MELA: Monitoring and Analyzing Elasticity of Cloud Services

Daniel Moldovan, Georgiana Copil, Hong-Linh Truong, Schahram Dustdar
Distributed Systems Group, Vienna University of Technology
E-mail: {d.moldovan, e.copil, truong, dustdar}@dsg.tuwien.ac.at

***Abstract*—Cloud computing has enabled a wide array of applications to be exposed as elastic cloud services. While the number of such services has rapidly increased, there is a lack of techniques for supporting cross-layered multi-level monitoring and analysis of elastic service behavior. In this paper we introduce novel concepts, namely *elasticity space* and *elasticity pathway*, for understanding elasticity of cloud services, and techniques for monitoring and evaluating them. We present MELA, a customizable framework that enables service providers and developers to analyze cross-layered, multi-level elasticity of cloud services, from the whole cloud service to service units, based on service structure dependencies. Besides support for real-time elasticity analysis of cloud service behavior, MELA provides several customizable features for extracting functions and patterns that characterize that behavior. To illustrate the usefulness of MELA, we conduct several experiments with a realistic data-as-a-service in an M2M cloud platform.**

***Keywords*-elastic computing, cloud service, elasticity monitoring, elasticity analysis**

I. Introduction

With the increasing popularity of cloud systems, the number of applications and systems born in or migrated to cloud environments has substantially increased. Diverse types of stakeholders in cloud environments, ranging from business oriented software providers to e-science professionals, have continuously investigated techniques exposing diverse functionalities as a service (XaaS). In this context, substantial effort have been paid for the development of emerging *elastic cloud services*, which scale up/out as long as 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, in general, elastic cloud services have three main dimensions: *"resource elasticity"*, *"cost elasticity"* and *"quality elasticity"* [1].

Developing and managing elastic cloud services with such multi-dimensional elasticity is challenging. One of the main questions is how to monitor and evaluate cloud service's elastic behavior, determine the proper cost and quality indicators and their boundaries, and utilize them for optimizing and controlling the services' *elasticity*. Currently, the process of deciding cost and quality indicators and their boundaries is somewhat of a black art, usually done by cloud experts. Existing monitoring and analysis tools focus either on cloud service level [2] or on the underlying virtual infrastructure [3][4], and do not provide a cross-layered, multi-level service elasticity behavior picture, hindering the discovery of the cause for stakeholder goal's violations. For controlling elastic cloud services one must be able to detect if a goal violation originates in a poorly chosen cloud service, subscription scheme, resource congestion, or failing service unit. In particular, we think that in order to understand elastic cloud services, we need to investigate new concepts that can be used to characterize and extract cloud service's elastic behavior from multi-dimensional monitoring data.

In this paper, we present MELA, a framework for monitoring and analyzing elasticity of cloud services. MELA allows cloud service developers and providers to trace their service behavior from the whole service level to the underlying virtual infrastructure, extracting characteristics and providing crucial insight in their elastic behavior. Providing aggregated monitoring data and analysis of elastic cloud services, MELA acts as a base for analyzing and controlling elastic cloud services. The main contributions of our paper are:

- a novel concept of *elasticity space* and *elasticity pathway* for analyzing elastic behavior of cloud services at multiple levels.
- customizable mechanisms for extracting runtime boundaries of cloud service's elasticity that fulfill user-defined elasticity requirements.
- techniques for evaluating multi-level user-defined elasticity requirements over service cost, quality and resources, enabling custom analysis of cloud services' elastic behavior.

The rest of this paper is structured as follows. Section II presents the motivation and research problems. Section III presents our techniques for monitoring and analyzing elasticity of cloud services using the elasticity space and pathway. In Section IV we describe the MELA framework. Section V presents our prototype and experiments. We discuss related work in Section VI. Section VII concludes the paper and outlines the future work.

II. Motivation & Research problems

Let us consider a realistic data-as-a-service (DaaS) application for an M2M cloud platform-as-a-service, for which we have user-defined elasticity requirements[1] w.r.t service run-

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

[1]In this paper "users" refer to the users of our framework: cloud service provider and developer and elasticity controllers

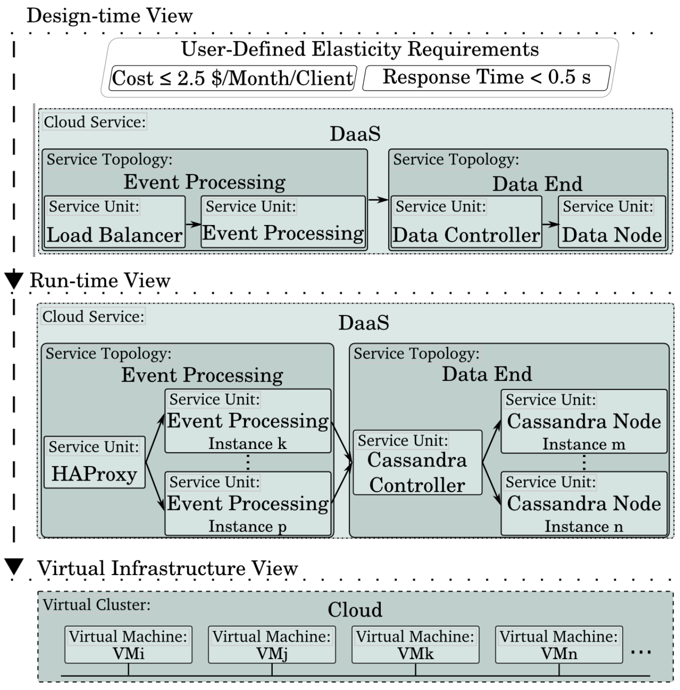


Fig. 1: Elastic Cloud Service Views

time performance and cost.

In general, we have three main views from which cloud services are described (Fig. 1): (i) design-time view, where we see the whole service dependency model (Cloud Service, Service Topology, Service Unit) and user-defined service requirements, (ii) run-time view, where instances of several Service Units are deployed and executed in virtual machines, and (iii) the virtual infrastructure, where several virtual machines, possibly grouped in virtual clusters, are used. From the design-time point of view, the DaaS has two service topologies, `Data End` and `Event Processing`, supporting horizontal scaling by addition and removal of VMs. The `Data End` service topology includes two service units, a `Data Node` holding data, and a `Data Controller` managing it. The `Event Processing` service topology also contains two service units: `Load Balancer`, distributing client requests, and `Event Processing` interacting with the `Data End`. At run-time, the `Data End` units uses Cassandra[2] for its service units, and the `Event Processing` uses HAProxy[3] for its `Load Balancer` service unit. Moreover, at run-time, due to user-defined elasticity requirements, service unit instances are added/removed dynamically, triggering allocation and deallocation of virtual machines at the virtual infrastructure level.

User-defined service requirements can be viewed as elasticity requirements, as they restrict the elastic behavior of the DaaS. To enforce such restrictions at the virtual infrastructure level, i.e. when to add/remove a VM, of what type, under which pricing scheme, the restrictions need to be linked and mapped to the run-time view. Monitoring data from the run-time view must also be linked back to the user-defined elasticity requirements. This enables the discovery of service units belonging to service topologies that cause requirements violations. For example, using MELA, one should be able to determine that a high service cost is caused by to few clients served at the `Event Processing` service topology level because of overloaded `Event Processing` service unit instances, or high communication latency between the `Event Processing` instances and the `Data Controller`.

While using various monitoring techniques such as [4] or [5] we can capture monitoring data from the whole service level or virtual infrastructure level, such data typically does not answer the following crucial questions:

- what should be the behavior of the service topologies and units when user-defined elasticity requirements are fulfilled
- when is the service behavior elastic, i.e. adapting and fulfilling user-defined elasticity requirements
- what is the cause of an elasticity requirement violation, traced from the design-time view to the underlying virtual infrastructure
- how does the service elastic behavior evolve in time, i.e. what are the correlations and patterns in the service behavior

Capturing, describing and analyzing elastic behavior of cloud service is crucial not only for developers who build and optimize cloud services, but also for software controller that change the topology of such services at run-time, enforcing user-defined elasticity requirements. Such controllers need a monitoring and analysis mechanism that extracts elasticity characteristics, which can be used to refine user-defined elasticity requirements or predict the service behavior, leading to better service control and quality.

This motivates us to investigate the following issues:

- Which concepts can be used to capture the elastic behavior of cloud services?
- How to extract characteristics that describe the service elastic behavior to support both reactive and predictive control of elastic services?
- How to analyze the cloud service behavior, 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 analyzing their behavior from multiple views, and characterizing the elastic behavior of each cloud service based on user-defined elasticity requirements.

III. Monitoring and Analyzing Elastic Cloud Services

A. *Runtime Properties of Elasticity of Cloud Services*

In order to support multi-dimensional analysis of elastic services behavior [1], we categorize monitoring data in three dimensions: $Cost$, $Quality$ and $Resource$. Each dimension has a set of metrics, shown in Figure 2. These categories are sufficient for capturing low-level data about any monitored

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

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

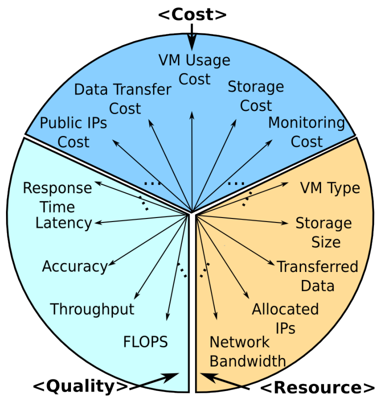


Fig. 2: Elasticity Dimensions

element (e.q., service topology or service unit) within a cloud service that can be used for understanding the elastic behavior of that service. Conceptually, to capture monitoring data associated with a monitored element at a specific time t, we define the *monitoring snapshot*, ms:

$$ms = (\langle c_i \rangle, \langle q_j \rangle, \langle r_k \rangle, t) \tag{1}$$

where $c_i \in Cost$, $q_j \in Quality$, and $r_k \in Resource$.

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 analyze the elastic behavior of a monitored element, we represent metric boundaries extracted from user-defined elasticity requirements as well as boundaries detected/evaluated from our monitoring techniques using the *elasticity boundary*:

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:

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

where c_i^u and c_i^l denote the upper bound and the lower bound of metric c_i, respectively, q_j^u and q_j^l for q_j, and r_k^u and r_k^l for r_i. All metrics c_i, g_j, r_k are specified in elasticity requirements indicating the parameters under which the cloud service is elastic. In the rest of this paper, *user-defined elasticity boundary* and *evaluated elasticity boundary* are used to indicate the boundary extracted from user-defined requirements and determined from elasticity monitoring, respectively.

B. Elasticity Space and Pathway

Given a set of monitoring snapshots and a user-defined elasticity boundary, we need to understand when a monitored element is in elastic behavior, if its behavior violates the elasticity boundary and if we can characterize the service behavior using some specific "pathways". Naturally, we expect that the meaning of "elasticity" will be dependent on the types of monitored elements, their runtime settings and requirements. To this end, we define the concept of *elasticity space* to determine if a monitored element is in elastic behavior:

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

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: detect when an elastic behavior starts and stops, and extract only monitoring data describing the service 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, such as specific types of service units, service topologies, or the whole cloud service. Furthermore, these functions are also dependent on the metrics in user-defined elasticity boundaries.

An elasticity space function is designed to extract useful information about the overall behavior of the cloud service when elasticity requirements are fulfilled. A space would contain, for example, only the $throughput$ and $cost/VM/h$ metrics from which the $serviceCost/client/h$ targeted by requirements can be extracted, not including metrics that have no impact on it. One can analyze the behavior of an elastic cloud service by checking if its elasticity space is within the user-defined elasticity boundaries.

While the elasticity space enables cloud service elasticity analysis, it does not provide insight into relationships between metrics influencing the elastic behavior over time, e.g. $throughput$ and $cost/VM/h$ might or might not follow a linear relationship. In order to characterize the elastic behavior 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 characterizes the elasticity relationship among* m_i *over the time.*

Formally, an elasticity pathway $elPtw$ is extracted 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 behavioral patterns or characteristics of the 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.

An elasticity pathway function is designed to perform a complex evaluation of the cloud service behavior, extracting characteristics that can be used to predict future behavior. One function could extract metric correlations, e.g. between $throughput$ and $cost/VM/h$ for a service unit, determining the influence of throughput on the cost. Another function could classify the cloud service behavior in usual and unusual combinations of metric values. The elasticity pathway function is applied to extract behavioral characteristics of monitored elements from the elasticity space, the quality of the extracted elasticity pathway being heavily influenced by the size and

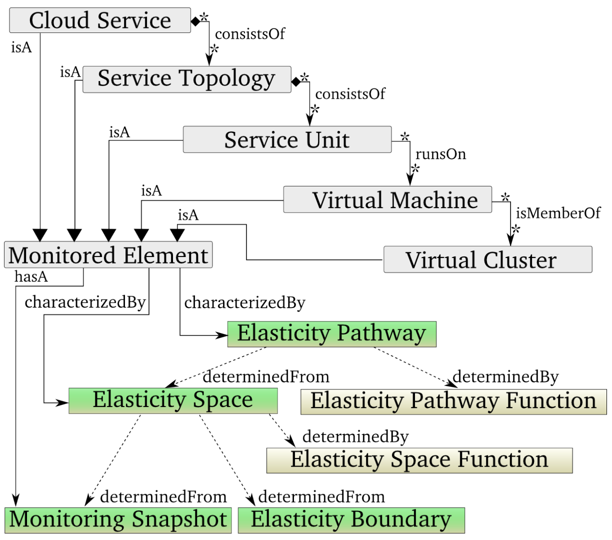


Fig. 3: Association between Elasticity Space/Pathway and Monitored Elements

data in the elasticity space.

C. Multi-Level Elasticity Space of Cloud Services

In order to monitor and evaluate the elasticity space and pathways of a cloud service, we need to (i) build the cloud service dependency model, (ii) build the monitoring snapshot for all monitored elements, and (iii) apply particular elasticity space and pathway functions over the monitoring snapshots.

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 decomposition of user-defined elasticity requirements in lower level requirements which can be mapped to the underlying virtual infrastructure. To support analysis of service behavior at different levels, we represent a cloud service as composed of service topologies, each topology containing several service units, which are deployed on one or more virtual clusters running virtual machines (Figure 3). Based on the dependency model, we build composite monitoring snapshots which aggregate metrics bottom-up, creating based on lower level metrics higher level ones which can be targeted by user-defined elasticity requirements. At different levels we allow the association with different elasticity space and pathway functions, enabling custom analysis of cloud service behavior at multiple levels. Using the dependency model we combine monitoring snapshots, elasticity spaces and pathways from one level and propagate them to the other levels, obtaining a complete view over the cloud service behavior.

To build the elasticity space, the dependency model of the cloud service is extracted or defined by users/tools, and used for analyzing the service behavior. For easy integration with other tools, we use an XML-based representation. Listing 1 shows the M2M DaaS containing two service topologies and their service units.

Listing 1: Example of M2M DaaS Dependency Model

```
<CloudService id="S" name="M2MDaaS">
    <ServiceTopology id="ST1" name="EventProcessing">
        <ServiceUnit id="ST1_SU1" name="LoadBalancer"/>
        <ServiceUnit id="ST1_SU2" name="EventProcessing"/>
    </ServiceTopology>
    <ServiceTopology id="ST2" name="DataEnd">
        <ServiceUnit id="ST2_SU1" name="DataController"/>
        <ServiceUnit id="ST2_SU2" name="DataNode"/>
    </ServiceTopology>
</CloudService>
```

2) Cross-layered Metric Composition: Monitoring snapshots capture (low-level) metrics, but not necessary the same as elasticity requirements defined by the user. For example, a monitoring snapshot for the Event Processing service unit might only include $responseTime$, $throughput$ and $cost/VM/h$ while the elasticity requirements might target $serviceCost/client/h$ and $numberOfClients$. Therefore, to obtain a complete view over the cloud service behavior, from low level metrics to higher ones, we develop techniques for cross-layer composition of monitoring snapshots, elasticity spaces and elasticity pathways, following the cloud service structure. This composition introduces the problem of combining/aggregating metrics. Depending on the types of metrics, a valid composition of two metrics might involve different operations. We define an XML-based domain-specific language for describing such metric composition rules as a cascading sequence of operations which apply one or more operators over one or more operands. Defining the composition rules requires domain specific knowledge, e.g., in determining that the cost of a service unit is computed from the hourly cost per VM instance and the network data transfer.

Based on the metric composition, we provide an analysis mechanism for extracting the elasticity space and pathway at different dependency model levels, providing a multi-level decomposition of the cloud service behavior. Such a decomposition is beneficial to service 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 behavior of cloud services enables service provider/developer and software controllers to reason on the same service using separate perspectives.

In Listing 2 we show a composition rule obtaining the cost of data transfered per client per hour for the `Data`

Listing 2: Example of Metric Composition

```
<CompositionRules TargetLevel="ServiceTopology" >
   <CompositionRule TargetID="ST1">
      <SourceMetric name="dataOut" unit="GB/s"
             level="ServiceUnit"/>
      <Operations>
         <Operation name="SUM"/>
         <Operation name="DIV" unit="no/s"
                   sourceMetric="numberOfClients"
                   level="ServiceTopology"/>
      </Operations>
      <ResultingMetric name="dataCostPerClient" unit="$"
         type="COST"/>
   </CompositionRule>
</CompositionRules>
```

Algorithm 1 Multi-Level Elasticity Space/Pathway Analysis

```
FUNCTION analyze(element)
  elPathways, subSnapshots := []
  elSpace, monSnapshot:= NULL

  FOR EACH child IN element.children DO
    subSnapshots.push(analyze(child))
  END FOR

  FOR EACH rule IN element.compositionRules DO
    values := []
    FOR EACH operand IN rule.operands DO
      IF operand.type IS "metric" THEN
        values.push(
          getValue(operand, subSnapshots, monSnapshot))
      ELSE
        values.push(operand)
      END IF
    END FOR

    FOR EACH operator IN rule.operators DO
        values := operator(values)
    END FOR
    monSnapshot.push(values)
  END FOR

  elSpace := element.elSpaceFunction(monSnapshot)

  FOR EACH elPtwFunc IN element.elPtwFunctions DO
    elPathways.push(elPtwFunc(elSpace))
  END FOR
```

`End` service topology. The metric is obtained by summing up the values of the $dataOut$ metric from all service topology children having level service unit, and dividing the result with the value of the $numberOfClients$ metric retrieved from the target service topology.

3) Multi-level Elasticity Space/Pathway Analysis: Following the service dependency model, Algorithm 1 traverses the model in a depth-first manner and carries out runtime analysis. For every element at each level, its metric composition rules are applied, building a multi-level monitoring snapshot. Each composition rule contains a set of operands (metrics or values) and a set of operations from $\{sum, max, min, avg, div, add, sub, mul, concat, union, set, keep, keep\_last, keep\_first\}$. Over the space we apply the elasticity pathway functions, analyzing the cloud service behavior.

IV. Elasticity Monitoring as a Service

A. MELA Overview

Based on our concepts in Section III, we develop MELA , elasticity space monitoring and analysis as a service (Fig. 4). MELA contains a core *MELA Service*, and a lightweight *Data Collector* node. The *Data Collector* node is a customizable 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 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, which implies a configuration step using the *Elasticity Functions*

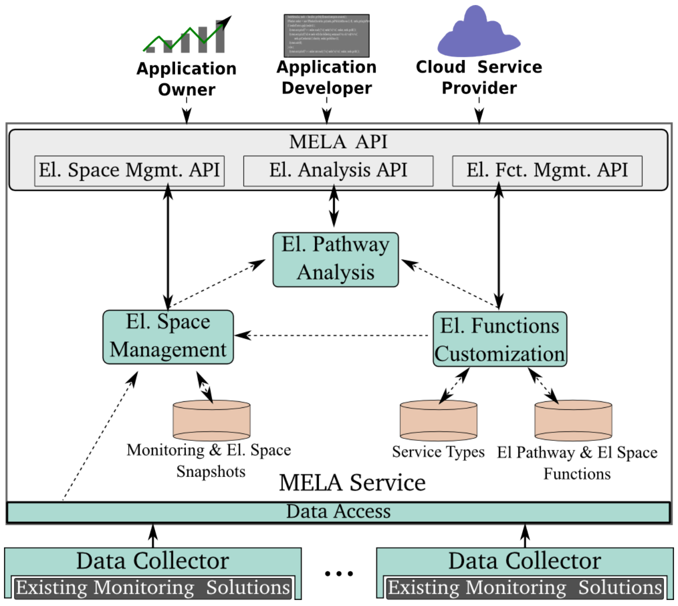


Fig. 4: MELA Overview

Management API, defining for the monitored elements at each level the composition operations to be applied. This step feeds data into the *El. Pathway & El. Space Functions* repository which also contains the composition rules for building the monitoring snapshot. As the elasticity space is determined from monitoring snapshots, the *El. Space Management* unit also handles the monitoring snapshot construction and stores them in the *Monitoring & El. Space Snapshots* database.

MELA exposes its functionality through REST services and Java API, providing methods for configuring MELA and analyzing elastic service behavior.

B. Elasticity Space and Pathway Function Prototypes

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

Based on the elasticity space, custom *elasticity pathway* functions can be defined as MELA plug-ins, enabling custom analysis of service behavior. For the prototype *elasticity pathway* function, we adapt an unsupervised behavior learning technique using self organizing maps (SOMs) [6], and classify monitoring snapshots by encountering rate in DOMINANT, NON-DOMINANT, and RARE. Such a pathway is important for understanding if the regular behavior of the service respects user-defined elasticity requirements.

As SOMs are unsupervised neural networks that map multi-dimensional spaces into low dimensional ones, we use them for grouping monitoring snapshots. Each SOM's neuron value is derived from its snapshots. Each monitoring snapshot is mapped to the group from which it has the smallest distance.

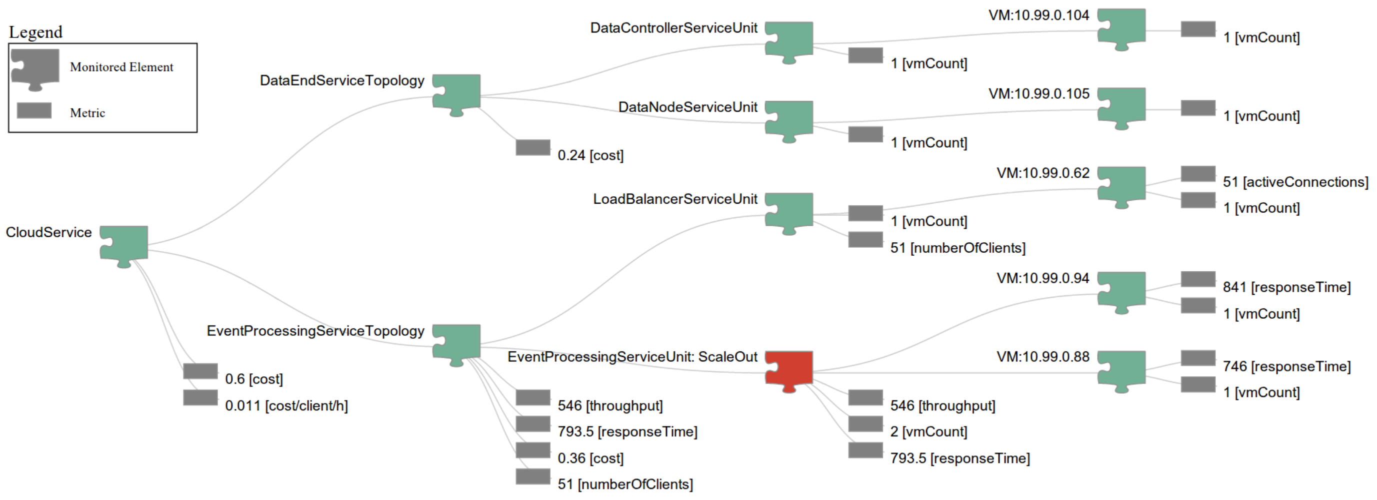


Fig. 5: MELA Visualization of Multi-Level Monitoring Data

With each new snapshot, the group and its SOM neighbors are updated using the function $V_{new}(group) = V_{old}(group) + A * N(group)(V(snapshot) - V_{old}(group))$, where A is a discount factor, and $N(group)$ is a neighborhood function determining the degree with which a group value is updated. We initialize the SOM with snapshot groups having all metrics equal to 0, and rely on its self-adaptive nature to map the input data. We use a neighborhood function of 1 for the directly targeted group and of $1/neighbourLevel/neighboursCount$ for the group neighbors. Updating the neighbors creates and update new groups, mapping the input data better. The discount learning factor is $1/neighbourLevel$, the neighborhood is 2 and the map size is 10×10 . A filtering step merges groups with same value, consolidating the monitored snapshots.

V. Experiments

We apply MELA[4] to monitor and analyze the realistic data-as-a-service application for an M2M cloud mentioned in Section II. From real sensor data, we simulated a Gaussian distribution of M2M sensors – the clients connected to our system, starting from 50 sensors per hour, increasing up to 350, and then decreasing again. Each request from a sensor will require between 1 and 10 operations on the data service. For practical reasons we simulated a hour of experiment each second. The service VMs were deployed on our OpenStack[5] cloud. For these experiments MELA utilizes Ganglia[6] as base for the Data Collector nodes, retrieving generic OS level and service specific monitoring data (e.g., numberOfClients, throughput, response time) from custom Ganglia plug-ins.

A. Monitoring Elasticity Space

We start from the scenario in which the cloud service provider wants to monitor and define elasticity requirements for the service cost and performance. Using our cross-layered metric composition mechanism, we derive service level metrics from low level ones. We start at the Service Topology level defining the *cost* based on *vmCount* (number of service unit instances) and the *cost per virtual machine* (assumed 0.12 EUR per machine per hour). Summing up at the Service Level the *cost* obtained from the Service Topology level, we obtain estimated hourly service *cost*. To monitor the service performance, we define multiple metrics, averaging the number of service *numberOfClients/h* (active connections), *responseTime* and *throughput* at every Service Unit level and propagating them to the upper service dependency model levels. For monitoring the service cost efficiency, a metric is defined at service level, dividing the *cost* by the value of the *numberOfClients/h* metric, obtaining the estimated *cost/client/h*. Figure 5 presents the MELA cross-layer metric monitoring snapshot obtained from composing and propagating low level metrics. Obtaining higher-level composite metrics, the service controller can define user-defined elasticity requirements on them, analyze if they are fulfilled, and, if not, enforce them by adding/removing VMs to/from service unit instances.

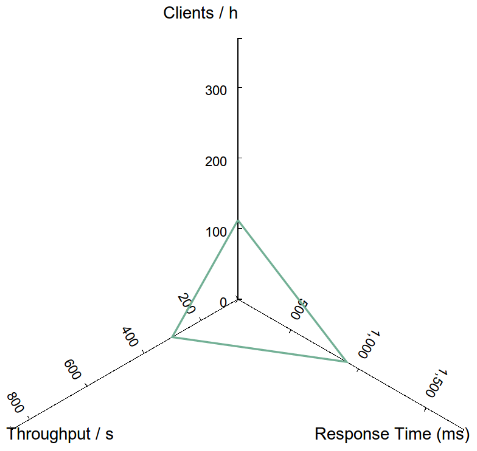
B. Elasticity Space Analysis

The second feature of MELA is the elasticity space analysis, which determines what are the behavioral boundaries in which the service fulfills supplied requirements. Continuing with the previous scenario, the cloud service provider wants to implement a 2.5$ monthly subscription for each service client (sensor). Using MELA, the provider can monitor the service elasticity behavior (via metrics) at each service level, to guarantee he/she does not end up paying more than 2.5$ per month per client to the cloud provider. Assuming a month of 30 days each with 24 hours, the elasticity requirement is a cost of at most 0.0034$ per served client per hour. Using our elasticity space function prototype, we determined the elasticity space, including the maximum and minimum encountered values for the service metrics in which the service

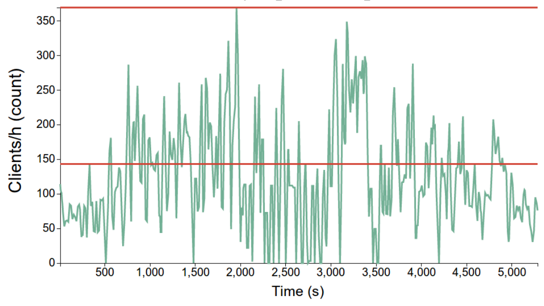
[4]Detailed MELA API, prototype implementation, and experiments can be found at http://dsg.tuwien.ac.at/research/viecom/mela/

[5]http://www.openstack.org/

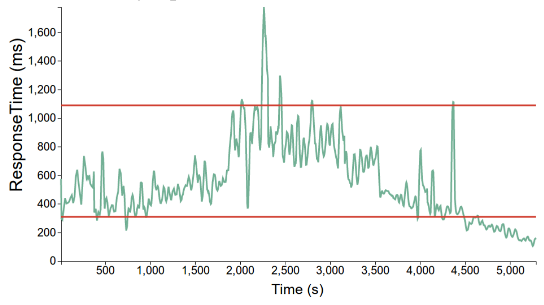
[6]http://ganglia.sourceforge.net/



(a) Elasticity Space Snapshot



(b) Elasticity Space "numberOfClients" Dimension



(c) Elasticity Space "responseTime" Dimension

Fig. 6: Elasticity Space for Event Processing Service Topology

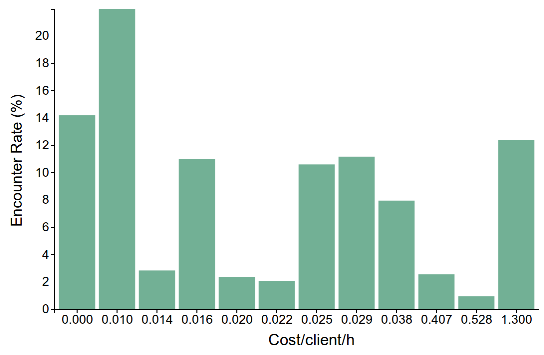


Fig. 7: Elasticity Pathway of CloudService *cost/client/h*

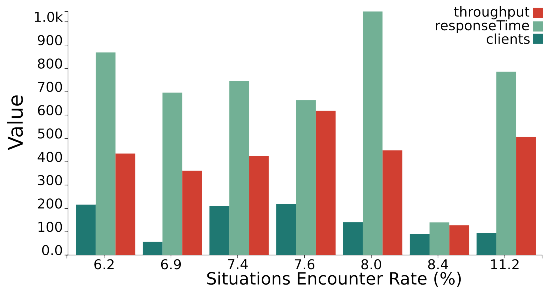


Fig. 8: Dominant Elasticity Pathway of Event Processing Service Topology

requirements are fulfilled, for each service level and monitored element.

From the snapshot of the elasticity space for the Event Processing Service Topology (Fig. 6a), one can examine how individual metrics evolve over the time. For example, we analyze the elasticity space *numberOfClients* dimension, marking with flat lines the minimum and maximum acceptable values (Fig. 6b). The *numberOfClients* minimum is 172, which multiplied by 0.0034, gives a 0.48$ per hour, indicating that the service uses at least 4 VMs at 0.12$ per hour. From this evaluated elasticity boundary we can deduce that given the current service control strategy, there should be at least 172 clients per hour to justify the cost of running in cloud. To learn more about the service behavior and what are the boundaries the service controller should enforce on different levels, boundaries are also extracted for the *responseTime* describing the time to process a request, and *throughput* which describes the number of operations performed by the service per second (a client can perform many operations). Focusing on *responseTime* (Fig. 6c), MELA extracts a minimum boundary, which might seem weird at first, as we would expect to always want minimum response time. In this case, this evaluated elasticity boundary tells us that a too low response time means that the service has underused virtual machines, which is not cost effective. This conclusion can be reached by further investigating the elasticity boundaries for the rest of the service levels, which we do not expand here due to lack of space.

Continuing with the scenario, knowing the boundaries for different service levels and monitored elements, the cloud service provider wants to extract service characteristics that indicate if the usual service behavior is within the determined space boundaries. Towards this we apply our elasticity pathway function prototype which groups combination of different metric values as DOMINANT, NON-DOMINANT and RARE, according to the rate at which they are encountered in the monitored data. Such a classification highlights the usual behavior of the service and the correlations between the analyzed metrics. Applying the pathway on the *cost/client/h* cloud service level metric (Fig. 7), from the encounter rates and the metric values, it is visible that the *cost/client/h* is less than 0.0034$ only in approximate 76% 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 out why, the provider uses the MELA multi-level analysis feature to focus on the Event Processing Service Topology. The elasticity pathway is applied on the *numberOfClients/h*, *responseTime* and *throughput* (operations) metrics, the metrics influencing the service performance, and thus, the number of served clients and the service cost efficiency. From the DOMINANT behavior (Fig. 8), it is visible that by summing up the encounter rates, in approximate 35% of the encountered situations, the service had less than the required 172 clients. Analyzing further, only in 8% of these situations we detect few clients with low throughput, while in the rest 27% we detect few clients but high response times and throughput, indicating a potential performance bottleneck. To determine the correct cause of the high cost per client, the cloud service provider needs to further investigate the elasticity space and pathways of the other monitored elements, from topology level information such as latency or cost for the Data Service Topology, to the service units, and low level performance (CPU usage, memory usage, latency) of associated VMs.

The importance to monitor, analyze and characterize the service behavior at multiple levels was highlighted in this experiment, and the MELA features of multi-level metric composition and service behavior analysis where showcased on monitoring and determining the source of high service cost per client.

VI. Related Work

In [3] the authors describe a framework for scalable real-time data collection and aggregation, [7] predicts performance anomalies by monitoring virtual machines, and in [8] the authors focus on monitoring and predicting cloud performance degradation due to bursting loads. Our tool differs from these approaches as it links service level to virtual infrastructure level monitoring data, providing a complete view over the monitored cloud service behavior.

The authors of [9] improve existing monitoring systems using custom metric aggregation scripts and service model information, while we provide a customizable mechanism for mapping metrics to the service model and deriving higher level information. A mechanism for adapting cloud allocation is presented in [2], using an aggregator that monitors the workload at each service tier, while we employ a generic approach that can analyze a wide array of services.

In [10] the authors monitor cloud resource usage for the infrastructure owner, while we adopt the cloud infrastructure user perspective. The authors of [5] present a comprehensive monitoring system collecting both virtual infrastructure and service level information, focusing on monitoring data delivery. Monitoring performance and data delivery is also the focus of [11], presenting a solution tailored for a specific virtualization framework. An elastic monitoring framework for cloud infrastructures is presented in [12], providing a powerful query mechanism for retrieving service level information. In our approach we also enrich monitoring data with derived metrics, and perform service behavior analysis towards extracting elasticity characteristics. In [4] the authors present an adaptive cloud monitoring system which provides estimations on monitoring accuracy, which could act as data source for our tool, as we focus on monitoring data structuring and analysis.

VII. Conclusions and Future Work

To support the monitoring and analysis of elasticity of cloud services, this paper introduced elasticity space and elasticity pathway as novel concepts characterizing behavior of elastic cloud services. We have presented MELA which supports real-time multi-level analysis of elastic cloud services. MELA also provides several features for customizing and integrating different elasticity analysis functions to support the analysis of other complex elastic behaviors.

For the future we will enhance our prototype by incorporating cloud infrastructure elasticity analysis with our elastic cloud service analysis. We will enhance MELA with pattern based elasticity space and pathway functions. Furthermore, we will work on automatic extraction of elasticity dimensions dependencies and automatic discovery of cloud service's elasticity space boundaries, as a means of learning about the service boundaries and behavior.

References

[1] S. Dustdar, Y. Guo, B. Satzger, and H. L. Truong, "Principles of elastic processes," *IEEE Computing*, no. 5, pp. 66–71, 2011.

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

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

[4] 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*, ser. CLOUD. IEEE, 2012, pp. 951–958.

[5] G. Katsaros, G. Kousiouris, S. V. Gogouvitis, D. Kyriazis, A. Menychtas, and T. Varvarigou, "A self-adaptive hierarchical monitoring mechanism for clouds," *Journal of Systems and Software*, vol. 85, no. 5, pp. 1029 – 1041, 2012. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121211002998

[6] D. J. Dean, H. Nguyen, and X. Gu, "Ubl: unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems," in *International Conference on Autonomic Computing*, ser. ICAC. ACM, 2012, pp. 191–200.

[7] Y. Tan, H. Nguyen, Z. Shen, X. Gu, C. Venkatramani, and D. Rajan, "Prepare: Predictive performance anomaly prevention for virtualized cloud systems," in *International Conference on Distributed Computing Systems*, ser. ICDCS, 2012, pp. 285 –294.

[8] Q. Wang, Y. Kanemasa, M. Kawaba, and C. Pu, "When average is not average: large response time fluctuations in n-tier systems," in *International Conference on Autonomic Computing*, ser. ICAC, 2012, pp. 33–42.

[9] J. Shao, H. Wei, Q. Wang, and H. Mei, "A runtime model based monitoring approach for cloud," in *International Conference on Cloud Computing*, ser. CLOUD, 2010, pp. 313 –320.

[10] M. Dhingra, J. Lakshmi, and S. K. Nandy, "Resource usage monitoring in clouds," in *International Conference on Grid Computing*, ser. GRID. ACM/IEEE, 2012, pp. 184–191.

[11] M. Kutare, G. Eisenhauer, C. Wang, K. Schwan, V. Talwar, and M. Wolf, "Monalytics: online monitoring and analytics for managing large scale data centers," in *Proceedings of the 7th international conference on Autonomic computing*, ser. ICAC. ACM, 2010, pp. 141–150.

[12] B. Konig, J. Alcaraz Calero, and J. Kirschnick, "Elastic monitoring framework for cloud infrastructures," *IET Communications*, vol. 6, no. 10, pp. 1306–1315, 2012.