QUELLE – a Framework for Accelerating the Development of Elastic Systems[⋆]

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

Distributed Systems Group, Vienna University of Technology
{d.moldovan, e.copil, truong, dustdar}@dsg.tuwien.ac.at

Abstract. A large number of cloud providers offer diverse types of cloud services for constructing complex "cloud-native" software. However, there is a lack of supporting tools and mechanisms for accelerating the development of cloud-native software-defined elastic systems (SES) based on elasticity capabilities of cloud services. In this paper we introduce QUELLE – a framework for evaluating and recommending SES deployment configurations. QUELLE presents models for describing the elasticity capabilities of cloud services and capturing elasticity requirements of SESs. Based on that QUELLE introduces novel functions and algorithms for quantifying the elasticity capabilities of cloud services. QUELLE's algorithms can recommend SES deployment configurations from cloud services that both provide the required elasticity, and fulfill cost, quality, and resource requirements, and thus can be incorporated into different phases of the development of SESs. We present several experiments based on real-world cloud services for the development of an elastic machine-to-machine data-as-a-service system.

Keywords: cloud service, software-defined, elasticity capability, elasticity quantification

1 Introduction

Rapid development in cloud computing has introduced diverse types of cloud services offered by a large number of cloud software providers. This leads to several efforts to investigate and develop native cloud systems by leveraging such cloud services in a multi-cloud environment [15,1,14]. In our work, we are interested in the development of cloud-native software-defined elastic systems (SESs), whose functionality is collectively provided by cloud services and whose elasticity capabilities can be controlled via software-defined APIs by intelligent controllers [12]. Services constituting a SES might or might not be software-defined elastic – they might not have elasticity capabilities controlleable via APIs – but overall when combined, we expect the SES to be elastic. This triggers a

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

challenging question on how to quantify the elasticity of these services and the SES to ensure that it meets the user's elasticity requirements.

Although elasticity appears at run-time, through dynamic system reconfiguration with respect to certain requirements, using services providing the necessary elasticity capabilities when constructing SESs is crucial in order to answer the above-mentioned question. For example, we want to avoid using a cloud service which must be reserved for 1 year, when we expect to create and destroy such service instances every hour. Although several frameworks allow the developer to explicitly model such systems, they are often limited to the exact specification of the required cloud services [8], or the amount and type of required resources [9], without considering the elasticity of such services. Currently, a SES developer has to manually search through cloud providers and select services which seem feasible for the system s/he needs to construct, without supporting frameworks to evaluate if their elasticity capabilities support the required SES elasticity.

We believe that, to accelerate the development of SESs, we must be able to quantify elasticity capabilities of cloud services and provide suitable functions for recommending services based on their elasticity, that can be incorporated into different phases of the native cloud software development. In this paper, we introduce novel functions and algorithms for recommending SES deployment configurations using cloud services which provide the necessary elasticity capabilities, and which fulfill resources, quality, and cost requirements. We define an *Elasticity Quantification* function for quantifying the elasticity of cloud offered services. Based on the quantification functions and algorithms, and considering multi-level SES requirements over cost, quality, and resources, we provide a framework for accelerating the construction of SESs by recommending SES deployment configuration using existing cloud services, which can be integrated in third party cloud provisioning frameworks [4] or recommender systems [10]. Overall, this paper presents the following main contributions: (i) models for capturing elasticity capabilities of cloud services and multi-level elasticity requirements of SESs, and (ii) a set of customizable quantification functions and algorithms for evaluating the elasticity of cloud services. These contributions are provided as a set of models, functions and algorithms under the QUELLE (QUantifying ELasticity utiLity Engine) framework, which can be used by developers, automatic cloud composition tools or elasticity controllers to determine suitable SES deployment configurations fulfilling elasticity requirements.

The rest of this paper is structured as follows. Section 2 presents the motivation and approach. Section 3 discusses elasticity quantification of cloud services. Section 4 introduces our algorithms for recommending SES deployment configurations. Section 5 presents our prototype and experiments. We discuss related work in Section 6. Section 7 concludes the paper and outlines the future work.

2 Motivation and Approach

To understand the challenges in constructing *cloud-native, software-defined elastic systems (SES)*, let us consider the development of a cloud-native Data-as-a-

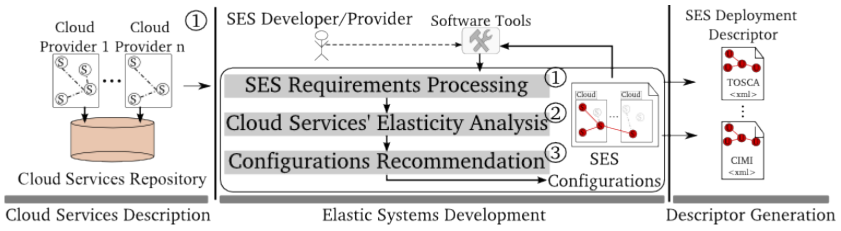
Cloud Provider 1
Cloud Provider n
Cloud Services Repository
SES Developer/Provider
Software Tools
SES Requirements Processing
Cloud Services' Elasticity Analysis
Configurations Recommendation
SES Configurations
SES Deployment Descriptor
TOSCA <xml>
CIMI <xml>
Cloud Services Description
Elastic Systems Development
Descriptor Generation

Fig. 1. Constructing software-defined elastic systems

Service (DaaS)[1], which provides data storage and exchange services for Machine-to-Machine (M2M) platforms, such as smart cities. The system would be built from several cloud services, from basic IaaS VM services, to PaaS complex event processing for sensor data, data storage, and a message oriented middleware for events notifications. A core requirement for the elastic DaaS is that it should be able to be reconfigured at run-time to maintain a performance/cost balance. The development of DaaS is completely based on existing cloud offered services from IaaS to SaaS, and the elasticity capabilities they provide.

To develop the DaaS, in current approaches [10,7], the developer has to manually investigate all services offered by various cloud providers, and evaluate if their elasticity capabilities provide the required elasticity control options. Then, s/he can use existing design and modeling tools such as Winery [8] or MODAClouds [9] to design and deploy the DaaS on cloud infrastructures. Manually selecting each service needed for constructing the DaaS is laborious, complex, and error prone. These problems can be reduced and the development can be accelerated if we could provide features, shown in (Fig. 1), for:

– capturing and modeling elasticity capabilities of services from different ecosystems and multi-level SES requirements (indicated by ①),
– providing service elasticity quantification functions for software development tools (indicated by ②),
– recommending SES configurations, which can later on be mapped to software-interpretable deployment descriptor (indicated by ③).

Due to the complexity of existent services, their components dependencies, and heterogeneity of cloud providers, it is very challenging to develop functions and algorithms for quantifying elasticity of cloud services from multiple service ecosystems. Such functions and algorithms have currently not been developed, thus hindering the automation of the software development for SESs. In this paper, we focus on providing a set of customizable functions and algorithms for quantifying the elasticity of cloud services, under the form of an elasticity quantification framework which can be integrated in semi or fully automated third party SES development and/or provisioning tools.

[1] A non cloud-native version of DaaS - (although designed for and running in the cloud) is available at https://github.com/tuwiendsg/DaaSM2M

3 Quantifying elasticity of cloud services

3.1 Modeling elasticity capabilities of cloud services

Elasticity capabilities of a service can affect how its cost, quality, and resources can be configured during its life-cycle (instantiation or run-time), influencing available control options for particular properties. Moreover, such elasticity capabilities also characterize associations among services that influence service run-time behavior. Therefore, the elasticity capabilities of both individual and associations of services are crucial in providing a base for evaluating which services are suitable for a particular SES's elasticity. Therefore, we must understand and model the elasticity capabilities of cloud services and their dependencies, and quantify the elasticity of cloud services to support the development of SESs.

By studying main cloud providers, such as Amazon EC2[2], Rackspace[3], HP-Cloud[4], and Windows Azure[5], and through other studies [11], we found that elasticity capabilities of cloud services indicate which types of configurations are available and in which phases of the service's life-cycle. While some providers give hints about the capabilities of their services, (e.g., Amazon EC2 spot instances can be replaced faster than reserved instances), existing tools do not capture and evaluate such capabilities. Following the multi-dimensional principle of elasticity [5], we define elasticity capabilities of a service as *configuration possibilities with respect to cost, quality, resources, and associations with other services, and the dependencies among them.* Thus, an elasticity capability defines what resource, cost, quality or associations among services can be created, when (instantiation or run time), and how often the services can be reconfigured.

In order to provide all necessary elements for evaluating if a cloud service provides the necessary elasticity capabilities for a SES's run-time elasticity control, we need to capture when we can use an elasticity capability (elasticity phase), how often can we change it (volatility), and if it can be used standalone or not (dependency type). As most existing cloud services representation models capture resources and quality properties [6],[13], we focus on capturing elasticity capabilities (Fig. 2). An `Elasticity Capability` has an `elasticityPhase`, specifying if the capability is available during the service's `Instantiation-Time`, `Run-Time`, or `Both`. The elasticity dimension associated to the capability is defined by the `elasticityDim` property, and is one of `Cost`, `Quality`, `Resource`, or `Service Associations`. As one capability might indicate multiple configuration possibilities, the elasticity capability has a set of `Elasticity Dependency` instances. An `ElasticityDependency` specifies to which cost, quality, resource, or cloud service a cloud service can be associated using the `to` property. `Volatility` is the most important dependency property, defining the rate at which the dependency can be allocated/deallocated for a particular service (e.g., hourly, or

[2] `http://aws.amazon.com`
[3] `http://www.rackspace.com`
[4] `http://www.hpcloud.com/`
[5] `http://azure.microsoft.com/en-us/`

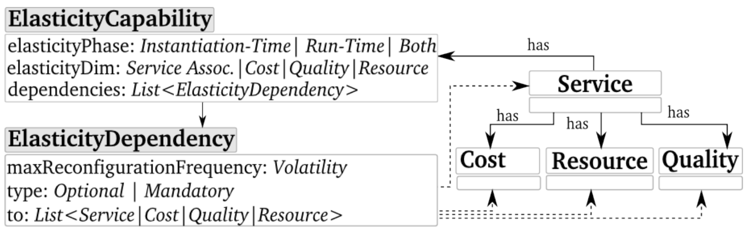
ElasticityCapability
elasticityPhase: Instantiation-Time | Run-Time | Both
elasticityDim: Service Assoc. | Cost | Quality | Resource
dependencies: List<ElasticityDependency>
ElasticityDependency
maxReconfigurationFrequency: Volatility
type: Optional | Mandatory
to: List<Service | Cost | Quality | Resource>
has
Service
has
has
has
Cost
Resource
Quality

Fig. 2. Representing elasticity capabilities of cloud services

monthly), heavily influencing the service's elasticity. For example, a service having dependencies which can be allocated/deallocated hourly is more elastic than one with dependencies which can be reconfigured only on a monthly basis. We describe if a dependency is `Mandatory`, or `Optional` using the `type` property. Mandatory dependencies decrease the elasticity of a service by requiring for the dependency to be always allocated with the service, reducing its usage flexibility.

3.2 Representing elasticity requirements for SES

Based on the previously described elasticity capabilities model, towards accelerating the development of SESs, we provide a set of customizable functions quantifying the elasticity of cloud services, to be used in recommending services which are best suited to the expected SES run-time elasticity control. For this we need to understand and model requirements, run-time properties, and service selection strategies of SESs.

Different stakeholders might have different perspectives over a SES. Using our framework, requirements can be specified at different SES levels, according to the model defined in [2]. An SES is composed of units, logically grouped in topologies (Fig. 3). Elasticity of a SES appears at run-time, through dynamic reconfiguration with respect to SES requirements. Thus, describing and analyzing the expected run-time properties of the SES is crucial in discovering services that support the expected behavior. Through `Runtime Elasticity Properties`, we capture the expected run-time behavior of a SES using `Volatility` and `Dynamism`. Volatility is used in recommending services providing capabilities with the same volatility, while Dynamism describes the rate at which units are expected be allocated/deallocated. For example, we can describe a SES unit which uses its instances on average one hour (volatility), and allocates/deallocates 10 instances per hour (dynamism), due to run-time elasticity control.

A SES might require different elasticity control strategies over its units, topologies, or whole SES, such as maximize performance for a unit, and quality for another. Thus, for selecting services which support the required control, we use `Services Selection Strategies`. We first define `Elasticity-based` selection strategies, which recommend services based on their elasticity capabilities, relying on a set of elasticity quantification functions defined in the next section.

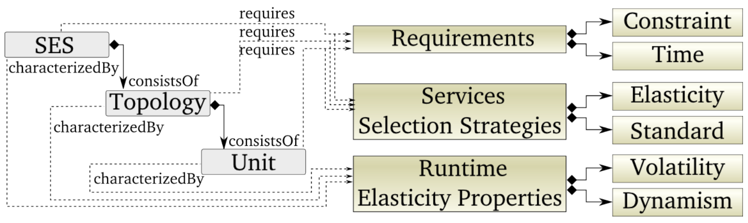

Fig. 3. Representing elasticity requirements for SES

These strategies are crucial in considering the elasticity capabilities of cloud services when building SESs. We define 5 `Elasticity-based` strategies: `Max` {`Overall, Cost, Quality, Resource, Service Association`} `Elasticity`. To also cover traditional user requirements, we support `Standard` strategies. The `Standard` supported strategies are: `Max` {`Fulfilled Requirements, Quality, Resources`}, and `Min Cost`. Multiple different strategies can be specified for each SES unit, topology, or whole SES, covering all potential SES requirements.

In turn, `Requirements` specify the cost, quality and resources required by the SES, and are represented as functions of form $f_{elReq}(constraint, g(time))$, where the `constraint` is a function depending on the cost, quality or resource metric on which the requirement is made, the type of constraint (e.g., greater than) and the required values ($h_{constraint}(metric, operator, value)$). A `time` parameter enables the specification of time-varying requirements.

3.3 Functions for quantifying elasticity of cloud services

Diverse SESs might have different elasticity requirements, depending on SES particularities, and designed elasticity control mechanisms. For example, for one SES, cost elasticity might be more important than quality elasticity. Thus, we provide a set of customizable coefficients for quantifying the elasticity of services, which can be tailored to suit particular SES requirements. Quantifying elasticity enables a numerical ordering of services after their elasticity, crucial in recommending services for SES deployment configurations.

One important factor in evaluating elasticity of cloud services is the phase during the service's lifetime when elasticity capabilities are active: instantiation-time, run-time, or both. Let v_i, v_r, and v_{ir} be user-defined values representing the importance of `Instantiation-Time`, `Run-Time`, and `Both` phases, respectively, for a particular SES; $v_i, v_r, v_{ir} \in [0, 1]$. Thus, we define an `ElPhaseQ` coefficient for quantifying the *phase* in which a service can exhibit elasticity, as follows:

$$ElPhaseQ(phase) = \begin{cases} v_i & \text{if } phase = \texttt{Instantiation-Time} \\ v_r & \text{if } phase = \texttt{Run-Time} \\ v_{ir} & \text{if } phase = \texttt{Both} \end{cases} \quad (1)$$

Typically, to obtain SES configurations with maximum elasticity, v_r should be at least twice as v_i, and v_{ir} their sum (e.g., $v_i = 0.33$, $v_r = 0.67$, and $v_{ir} = 1$).

Elasticity is also affected by association dependencies between services, which can increase (optional associations) or decrease (mandatory associations) the service's elasticity. Let v_o, v_m be user-defined values representing the "importance" of `Optional-Association` and `Mandatory-Association`, respectively, for a particular SES; $v_o, v_m \in [-1, 1]$. We define an `ElDepQ` for quantifying the elasticity dependencies between services as follows:

$$ElDepQ(dep) = \begin{cases} v_o & \text{if } dep = \texttt{Optional-Association} \\ v_m & \text{if } dep = \texttt{Mandatory-Association} \end{cases} \tag{2}$$

Typically, to obtain SES configurations with maximum elasticity, v_o and v_m should have the same value but opposite signs, with $v_m < 0$ as mandatory dependencies decrease elasticity (e.g., $v_o = 1$, and $v_m = -1$).

The `Volatility` of a cloud service heavily influences the service's elasticity, and might have different importance for different SESs. Thus, we consider a custom `VolatilityQ` coefficient for quantifying volatility, supplied as to suit particular SES requirements. Typically, `VolatilityQ` would have the form $numberOfAllowedReconfigurations/timeInterval$.

Thus, we quantify a single elasticity capability of a cloud service as:

$$\begin{aligned} ECQ(C) = {} & ElPhaseQ(C.phase) \\ & * \Sigma_{dep \in C.dependencies}\ VolatilityQ(dep) * ElDepQ(dep) \end{aligned} \tag{3}$$

where C is an elasticity capability, $C.phase$ its elasticity phase, $C.dependencies$ its elasticity dependencies, and dep a single elasticity dependency.

For evaluating the overall elasticity of a cloud service S over all elasticity dimensions (Cost, Quality, Resource, and Services Associations) we define an `Elasticity Quantification (EQ)` function as:

$$EQ(S) = \Sigma_{D \in cost,quality,res,servicesAssoc}\ W_D * \Sigma_{C \in D.capabilities}\ ECQ(C) \tag{4}$$

where D is an elasticity dimension, $W_D \in [0, 1]$ its weight, and C an elasticity capability of S on dimension D. Different W_D coefficients for each dimension D can be set to suit particular SES requirements. For example, a SES interested only in cost elasticity would set W_{cost} to 1, and the other W_D coefficients to 0.

4 Algorithms for recommending SES configurations

Based on elasticity requirements and quantifying functions, we introduce algorithms for generating SES deployment recommendations using existing cloud services, based on their elasticity capabilities. Algorithm 1 evaluates an entity (service, quality, cost, or resource) with respect to a SES unit requirements. The result is a report containing potential services' configurations from available elasticity dependencies, and their fulfilled/unfulfilled requirements. One cloud service might have different mandatory and optional elasticity dependencies on other entities with different properties (e.g., different cost options). Thus, after

Algorithm 1 Evaluating cloud service against SES unit requirements

Input: *entity*,*requirements*; **Output:** *evaluationResult*

```
 1: function EvalEntity(entity, requirements)
 2:     fulfilledReqs = EvalRequirements(requirements,entity)
 3:     if !fulfilledReqs.isEmpty() then
 4:         requirements.remove(fulfilledReqs)
 5:         evaluationResult.add(entity)
 6:         for d in entity.elasticityCapabilities.mandatoryDependencies do
 7:             sEvalResult = EvalEntity(d,requirements)
 8:             requirements.remove(sEvalResult.fulfilledReqs)
 9:             evaluationResult.add(sEvalResult)
10:         end for
11:         for d in entity.elasticityCapabilities.optionalDependencies do
12:             if !requirements.isEmpty() then
13:                 sEvalResult = EvalEntity(d,requirements)
14:                 requirements.remove(sEvalResult.fulfilledReqs)
15:                 evaluationResult.add(sEvalResult)
16:             end if
17:         end for
18:     end if
19:     evaluationResult.unfulfilledReqs = requirements
20:  return evaluationResult
21: end function
```

the algorithm evaluates the static properties of the cloud service (Line 2), it continues by applying the `EvalService` function recursively over its *mandatory* dependencies (must be used). Lines 6-10 determine what unit requirements are fulfilled by these dependencies, and adds them to the solution. The remaining unfulfilled requirements are evaluated against the *optional* dependencies (Lines 11-17), trying to cover as many requirements as possible.

Algorithm 2 applies supplied elasticity quantification functions to evaluate the elasticity of existing cloud services, and, using Algorithm 1, generates a user-specified number of SES deployment configurations. Input SES requirements, run-time properties, service selection strategies, and custom `EQ` functions can be defined at any SES level, from the whole SES, to topologies and units, and are mapped to SES units (Line 2). If conflicts are detected between different levels (e.g. topology and unit), the lower level is considered a specialization, and is applied. For each unit, and each cloud service, the `EvalService` function is applied, obtaining a set of potential services configurations (Lines 6-11). The elasticity of all potential services is evaluated using the `EQ` function defined for that particular SES unit (Line 9). The selection strategies supplied by the user are applied sequentially in selecting from the potential services the first `cfgsCount` decreasingly elastic SES deployment configurations (Line 12).

Quantifying elasticity towards selecting cloud services ensures that during the SES execution, an elasticity controller has the appropriate control options to be enforced depending on SES requirements and run-time behavior.

Algorithm 2 Elasticity-driven SES configurations generation

Input: SES,$services$, $eqFunctions$, $strategies$, $cfgsCount$

Output: $cfgs$ - set of possible SES configurations

```
 1: function RecommendCFGs(SES, services, eqFunctions, strategies, cfgsCount)
 2:     unitsRequirements = mapRequirements(SES)
 3:     for unit in unitsRequirements do
 4:         EQ = eqFunction(unit)
 5:         potentialCfgs = []
 6:         for s in services do
 7:             evalResult = EvalEntity(s,unit.reqs)
 8:             if evalResult != empty then
 9:                 potentialCfgs.add(evalResult.services, EQ(evalResult.services))
10:             end if
11:         end for
12:         configurations.add(unit, potentialCfgs.getFirst(cfgsCount, strategies))
13:     end for
14:   return configurations
15: end function
```

5 Prototype and experiments

5.1 Prototype

We provide the QUELLE framework[6](Figure 4), including the functions, algorithms and models described in Sections 3 and 4. For managing the `Cloud Services Model`, we implemented a graph-based Neo4j[7] `Cloud Services Persistence Adapter`. Graph nodes capture information for categorizing the type of stored entity, and not particular property values; the actual values are captured as relationships properties. This enables selecting services with specific elasticity capabilities. The population of the services repository (see Fig. 1) should ideally be an automatic process, with the increase in cloud providers' description APIs. However, currently we rely on available JSON description services and HTML parsing to populate our model. For SES configurations visualization, we have implemented a TOSCA[8]-based output formatter for Winery[8].

5.2 Evaluating elasticity of Amazon cloud services

Most cloud providers offer only basic cloud services, with reduced configuration and combination options, and implicitly, reduced elasticity, limiting our experiment possibilities. Thus, we focus on a single provider, Amazon EC2, providing 29 IaaS VM cloud services with elasticity capabilities that generate a total of 253 possible configurations, sufficient for showcasing our elasticity quantification functions. Additionally, it provides EBS storage, Monitoring and Messaging services, with individual elasticity capabilities, sufficient for building our DaaS.

[6] Prototype and supplement materials: `https://github.com/tuwiendsg/QUELLE`

[7] `http://www.neo4j.org/`

[8] `https://www.oasis-open.org/committees/tosca`

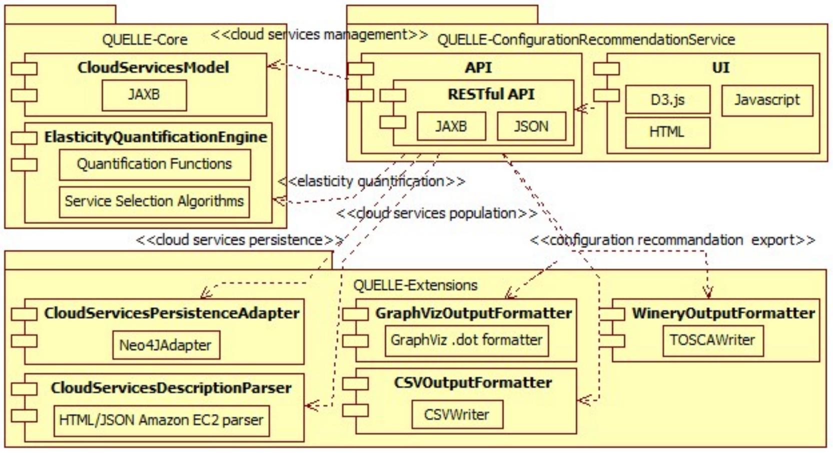
QUELLE-Core
CloudServicesModel
JAXB
ElasticityQuantificationEngine
Quantification Functions
Service Selection Algorithms
<<cloud services management>>
QUELLE-ConfigurationRecommendationService
API
RESTful API
JAXB
JSON
UI
D3.js
Javascript
HTML
<<elasticity quantification>>
<<cloud services population>>
<<cloud services persistence>>
<<configuration recommandation export>>
QUELLE-Extensions
CloudServicesPersistenceAdapter
Neo4JAdapter
CloudServicesDescriptionParser
HTML/JSON Amazon EC2 parser
GraphVizOutputFormatter
GraphViz .dot formatter
CSVOutputFormatter
CSVWriter
WineryOutputFormatter
TOSCAWriter

Fig. 4. QUELLE framework

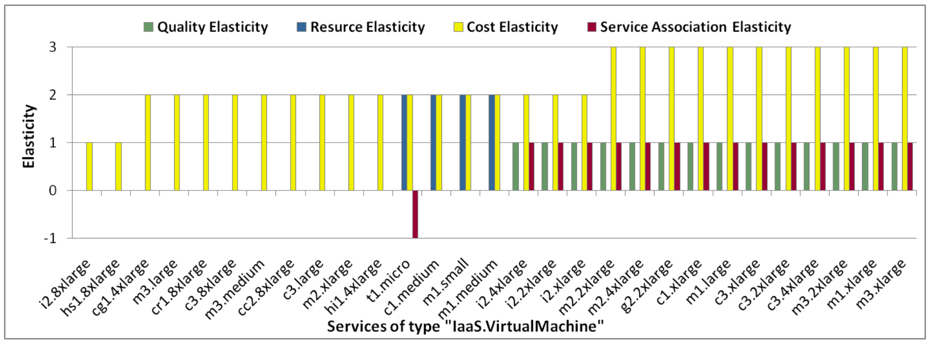

Fig. 5. Elasticity quantification and evaluation of Amazon EC2 IaaS services

As the desired elasticity might vary depending on the stakeholder, our framework provides a customizable elasticity quantification function relying on user-defined `VolatilityQ`, `ElDepQ`, and `ElPhaseQ` coefficients. As the user is interested in building an elastic system, s/he expects services to be allocated/deallocated often. As Amazon bills its services minimum on a hourly basis, the supplied volatility quantification coefficient is `VolatilityQ = 1/minLifetime (Hours)`, generating a volatility of 1 for hourly reserved services, and 1 / (365 * 24) for yearly reserved services. As the user wants to use services which have as few dependencies on other services, s/he supplies an elasticity dependency quantification coefficient `ElDepQ =` {`1 if Optional-Association,` and `-1 if Mandatory-Association`}. Finally, the supplied elasticity phase quantification coefficient is `ElPhaseQ =` {`0.33 if Instantiation-Time, 0.67 if Run-Time, 1 if Both`}, and all elasticity dimensions have same weight coefficient W_d=1.

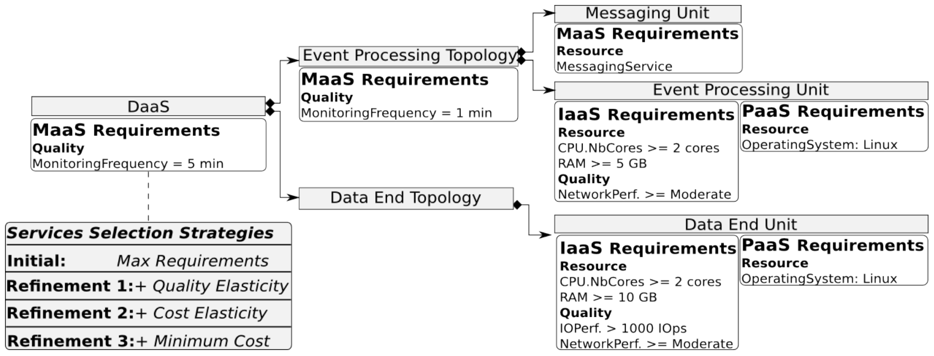

Fig. 6. Multi-level DaaS elasticity requirements

The result of quantifying the elasticity of Amazon EC2[9] services over cost, quality, resources, and services associations is depicted in Fig. 5. As the defined `VolatilityQ` function quantifies close to zero all options of reserving a service for 1 or 3 years, the cost elasticity of most services, such as `m3.large` is quantified close to 2. Amazon EC2 services which have optional dependencies have additional cost and quality control options. Thus, Amazon EC2 IaaS services with can be associated with an `EBS` service have their service association elasticity quantified to $\geq$ `1`, and cost elasticity quantified to $\simeq$ `3`.

5.3 Recommending SES configurations

We aim to accelerate the development of SESs by recommending deployment configurations using cloud services providing the required elasticity for its units. Thus, we define a four phase recommendation process: (i) processing SES requirements, (ii) quantifying elasticity of cloud services, (iii) recommending elasticity-driven SES configurations, and (iv) exporting SES configurations as cloud deployment descriptor.

As a user might not initially know the complete SES requirements, we apply an iterative approach, in which recommended configurations are analyzed by a user, the SES requirements refined accordingly, and resubmitted. First, mixed SES requirements w.r.t. cost, resource and quality, are described by the user in a top-down fashion, from the entire SES to individual units. At the `SES` level, a requirement for a Management as a Service (MaaS) service with a monitoring frequency of 5 minutes is specified, which will be applied to all SES's units. As the units belonging to the `Event Processing Topology` level perform sensitive computation, a MaaS requirement for a monitoring service providing a monitoring frequency of 1 minute is specified, overriding the SES level requirement. The `Event Processing Unit` requires an IaaS service with over 2 CPU cores and 5 GB of RAM, and a Moderate network performance. In turn, the `Messaging`

[9] Services' description accurate at time of writing

Service Selection Strategies	Recommended IaaS Services	Quality Elasticity			Cost Elasticity		
		Avg.	Min.	Max.	Avg.	Min.	Max.
Max Requirements	23	0.6	0	1	2.39	1.0004	3.0004
+ Quality Elasticity	14	1	1	1	2.78	2.004	3.0004
+ Cost Elasticity	11	1	1	1	3.0004	3.0004	3.0004
+ Minimum Cost	1	1	1	1	3.0004	3.0004	3.0004

Table 1. Iterative services selection for Event Processing unit

Service Selection Strategies	Recommended IaaS Services	Avg. Quality Elasticity	Avg. Cost Elasticity
Max Requirements + Minimum Cost	m3.large, m1.large m2.xlarge	0.33	2.33
Max Requirements + Quality Elasticity + Cost Elasticity + Minimum Cost	m1.xlarge	1	3.0004

Table 2. Elasticity-based versus standard service selection for Event Processing unit

`Unit` requires a PaaS service of type messaging. Similarly, the `Data End Unit` requires an IaaS service providing at least 10 GB of RAM, I/O Performance of at least 1000 IOps together with at least a Moderate network performance.

While in the following we focus on IaaS services as they are most abundant and exhibit most elasticity in current cloud computing, we apply the same approach for PaaS and MaaS requirements, as shown at the end of this section.

First iteration: The user submits to QUELLE SES requirements, without elasticity-based selection strategies. Focusing on the `Event Processing Unit`, the user sees that 23 IaaS services were recommended (Table 1), with varying quality and cost elasticity. **Second iteration:** The user adds a `Quality Elasticity` strategy, maximizing the quality options available at run-time. In turn, 14 services are recommended, with quality elasticity equal to 1, as the only modeled quality elasticity capability is an `EBS Optimized` storage option. **Third iteration:** The user adds a `Cost Elasticity` strategy, ensuring the SES can switch between as many pricing schemes as possible during run-time. Thus, 11 services are recommended, with cost elasticity of $\simeq$ `3.004`, due to supplied `VolatilityQ` function evaluating yearly cost schemes $\simeq$ `0`, and hourly pricing schemes (e.g., `Spot`) to `1`. **Fourth iteration:** The user also wants `Minimum Cost`, reducing the recommended services to 1, fulfilling most resource requirements, having maximum quality and cost elasticity, and minimum cost.

In Table 2 we showcase the importance of quantifying elasticity capabilities of cloud services in SES construction, by comparing the usage of `Elasticity-based` service selection strategies with only using the `Standard` strategies `Minimum Cost` and `Max Requirements`. With the standard strategies, requirements are matched and services with minimum cost selected in a traditional fashion, recommending 3 service with varying quality and cost elasticity. By applying elasticity-based strategies, the SES'e elasticity is increased, recommending a `m1.xlarge` service with more control options over its quality and cost elasticity dimensions.

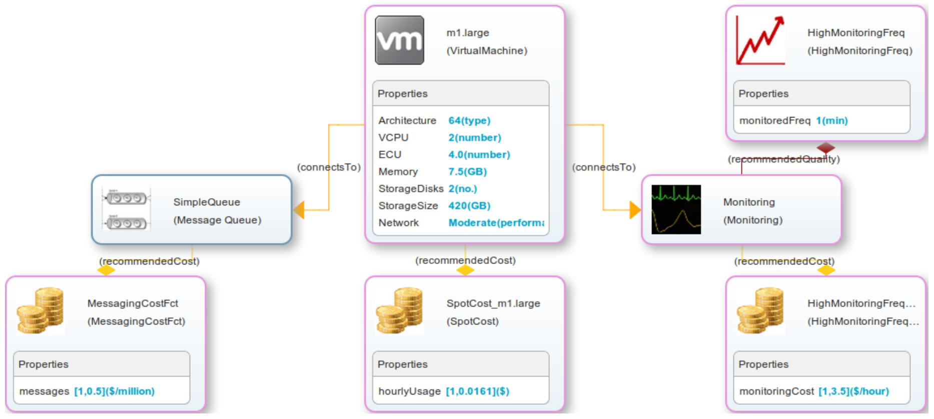

Fig. 7. Complete configuration recommendation for Event Processing topology

Processing all IaaS, PaaS, and MaaS requirements refined above, our prototype generates a TOSCA[10] descriptor containing the recommended SES configuration. For the `Event Processing Topology`, the recommendation is visualized in (Fig. 7) using Winery[8], a TOSCA modeling and visualization tool. The recommendation contains an `m1.large` IaaS service fulfilling the resource and network performance quality requirements with associated `SpotCost`, due the `Minimum Cost` strategy. A PaaS `Monitoring Service` with a `High Monitoring Frequency` is recommended for the monitoring frequency requirement, and a MaaS `SimpleQueue` service for the message oriented middleware requirements. In a similar fashion, recommendations are provided for the `Data End Topology`.

In these experiments we highlighted that, using our framework, a SES developer does not have to search trough all cloud providers for services providing necessary elasticity, and thus, accelerates the SES's time to deployment.

6 Related Work

SES design and cloud provisioning: Tools, such as Winery[8], Slipstream[11], Azure's Octopus Deploy[12] or ModaClouds [9], support construction of cloud services. Such tools require from the user a completely specified SES configuration, and the selected cloud provider. We differ, as we provide recommendations for SES deployment configurations, considering required elasticity capabilities, aiding in the process of choosing cloud services which provide the required run-time elasticity control.

Cloud ecosystem modeling: Several approaches focus on modeling cloud providers towards cloud services provisioning. Goncalves et al. [6] define CloudML,

[10] `https://www.oasis-open.org/committees/tosca`
[11] `http://sixsq.com/products/slipstream.html`
[12] `http://octopusdeploy.com`

a cloud modeling language, describing the resources and functional capabilities of cloud services. Villegas et al. [13] analyze provisioning and allocation policies in IaaS clouds by associating cost of services with their run-time. Wittern et al. [14] capture properties of cloud services and requirements using variability modeling, and integrate human decision-makers, towards filtering cloud services for constructing cloud systems. Most related work focuses on services of VM type and does not evaluate the elasticity of cloud services, while we capture elasticity capabilities of services for all types of services, from IaaS to SaaS.

Cloud service selection: Zhang et al. [15] introduce an ontology-based mechanism for discovery of cloud services based on their functionality and QoS parameters, towards deploying systems in cloud. A mathematical formulation of the cloud service provider selection problem towards maximizing selection benefits withing a given budget is introduced by Chang et al. [1]. Liu et al. [14] use cloud feature models for representing cloud service properties and their relationships, and filter alternative models based on ranking preferences. Dastjerdi et al. [3] use negotiation strategies for selecting VMs with maximum availability and minimum cost. Moving from the VM view, [10] ranks and selects cloud services suitable for building cloud systems using a fuzzy quantification approach. Kamateri et al [7] semantically interconnect heterogeneous PaaS offerings across different cloud providers for deploying cloud systems. The authors of [4] introduce GEMBus, an automated services composition platform providing federated network access to distributed applications and resources towards creating service oriented architectures. We differ as we do not focus only on initial system construction and deployment. Instead, we analyze the elasticity capabilities of selected services, recommending SES configurations which provide the required elasticity capabilities for controlling the SES's elasticity during run-time.

7 Conclusions and Future Work

In this paper we have presented a novel approach for accelerating the development of software-defined elastic systems (SES) by introducing the QUELLE framework which supports the quantification of elasticity capabilities and dependencies among cloud services. We demonstrated that QUELLE can be useful for many situations via the evaluation of elasticity of individual cloud services, and integration of QUELLE into software development phases of elastic systems.

We believe that the introduced models, functions and algorithms will simplify and reduce development effort in complex, diverse cloud service ecosystems. Currently, we are focusing on modeling and evaluating the elasticity dependencies between service units and topologies, and use these dependencies in new functions for the SES development tools. We are also working on the integration of QUELLE into an integrated SES development environment.

References

1. Chang, C.W., Liu, P., Wu, J.J.: Probability-based cloud storage providers selection algorithms with maximum availability. In: 2012 41st International Conference on

Parallel Processing (ICPP). pp. 199–208 (2012)

2. Copil, G., Moldovan, D., Truong, H.L., Dustdar, S.: Multi-level Elasticity Control of Cloud Services. In: International Conference on Service Oriented Computing (ICSOC). Springer Berlin Heidelberg (2013)
3. Dastjerdi, A., Buyya, R.: An autonomous reliability-aware negotiation strategy for cloud computing environments. In: International Symposium on Cluster, Cloud and Grid Computing (CCGRID). pp. 284–291. IEEE/ACM (2012)
4. Demchenko, Y., Ngo, C., Martnez-Julia, P., Torroglosa, E., Grammatikou, M., Jofre, J., Gheorghiu, S., Garcia-Espin, J., Perez-Morales, A., Laat, C.: Gembus based services composition platform for cloud paas. In: European Conference on Service-Oriented and Cloud Computing (ESOCC), pp. 32–47. Springer Berlin Heidelberg (2012)
5. Dustdar, S., Guo, Y., Satzger, B., Truong, H.L.: Principles of elastic processes. IEEE Computing (5), 66–71 (2011)
6. Goncalves, G., Endo, P., Santos, M., Sadok, D., Kelner, J., Melander, B., Mangs, J.E.: Cloudml: An integrated language for resource, service and request description for d-clouds. In: International Conference on Cloud Computing Technology and Science (CloudCom). pp. 399–406. IEEE (2011)
7. Kamateri, E., Loutas, N., Zeginis, D., Ahtes, J., DAndria, F., Bocconi, S., Gouvas, P., Ledakis, G., Ravagli, F., Lobunets, O., Tarabanis, K.: Cloud4soa: A semantic-interoperability paas solution for multi-cloud platform management and portability. In: European Conference on Service-Oriented and Cloud Computing (ESOCC), pp. 64–78. Springer Berlin Heidelberg (2013)
8. Kopp, O., Binz, T., Breitenbcher, U., Leymann, F.: Winery a modeling tool for tosca-based cloud applications. In: Basu, S., Pautasso, C., Zhang, L., Fu, X. (eds.) International Conference on Service Oriented Computing (ICSOC), vol. 8274, pp. 700–704. Springer Berlin Heidelberg (2013)
9. Nitto, E.D.: Supporting the development and operation of multi-cloud applications: The modaclouds approach. In: International Symposium on Symbolic and Numeric Algorithms for Scientific Computing (SYNASC). IEEE (2013)
10. Patiniotakis, I., Rizou, S., Verginadis, Y., Mentzas, G.: Managing imprecise criteria in cloud service ranking with a fuzzy multi-criteria decision making method. In: European Conference on Service-Oriented and Cloud Computing (ESOCC), vol. 8135, pp. 34–48. Springer Berlin Heidelberg (2013)
11. Suleiman, B., Sakr, S., Jeffery, R., Liu, A.: On understanding the economics and elasticity challenges of deploying business applications on public cloud infrastructure. Journal of Internet Services and Applications pp. 173–193 (2011)
12. Truong, H.L., Dustdar, S., Copil, G., Gambi, A., Hummer, W., Le, D.H., Moldovan, D.: CoMoT - A Platform-as-a-Service for Elasticity in the Cloud. In: International Workshop on the Future of PaaS. IEEE (2014)
13. Villegas, D., Antoniou, A., Sadjadi, S., Iosup, A.: An analysis of provisioning and allocation policies for infrastructure-as-a-service clouds. In: International Symposium on Cluster, Cloud and Grid Computing (CCGRID). pp. 612–619. IEEE/ACM (2012)
14. Wittern, E., Kuhlenkamp, J., Menzel, M.: Cloud service selection based on variability modeling. In: International Conference on Service-Oriented Computing (ICSOC), pp. 127–141. Springer Berlin Heidelberg (2012)
15. Zhang, M., Ranjan, R., Nepal, S., Menzel, M., Haller, A.: A declarative recommender system for cloud infrastructure services selection. In: International Conference on Economics of Grids, Clouds, Systems, and Services (GECON), pp. 102–113. Springer Berlin Heidelberg (2012)