On Analyzing Elasticity Relationships 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*—With the increasing cloud popularity, substantial effort has been paid for the development of emerging *elastic cloud services*, consisting of different units distributed among virtual machines/containers in different clouds. Due to the software stack and deployment complexity in single and multi-cloud scenarios, developing and managing such services is impeded by a lack of tools and techniques for understanding the elasticity relationships among individual service units, which influence the service's overall elasticity. In this paper we characterize the *elasticity relationships*, and develop mechanisms for analyzing them, based on service monitoring information and elasticity requirements. From collected monitoring information we abstract the elasticity behavior of the whole cloud service and individual units, over which we design a customizable algorithm for relationships analysis. We illustrate our approach via several experiments with an elastic data service for M2M platforms, highlighting the importance of determining elasticity relationships for the development and operation of elastic services.**

***Keywords*-elastic service, relationship analysis, cloud computing**

I. Introduction

Due to the increasing number of available technologies for developing cloud services, from hypervisors and virtual containers to platforms, cloud services are becoming more and more complex. Service developers are able to run service units on top of virtual containers (e.g., Docker[1]), distributed among virtual machines in different clouds. However, in such cloud services, individual service units are typically not behaving independently. Instead, due to communication dependencies (e.g., unit A sends/retrieves data from unit B) or run-time control dependencies (e.g., data end re-balancing after scaling), there exist different relationships between service units, influencing their run-time behavior.

We will refer to such relationships, which affect the run-time elasticity of the service, as *elasticity relationships*. Particular relationships can be of interest for particular stakeholders, including service owners, developers, and elasticity controllers. For example, a relationship between performance and resource usage could be used by a developer to estimate the maximum achievable performance before the resource becomes a bottleneck. Another relationship between cost and performance could indicate how much is a service owner expected to pay for certain performance. We have seen that existing tools for analyzing elasticity of cloud services focus on individual performance metrics [1], or discovering elasticity boundaries for individual service units [2]. However, relying only on information provided by these tools, service developers are unable to discover hidden design issues with future service elasticity, which can be captured by relationships between apparently unrelated service units. Moreover, current elasticity controllers can evaluate only the impact of their decisions on individual units, and are unable to understand how enforcing one elasticity capability on one unit affects the other units in the service. Thus, starting from service monitoring information, we must further analyze and understand if there exist relationships between individual service units, towards assisting the development and refinement of elastic cloud services and controllers.

However, analyzing such relationships is challenging. First, due to the potential complexity of the service's software stack, each software layer can introduce different relationships. Second, due to possible multi-cloud service deployments, the relationships can vary with different cloud providers. Thus, there is a need to investigate new concepts and techniques for determining and analyzing elasticity relationships in complex multi-cloud elastic services, based on collected monitoring information.

To this end, we focus on determining, based on monitored metrics, relationships between any of the service's performance, cost, and resource usage. For this, we characterize *elasticity relationships* of elastic cloud services, based on which we apply various analysis techniques for determining them. In this paper we make the following contributions:

- characterization of elasticity relationships of cloud services
- a mechanism for analyzing elasticity relationships based on service monitoring information
- a framework for run-time analysis of elasticity relationships of cloud services

We evaluate our approach on an elastic service deployed in single and multi-cloud configurations, on both private and public cloud providers.

The rest of this paper is structured as follows. Section II presents the motivation and approach, and discusses related work. Section III introduces the concept of *elasticity relationship* and our approach for discovering and analyzing elasticity relationships. Section IV describes our prototype. Section V

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

[1] https://www.docker.com/

978-1-4799-4093-6/14 $31.00 © 2014 IEEE
DOI 10.1109/CloudCom.2014.93

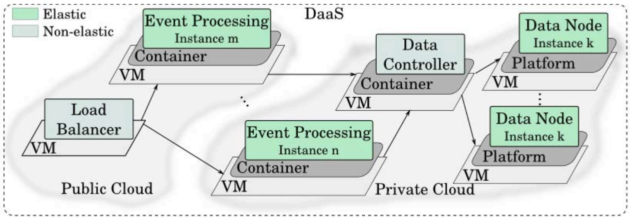


Fig. 1: Elastic multi-cloud data-as-a-service (DaaS)

presents the experiments. Section VI concludes the paper and outlines the future work.

II. Motivation and Related Work

A. Motivation

Let us consider a realistic elastic data-as-a-service (DaaS) application for an M2M cloud platform-as-a-service (Fig. 1), having as elasticity capabilities addition and removal of `Data Node` and `Event Processing` instances. At run-time, an elasticity controller scales the service using these capabilities, according to elasticity requirements defined over various monitored metrics, e.g., `response time` $\leq$ `100 ms` for the `Event Processing`, and `cpu usage` $\leq$ `90 %`.

The DaaS provides data storage and exchange services for Machine-to-Machine (M2M) gateways, such as smart cities or vehicle fleets. Data received from gateways is processed by `Event Processing` units running in virtual containers hosted on virtual machines, and is stored in a distributed data end running a distributed data store platform. Due to data privacy concerns, the data end units are hosted in a private cloud, while event processing instances can run both in private and public cloud providers.

From Fig. 1 we can see that elasticity relationships should exist between units which communicate directly, such as `Load Balancer` and `Event Processing`. However, other relationships might not be so obvious, being generated by indirect communication, such as between `Load Balancer` and `Data Node`. Depending on the service, the relationship's interpretation can also differ, a relationship between metrics belonging to the same individual unit being potentially less important than if determined between two different units.

Focusing on the DaaS potential elasticity relationships (TABLE I), due to communication dependencies, a relationship could be present between monitored `cpuUsage` on the `Data Node` units, and `responseTime` of the `Event Processing` units, indicating if the data end is a bottleneck or not. Another relationship could exist between `throughput` on `Event Processing` unit, and `cpuUsage` on `Data Controller`, indicating what is the maximum achievable `throughput` before `cpuUsage` is too high. While the previous relationships are direct, we can also have indirect relationships, such as the `connectionRate` on `Load Balancer` influencing `throughput` on `Event Processing`, which in turn influences `cpuUsage` on `Data Node` units. Finally, beside one-to-one relationships, we can also have many-to-one relationships. For example, if the previous requirements are used to scale the DaaS, the overall DaaS `cost` could depend on both requirements' metrics, `cpuUsage` on `Data Node`, and `responseTime` of the `Event Processing` units.

Elasticity relationship (*element:metric* → *element:metric*)
DataNode: cpuUsage → EventProcessing: responseTime
EventProcessing: throughput → DataController: cpuUsage
LoadBalancer: connectionRate → EventProcessing: throughput
EventProcessing: throughput → DataNode: cpuUsage
DaaS:cost → DataNode: cpuUsage & EventProcessing: responseTime

TABLE I: Potential DaaS elasticity relationships

Specific stakeholders could be interested in specific relationships. For example, a DaaS provider might be interested in cost relationships, to better plan their business. Service developers might be interested in performance relationships, which they can use to adjust the service to eliminate bottlenecks or reduce resource underutilization. Various parameters of elasticity relationships can further be interpreted by software controllers, ensuring better automated control decisions. For example, an elasticity controller would benefit from understanding that after scaling out the event processing end, the data end might not be able to handle the increasing number of requests, and thus, will become in turn a performance bottleneck.

While current tools [3], [4] can show metrics related to performance, cost, or resource usage of individual service units, or give indicators about the future evolution of such metrics [5], they do now answer the following crucial questions:

- what metrics are involved in elasticity relationships
- what are the functions describing the relationships
- how are the relationships affected by different clouds

To this end, we develop a mechanism for analyzing elasticity relationships of cloud services based on the service's monitored behavior abstracted w.r.t its elasticity requirements.

B. Background

As we aim to determine elasticity relationships, we expect that "elasticity" means different things for different cloud services and units, according to specific requirements. To this end, we denote with *Elasticity Metric* any monitored service metric which can be used to determine if the service is elastic or not. Following the multi-dimensional principle of elasticity [6], an elasticity metric belongs to one of the elasticity dimensions: *Cost*, *Quality*, or *Resources*. Different services and their units could have different elasticity metrics, such as response time for an elastic web service, or data access latency for a data repository.

Elastic services have elasticity requirements associated to elasticity metrics, describing their desired behavior. Thus, in determining elasticity relationships, we consider such requirements, and use the concept of *Elasticity Boundary*, introduced in [2], for representing requirements that bound the values of one or more metrics. An *Elasticity Boundary* has the form $ElBoundary(m) = \langle m^u, m^l)\rangle$, where m^u and m^l denote the upper and lower bound over the allowed values of metric m.

C. Related Work

Analysis of elastic services is approached from two perspectives in current research: (i) service monitoring and identification of abnormal events, and (ii) determining relations among different monitored metrics.

Doelitzscher et al. [7] use neural networks analysis on customer's usage behavior to identify anomalies in services running on IaaS clouds, and validate their approach using a cloud simulator. Trihinal et al. [3] introduce a monitoring framework for elastic cloud services, providing dynamic addition/removal of monitoring metrics and virtual resources during run-time. He et al. [1] propose a cloud services monitoring framework analyzing monitoring information and detecting abnormal behavior, while Venzano et al. [8] study traffic patterns on a private cloud, highlighting that relationships between metrics are influences by network, virtualization layer, and VM collocation. We differ as we do not focus on monitoring, and instead rely on existing solutions to collect monitoring information used in analyzing elasticity relationships.

Gullhav et al. [9] 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. Lloyd et al. [10] correlate physical and virtual machine resource utilization statistics to predict application performance across VMs, while Mdhaffar et al. [11] analyze different architectures and deployment patterns for complex event processing frameworks. Singh et al. [5] focus on estimating in terms of absolute values the behavior of distributed services when the underlying infrastructure changes, and Ding et al. [12] extract application dependency paths from the application-layer connectivity graph inferred from passive network monitoring data. Xiong et al. [4] introduce vPerfGuard, a framework for service performance diagnosis in consolidated cloud environments, automatically discovering metrics which are most descriptive of service performance, and adaptively detecting changes in performance

We differ as we analyze both direct and indirect relationships based on the elasticity behavior of the service, and not absolute metric values, crucial in analyzing elastic services which scale up/down, potentially bursting in different clouds.

III. Analyzing cloud services' Elasticity Relationships

A. Classifying elasticity relationships

Depending on the service's software stack and cloud deployment, various elasticity relationships can exist at different software layers between service units. Thus, we must be able to analyze multiple service types, from simple single-cloud services, to complex services running multiple units in virtual containers distributed among virtual machines hosted in different clouds. To this end, we use as input the model for representing elastic cloud services presented in [13], describing a cloud service as composed of service units (i.e. functional blocks) logically grouped in service topologies (Fig. 2). Any of the units, topologies, or whole service is considered an *Elastic Element*, as each can have elasticity metrics, requirements, and capabilities.

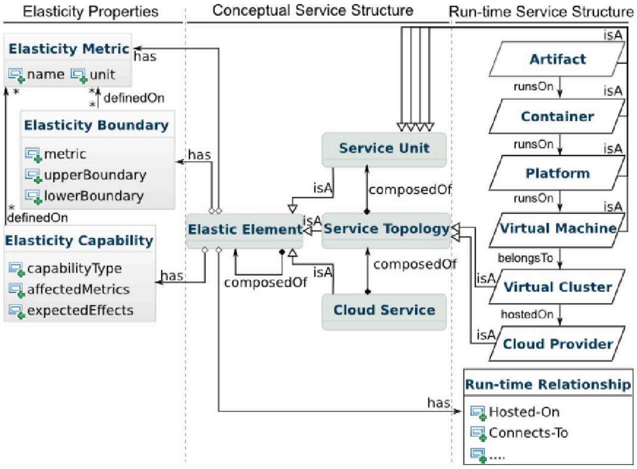


Fig. 2: Elastic cloud service

Usually, elasticity of cloud services is driven by quality, cost, resource usage, or a combination of the three. Moreover, service owners usually view their services from a perspective driven by cost, quality, or both. As different elasticity relationships can exist between different perspectives, we classify relationships after the two fundamental business dimensions, *Cost* and *Quality*, and the three elasticity dimensions, *Quality*, *Cost*, and *Resources* (TABLE II). The relationship category is given by the type of monitoring information used to determine it, different categories being potentially of interest to different stakeholders. Service developers and elasticity controllers might be interested in *Quality dependency* or *Resource quality* relationships, which they can use to eliminate bottlenecks or reduce resource underutilization. A DaaS owner might be interested in cost-related relationships, such as *Cost effectiveness*, *Benefit-Cost dependency*, or *Cost composition*. Various parameters of elasticity relationships can further be interpreted by intelligent software controllers, ensuring better control decisions, such as understanding that after scaling out the DaaS event processing end, the data end might not be able to handle the increasing number of requests.

B. Elasticity relationship

Elasticity of cloud services is driven by elasticity requirements, which specify boundaries over the service's metrics. To fulfill these requirements, elastic services change their structure and used virtual resources at run-time through reconfiguration actions. Due to this reconfiguration, we should not determine relationships based on absolute monitored values, as such relationships might not hold after a reconfiguration. Instead, we determine relationships based on the service's behavior with respect to its elasticity boundaries, which, by abstracting from the absolute monitored values, can be used to describe the service behavior under different configurations.

Thus, based on the previous model and the elasticity bound-

Category	Relationship	Interested stakeholder	Usefulness
Quality dependency	*Quality→Quality*	Developer, Controller	Indicates potential quality/performance bottlenecks
Benefit-Cost	*Quality→Cost*	Owner, Developer, Controller	Indicates potential resource bottlenecks
Resource quality	*Quality→Resource*	Owner, Developer	Describes expected quality/performance when using certain resources
Cost effectiveness	*Cost→Quality*	Owner, Developer	Describes expected quality/performance under certain cost scheme
Cost composition	*Cost→Cost*	Owner, Developer	Describes the cost elements contributing to overall service's cost, indicating potential cost hot spots
Cost utility	*Cost→Resource*	Owner, Developer	Indicates potential resource bottlenecks under certain cost schemes

TABLE II: Elasticity relationships

ary concept, we define an elasticity relationship, as follows:

Definition 1: An *Elasticity relationship* between one elastic element and a set of elements describes the change in the behavior of the first element w.r.t. its elasticity boundaries, triggered by a change in the behavior of the other elements.

The most important for a relationship is determining the change function describing how much the values of the elasticity metrics of one element change, w.r.t. metric's boundaries, when the values of the metrics monitored on other elements change. According to the internal processes executed by each element, the change function might be might be observed at run-time with a certain delay, and could attenuate over time.

Considering these issues, we capture an elasticity relationship *ElRelationship* between one or more elastic elements from a set ,$ElasticElements$, as a tuple of functions: $ChangeFct$, $DelayFct$ and $AttenuationFct$ as follows:

$$ElRelationship : ElasticElements \rightarrow (ChangeFct, DelayFct, AttenuationFct) \quad (1)$$

where $ChangeFct$ is the function describing the change in the metrics of the related elements as a result of the relationship, $DelayFct$ is the delay with which the $ChangeFct$ is observed at run-time, and $AttenuationFct$ the attenuation function which diminishes the effect of $ChangeFct$ over time.

We characterize the change function, $ChangeFct$, as taking for input a set of elastic elements $ElasticElements$, and having as output the estimated values for the elasticity behavior $Elasticity$ of elastic element e:

$$ChangeFct_e : ElasticElements \times ... \times ElasticElements \rightarrow Elasticity(e) \quad (2)$$

Relying on the change function, users can estimate the behavior of each element, predicting quality and cost problems, or resource bottlenecks, and improving the overall elasticity of the service, depending on the determined relationships.

C. Elasticity relationships analysis

As elasticity of services is evaluated based on boundaries over the service's metrics defined by elasticity requirements, for determining the $ChangeFct$ (Eq. 2), we need to abstract, from concrete monitored values, the behavior of elastic services with respect to their boundaries. To this end we define the concept of *Elasticity Work* of a cloud service as the current load on the service with respect to its elasticity boundaries, and the *Elasticity Energy* as the difference between the current and maximum acceptable load (upper boundary). Using these concepts illustrated for a single metric in Fig. 3, we can determine relationships between the elasticity energy of two services, and not individual metric values

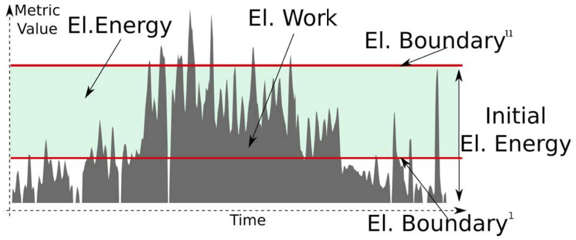


Fig. 3: Elasticity Boundary, Work and Energy concepts

First, for determining the $ChangeFct$ (Eq. 2), we quantify the absolute distance between the upper and lower elasticity boundaries for each elasticity metric of an elastic element using the *Initial Elasticity Energy* ($IElEnergy$):

$$IElEnergy(e) = \{ \|ElBoundary(m)^u - ElBoundary(m)^l\| \mid m \in elasticity\ metrics \in \{Cost, Quality, Resource\}\} \quad (3)$$

where $ElBoundary(m)^u$ and $ElBoundary(m)^l$ denote the upper and lower bound of elasticity metric m belonging to any of elasticity dimensions $Cost$, $Quality$, $Resource$.

Using the $IElEnergy$ we quantify the load monitored on the elasticity metrics of an elastic element w.r.t. its initial elasticity energy using the *Load Unit*, defined as a unit of usage over the energy of a metric in a time frame. Converting monitored values to load units, we capture the load on the elasticity metrics of an element using the *Elasticity Work*, $ElWork$, as the percentage of energy used relative to the initial energy of the element over its metrics:

$$ElWork(m) = \frac{x * LoadUnit(m)}{IElEnergy(m)} \quad (4)$$

where x is the number of load units used per 1 time unit over which the load is measured, from the initial energy $IElEnergy$ of metric m.

$ElWork$ is a result of service's load, or resource usage while idle, based on which we can compute the instant elasticity energy, `ElEnergy`, of an element `e` as the difference between its initial energy, normalized to 100, and the sum of the work done in idle ($ElWork_{idle}$), and in load ($ElWork_{load}$), as follows :

$$ElEnergy(e) = 100 - ElWork_{idle}(e) - ElWork_{load}(e) \quad (5)$$

$ElEnergy$ is used to describe the behavior of the service, a zero energy indicating it violates its requirements, while one close to the initial energy indicates that it is underused. Using $ElEnergy$ for representing the co-domain of the $ChangeFct$ from Eq. 2 (*Elasticity*), from an elasticity relationship between one element e_i and a set of other elements $e_k, ..., e_n$, we can compute the expected values of e_i's elasticity energy at time t by applying the relationship's $ChangeFct$ over the elasticity energy of metrics belonging to each related element, as $ChangeFct^t_{e_i}(ElEnergy(e_k)^t, ..., ElEnergy(e_n)^t)$, considering the $DelayFct$, and $AttenuationFct$ functions.

We develop Algorithm 1 for determining elasticity relationships. Depending on the type of relationships we want to determine, we must be able to investigate from only a subset of metrics, to all collected metrics for all service's elements. To this end, our algorithm can be applied for determining for any metric of interest, the elasticity relationships it has with another set of monitored metrics. For each analyzed elastic metric, the $ComputeElEnergy$ function (Lines 11-18) applies Eq. 5 to compute the elasticity energy over each metric monitored value, considering the monitored value as indicator of complete elasticity work, $elWork$. By applying $ComputeElEnergy$ over all analyzed metrics (Lines 2-6), we obtain for each metric a time series of elasticity energy values. The elasticity energy is determined based on the initial elasticity energy, which can change at run-time due to scaling actions, and energy work, which changes according to the service load. The energy time series provides us with information about the elasticity behavior of the analyzed elements over each analyzed element's metric in time. Over the elasticity time series, various analysis techniques can be applied (Line 8), depending on the type of analyzed relationship.

Algorithm 1 Determining elasticity relationships between metrics belonging to same or different elastic elements

```
Input: $m$ - elastic metric to discover relationships for
Input: $metrics$ - elasticity metrics potentially related to $m$
Output: $relationship$
1  function AnalyzeELRelationships(m, metrics)
2      mElEnergy = ComputeElEnergy (m);
3      metricsElEnergy;
4      foreach $m_i$ in metrics do
5          metricsElEnergy.add(ComputeElEnergy ($m_i$));
6      end
7      ▷ function for analyzing elasticity energy time series
8      return TimeSeriesAnalysis(mElEnergy, metricsElEnergy)
9  end function
10
11 function ComputeElEnergy(metric)
12     elEnergyInTime = [];
13     iELEnergy = $|| m.Boundary^u - m.Boundary^l ||$;
14     foreach $elWork$ in $metric.monitoredValues$ do
15         elEnergyInTime.add(100 - $\frac{elWork}{iElEnergy}$);
16     end
17     return elEnergyInTime;
18 end function
```

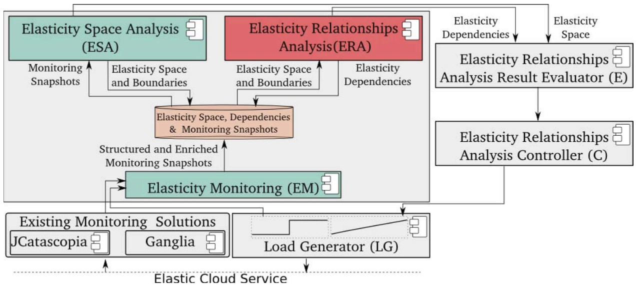

Elasticity Space Analysis (ESA)
Elasticity Relationships Analysis(ERA)
Elasticity Dependencies
Elasticity Space
Elasticity Relationships Analysis Result Evaluator (E)
Monitoring Snapshots
Elasticity Space and Boundaries
Elasticity Space and Boundaries
Elasticity Dependencies
Elasticity Space, Dependencies & Monitoring Snapshots
Elasticity Relationships Analysis Controller (C)
Structured and Enriched Monitoring Snapshots
Elasticity Monitoring (EM)
Existing Monitoring Solutions
JCatascopia
Ganglia
Load Generator (LG)
Elastic Cloud Service

Fig. 4: Elasticity relationships analysis framework

IV. Prototype Implementation

A. Architecture

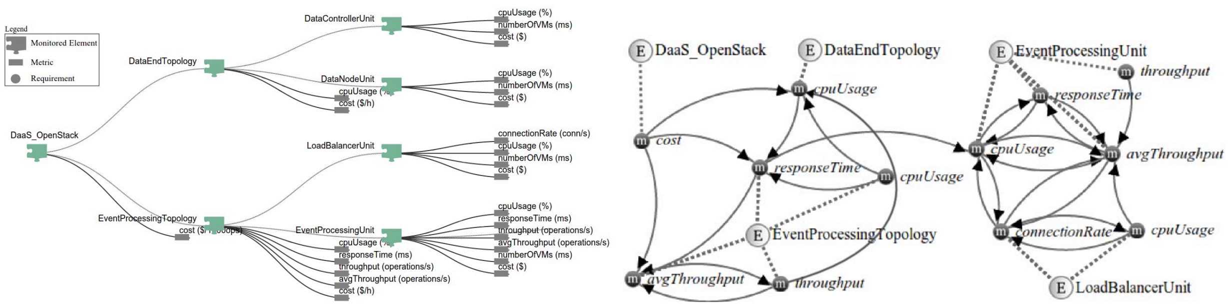
For applying our approach from Section III, we extend MELA [2], an elasticity monitoring and analysis as a service, with a new *Elasticity Relationship Analysis* service implementing our techniques for analyzing elasticity relationships (Fig. 4). MELA already provides an *Elasticity Monitoring* service which collects monitoring data, structures and enriches it, and an *Elasticity Space Analysis* service which uses this data to determine the service's elasticity space and boundaries.

While for the determined relationships we require elasticity boundaries over all cloud service's metrics for computing the service's elasticity energy, they might not be always known. Thus, we use MELA's *Elasticity Space Analysis* service for determining the `Elasticity Space` of the target service from supplied elasticity requirements and collected monitoring information. The elasticity space contains the `Elasticity Boundaries` determined for all elasticity metrics. Based on the determined boundaries and monitoring information, our *Elasticity Relationships Analysis* service uses an array of functions and techniques to determine the service's elasticity relationships. The elasticity relationships' analysis result is evaluated by a *Result Evaluator*, and an *Elasticity Relationships Analysis Controller* orchestrates all components.

B. Functions for determining elasticity relationship

In the current prototype, for determining the elasticity relationships' coefficient functions, i.e., energy change, delay, and attenuation, we apply R[2] functions. To use R, we compute from monitoring information the elasticity energy of each metric at each monitoring interval, obtaining elasticity energy time series over which we apply R analysis functions. To obtain a clear view over the usual behavior of the service, we apply a preprocessing step over the time series and remove

[2] http://www.r-project.org/



(a) DaaS with structured monitoring information

(b) DaaS with determined elasticity relationships graph

Fig. 5: Analyzing DaaS's elasticity relationships

outliers determined by R $mbox$ function. We further determine the delay function $DelayFct$ of an elasticity relationship by computing the lag between the evaluated energy time series using the cross-covariance estimation function ccf.

The change function of each relationship is determined using a linear regression approach, computing the linear correlations between two energy time series with the linear models fitting function lm available in R. The change function is extracted under the form $ChangeFunction(m_{dependent}) = constant{+}coeff_i{*}m_i{+}...{+}coeff_n{*}m_n$, where $m_{dependent}$ is the metric from the relationship whose values can be computed from the values of the other metrics in the relationship, by adding to the $constant$, the values of metrics m_x multiplied by their corresponding coefficients, $coeff_x$.

For each determined coefficient of the linear relationship, we check if the estimation error is one order of magnitude smaller than the coefficient, and if not, we discard the relationship as inaccurate. Finally, we obtain the change function, with the associated $Adjusted\ r$ coefficient of determination, an indicator on how well the extracted relationship fits the original data, from 0% (no fitting), to 100% (maximum fitting). As linear model fitting is used to estimate the values of the $m_{dependent}$ metric based on the related metrics, we evaluate the quality of the estimation by computing the standard, average, maximum, and minimum absolute variance based on the absolute difference between the metric's estimated values based on the relationship, and the monitored values.

V. Experiments

A. Setup

To evaluate the proposed approach, we deploy the DaaS in both single and multi-cloud configurations, on our private OpenStack[3] cloud, and Flexiant[4], a public commercial cloud, using virtual machines of similar types (1 CPU with 1 GB of RAM). The DaaS is structured in two logical topologies, (i) `Event Processing` topology, containing instances of `Event Processing` units and a `Load Balancer`, and (ii) a `Data End` topology containing instances of `Data Node` units and a `Data Controller` acting as data load balancer. The data end is implemented using Cassandra[5], the load balancer using HAProxy[6], and the event processing units as RESTful services. A software controllable `Load Generator` was designed for applying stepwise increasing/decreasing load over the DaaS, simulating sensors which connect and send data to the DaaS.

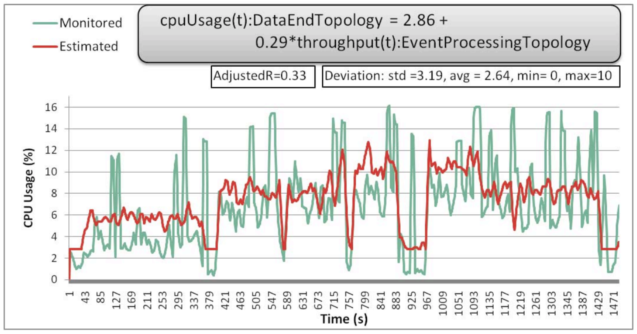


Fig. 6: DaaS on private cloud - Quality relationship

B. DaaS deployed on private cloud

First, monitoring information is structured using MELA (Fig. 5a), the metrics considered important being propagated and associated to each unit and topology. In this case `throughput`, `averageThroughput`, and `responseTime` for `Event Processing` units, and `cpuUsage` for all units, are obtained applying an `average` or `sum` operation on the values monitored for each unit instance running in a virtual machine, and are propagated and associated to each topology. From the `Load Balancer`, `connectionRate` is also collected, and cost per service unit is computed by multiplying the assumed virtual machine cost with the number of machines running instances of each unit.

First, the DaaS is deployed in our private OpenStack-based cloud, with one VM for each service unit. As a service developer, we want to understand if there exists any *Quality* relationship between the throughput on the `Event`

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

[4] http://www.flexiant.com/

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

[6] http://www.haproxy.org/

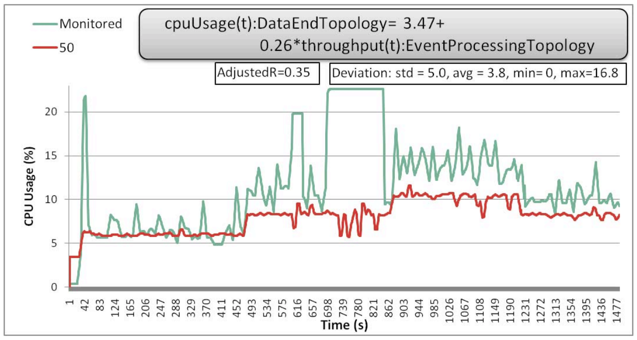


Fig. 7: DaaS on public cloud - Quality relationship

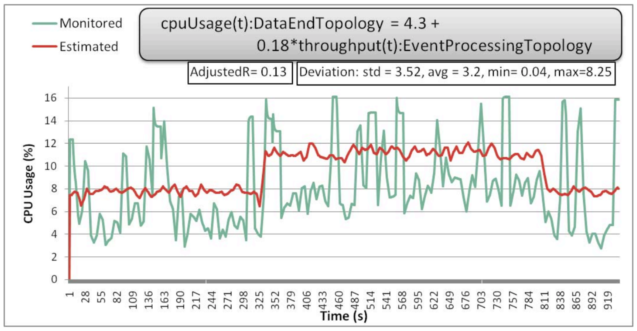


Fig. 8: DaaS on multi cloud - Quality relationship

Processing and the CPU usage of Data End, as to understand if data end CPU usage could be a bottleneck. The Load Generator is used to apply a workload starting with 30 sensors, increasing to 90 in steps of 30, and decreasing to 30 again, according to expected DaaS usage. Each load step takes around 5 minutes, providing enough monitoring information during the same load to enable relationship analysis.

Using our prototype, relationships are determined as linear functions, expressed under the form $metric(t) : element = constant + coeff_i * metric_i(t) \;\; : \;\; element_i + ...$ To understand resource quality, from the determined relationships (Fig. 5b), we focus on the relationship between the cpuUsage of the Data End and throughput of the Event Processing. The determined relationship is a linear equation in which the elasticity energy of cpuUsage at time t can be estimated by multiplying the throughput's energy value at time t with 0.29, and adding 2.86 to the result. Converting the abstract energy to concrete values with respect to the current elasticity boundaries of the service, we can estimate cpuUsage based on the throughput's monitored values (Fig. 6). From the relationship, we estimate that, using this deployment structure, with maximum accepted CPU usage (from elasticity requirements) of around 90%, the maximum achievable throughput is (90-2.86)/0.29 ≈ 300 sensors per second. From the relationship's quality indicators, i.e., standard deviation (std.) of 3.19, average (avg.) of 2.64 and maximum (max.) of 10, this relationship is trustworthy, indicating that when more than 300 sensors are connecting to the DaaS, the data end should be scaled out.

C. DaaS deployed on public cloud

We are further interested if the same relationship holds on the public cloud, and analyze the DaaS deployed on Flexiant public cloud, with same load and number of VMs. Although the DaaS behavior on the public cloud differs in terms of CPU usage pattern (Fig. 7), the same relationship type is detected, cpuUsage(t)=3.47+0.26*throughput(t), with minor differences both in its coefficients, and quality indicators, increasing the confidence that the determined relationship is not generated by particular cloud infrastructures, but instead is present in the service design, and thus, must be considered when controlling the service's elasticity on any cloud.

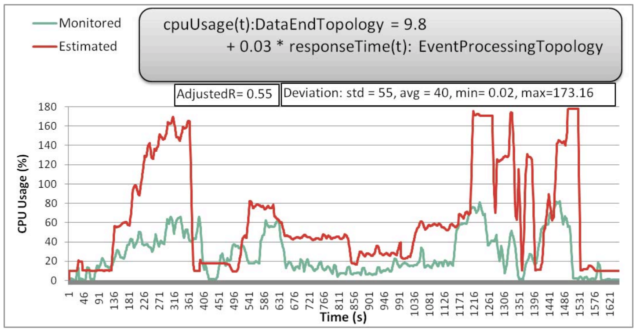


Fig. 9: DaaS with controller - Quality relationship

D. DaaS deployed on multi-cloud

As both evaluations returned similar relationships, we further want to evaluate if the same relationship holds when the DaaS "bursts" into a public cloud due to elasticity requirements. To this end, we deploy the DaaS in a multi cloud configuration, with 2 Event Processing instances on each cloud, and the data end deployed on the private cloud. The load on the DaaS is doubled, as the service is expected to burst in public clouds only during high load periods.

From the same relationship determined for the multi-cloud scenario (Fig. 8), we notice that the coefficient for throughput, 0.18, is smaller than in the single cloud scenario, and thus, it has less influence on the overall CPU usage. This might indicate that other relationships between other metrics are influencing the DaaS, and we would need to investigate also the other determined relationships in order to understand the DaaS's behavior.

E. DaaS deployed on private cloud with elasticity controller

A developer might further want to determine if the previous relationships also hold during run-time elasticity control. Thus, we deploy the DaaS in the private cloud with an attached elasticity controller (in this case RSYBL [13]). Due to requirements of responseTime on Event Processing topology ≤ 100 ms and cpuUsage on Data End ≤ 90%, at run-time, the controller adds/removes unit instances.

In this scenario, the determined relationship (Fig. 9) also includes responseTime, indicating that, during elasticity

Determined Elasticity Relationships			Relationship quality statistics				
Category	No.	Linear relationship function	Adjusted r	Deviation			
				Std	Max	Min	Avg
Resource quality	1	*cpuUsage(t):EventProcessingTopology* = 11.7 + 0.87* *cpuUsage(t):DataEndTopology*	0.55	20.2	51.5	0	17
	2	*responseTime(t):EventProcessingTopology* = 11.5 + 2.35* *cpuUsage(t):DataEndTopology*	0.19	122.9	347.9	0.1	93.2
	3	*cpuUsage(t):LoadBalancerUnit* = 4.9 + 0.53* *connectionRate(t):LoadBalancerUnit*	0.66	63.7	137.2	0.05	52.6
	4	*cpuUsage(t):EventProcessingUnit* = 16.9 + 0.71* *connectionRate(t):LoadBalancerUnit*	0.4	68.7	187.2	0.01	54
Quality dependency	5	*throughput(t):EventProcessingUnit* = 2.5 + 0.56* *connectionRate(t):LoadBalancerUnit*	0.54	41.9	161	0.00	30.6

TABLE III: DaaS during run-time control - Other determined relationships

control, `responseTime` has a contribution on `cpuUsage`, even if small. From the relationship, we notice that the estimated `cpuUsage` goes over 100% between certain time frames, indicating potential bottlenecks. The estimated bottlenecks are not encountered in the monitored `cpuUsage` due to the controller scaling out the data end. From the computed quality indicators, i.e., std. deviation of 55, avg. of 40 and max. of 173.6, we notice that due to enforcing elasticity actions, the determined relationship is not trustworthy, as an average error of 40% in `cpuUsage` is too high.

Thus, we investigate other relationships, captured in TABLE III. The first determined `Resource quality` relationships indicates that `cpuUsage` on the data end influences both the `cpuUsage` on the event processing topology (1), and the `responseTime` (2), meaning it still must be considered as a metric influencing the elasticity of the service. Relationship 3 between the `connectionRate` reported by the `Load balancer` and its `cpuUsage` can be used by the elasticity controller to decide if and when the load balancer should be scaled vertically, depending on the number of connected DaaS users. From the `Quality dependency` relationship 5, we notice that the achieved `throughput` can be estimated to 60% of the `connectionRate` monitored on the load balancer, indicating potential performance problems.

Based on the above elasticity relationships, the DaaS's elasticity could be improved by removing indicated potential bottlenecks, and its elasticity controller redesigned to enforce elasticity actions preemptively, based on estimated values.

VI. Conclusions and Future Work

In this paper we focused on analyzing elasticity relationships in cloud services, enabling different stakeholders, from developers to elasticity controllers, to understand the elasticity relationships governing the run-time behavior of complex cloud services. To this end, we have characterized the elasticity relationships, and developed a mechanism for determining elasticity relationships of cloud services, which can be applied to a large array of service configurations, from single cloud to multi-cloud services with complex software stacks.

We evaluated our approach on an elastic cloud service in single and multi-cloud configurations, on both private and public clouds, with and without an elasticity controller. We have shown that using our approach, a user can easily discover relationships crucial for understanding how service units and topologies influence each other at run-time. We highlighted the need to understand such relationships for different cloud environments and elasticity controllers, as each can can generate different relationships, of interest to different stakeholders. Currently, we plan to enhance our relationships analysis mechanism in order to also discover non-linear relationships, and integrate cloud service patterns in the analysis process.

References

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

[2] D. Moldovan, G. Copil, H.-L. Truong, and S. Dustdar, "Mela: Monitoring and analyzing elasticity of cloud services," in *Cloud Computing Technology and Science (CloudCom), 2013 IEEE International Conference on*, 2013, pp. 80–87.

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

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

[5] R. Singh, P. Shenoy, M. Natu, V. Sadaphal, and H. Vin, "Analytical modeling for what-if analysis in complex cloud computing applications," *SIGMETRICS Performance Evaluation Review*, vol. 40, no. 4, pp. 53–62, Apr. 2013.

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

[7] F. Doelitzscher, M. Knahl, C. Reich, and N. Clarke, "Anomaly detection in iaas clouds," in *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, vol. 1, Dec 2013, pp. 387–394.

[8] D. Venzano and P. Michiardi, "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*, 2013.

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

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

[11] A. Mdhaffar, R. Ben Halima, M. Jmaiel, and B. Freisleben, "A dynamic complex event processing architecture for cloud monitoring and analysis," in *IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, vol. 2, Dec 2013, pp. 270–275.

[12] M. Ding, V. Singh, Y. Zhang, and G. Jiang, "Application dependency discovery using matrix factorization," in *IEEE Iternational Workshop on Quality of Service (IWQoS)*, June 2012.

[13] G. Copil, D. Moldovan, H.-L. Truong, and S. Dustdar, "Multi-level elasticity control of cloud services," in *Service-Oriented Computing*, ser. Lecture Notes in Computer Science, S. Basu, C. Pautasso, L. Zhang, and X. Fu, Eds., 2013, vol. 8274, pp. 429–436.