2012) 'Reliable state monitoring in cloud datacenters', in *International Conference on Cloud Computing Technology and Science, CLOUD*, IEEE, pp.951–958.

Miglierina, M., Gibilisco, G., Ardagna, D. and Di Nitto, E. (2013) 'Model based control for multicloud applications', in *International Workshop on Modeling in Software Engineering (MiSE)*, pp.37–43.

Moldovan, D., Copil, G., Truong, H-L. and Dustdar, S. (2013) 'MELA: monitoring and analyzing elasticity of cloud services', in *International Conference on Cloud Computing Technology and Science, CloudCom*, to appear.

Sampaio, A. and Mendonça, N. (2011) 'Uni4cloud: an approach based on open standards for deployment and management of multi-cloud applications', in *International Workshop on Software Engineering for Cloud Computing (SEECLOUD)*, ACM, pp.15–21.

Singh, R., Sharma, U., Cecchet, E. and Shenoy, P. (2010) 'Autonomic mix-aware provisioning for non-stationary data center workloads', in *International Conference on Autonomic Computing, ICAC*, pp.21–30.

Trihinas, D., Pallis, G. and Dikaiakos, M.D. (2014) 'JCatascopia: monitoring elastically adaptive applications in the cloud', in *International Symposium on Cluster, Cloud and Grid Computing, CCGRID*.

Truong, H-L., Dustdar, S., Copil, G., Gambi, A., Hummer, W., Le, D-H. and Moldovan, D. (2014) 'CoMoT – a platform-as-a-service for elasticity in the cloud', in *International Workshop on the Future of PaaS*.

Vaquero, L.M., Morán, D., Galán, F. and Alcaraz-Calero, J.M. (2012) 'Towards runtime reconfiguration of application control policies in the cloud', *Journal of Network and Systems Management*, Vol. 20, No. 4, pp.489–512.

Venzano, D. and Michiardi, P. (2013) 'A measurement study of data-intensive network traffic patterns in a private cloud', in *DCC 2013, Workshop on Distributed Cloud Computing, IEEE/ACM Conference on Utility and Cloud Computing (UCC)*, Dresden, Germany'.

Wang, C., Schwan, K., Talwar, V., Eisenhauer, G., Hu, L. and Wolf, M. (2011) 'A flexible architecture integrating monitoring and analytics for managing large-scale data centers', in *International Conference on Autonomic Computing, ICAC*, pp.141–150.

Warneke, D. and Kao, O. (2011) 'Exploiting dynamic resource allocation for efficient parallel data processing in the cloud', *IEEE Transactions on Parallel and Distributed Systems*, Vol. 22, No. 6, pp.985–997.

Xiong, P., Pu, C., Zhu, X. and Griffith, R. (2013) 'vPerfGuard: an automated model-driven framework for application performance diagnosis in consolidated cloud environments', in *ACM/SPEC International Conference on Performance Engineering (ICPE)*, pp.271–282.