CoMoT – A Platform-as-a-Service for Elasticity in the Cloud

Hong-Linh Truong, Schahram Dustdar, Georgiana Copil, Alessio Gambi, Waldemar Hummer, Duc-Hung Le, Daniel Moldovan
Distributed Systems Group, Vienna University of Technology
E-mail: {truong, dustdar, e.copil, a.gambi, hummer, d.le, d.moldovan}@dsg.tuwien.ac.at

Abstract

Platform-as-a-Service (PaaS) should support the design, deployment, execution, test and monitoring of native elastic systems constructed from elastic service units based on multi-dimensional elasticity requirements. In this paper, we discuss fundamental building blocks for enabling multi-dimensional elasticity programming of software-defined elastic systems. We describe CoMoT, a novel PaaS for elasticity in the cloud that is developed based on these fundamental building blocks.

1. Introduction

One of the main challenges for the development of (future) platform-as-a-service (PaaS) is the question of how such a PaaS will support native software-defined elastic systems (SES), of which the functionality is constructed from cloud service units and the elasticity capability is controlled via software-defined APIs. Given the dynamics and diversity of cloud service units, the entire lifecycle of SES, covering design, coding, deployment, testing and execution activities, should be supported by the PaaS. We believe that if cloud systems offer well-defined APIs, at different levels of abstraction, for deploying service units (e.g., beyond the typical virtual machine (VM) and static software deployment), for controlling service units (e.g., adding/reconfiguring (new) Web containers and VMs when needed), and for monitoring and analyzing runtime quality and cost properties (e.g., of interdependent Web services, Web containers and VMs), then different phases of SES engineering, management, and execution will be interwoven. The user would, e.g., compose his/her SES consisting of many service units and decide to deploy the SES based on the requirement and the best cloud information he/she has, while continuously testing and controlling the elasticity of his/her SES to change the SES to meet the requirements and to optimize the running costs and quality.

In this paper, we outline our novel PaaS aiming at supporting dynamic lifecycle development and execution of native SES. The main driver of our PaaS is the concept of multi-dimensional elasticity [1], in which a complex cloud system will be elastic based on resource, cost, and quality associated with service units of the system. This multi-dimensional elasticity concept forces us to investigate a different way of deploying, testing, controlling, and executing service units of a SES based on the elasticity capabilities of itself and its execution environments. In this paper we outline our view on elastic service units and their capabilities, and present fundamental building blocks on the development of monitoring, control, deployment, and testing for SES (Section 2). We outline how these fundamental building blocks are integrated to provide CoMoT for multi-dimensional elasticity programming in the cloud (Section 3). We illustrate our current prototype (Section 4). Finally, we briefly discuss related work (Section 5) and future work (Section 6).

2. Fundamental Building Blocks for Multi-dimensional Elasticity of Cloud Services

2.1. Conceptualizing Elastic Objects and Execution Environments

At runtime, we view a cloud system as a set of elastic objects that can be controlled. Elastic objects (e.g., a Web service) and their execution environments (e.g., a Web container) are described as elastic service units. An elastic service unit (i) has a function, with well-defined interface, (ii) offers different service models, such as provisioning, consumption, quality management, and pricing, (iii) has dependent service units, and (iv) has elasticity capabilities. Figure 1 presents general information associated with an elastic service unit in the cloud. We utilize them to model any type of software, hardware and human resources offered by and built atop cloud systems that can be used as fundamental service units in cloud services.

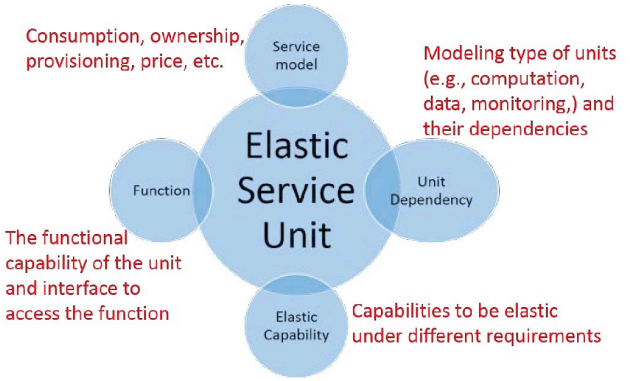

Fig. 1. Model of Elastic Service Units

Diverse types of elastic service units have different elasticity

978-1-4799-3766-0/14 $31.00 © 2014 IEEE
DOI 10.1109/IC2E.2014.44

capabilities, e.g., increasing computational resources, reducing cost, and increasing data accuracy. In our work, these capabilities are explicitly modeled and can be programmed via *elasticity primitive operations* which are mapped to concrete APIs of specific cloud systems and services. Figure 2 illustrates the relationships among elasticity capabilities, types of elastic service units, and cloud APIs.

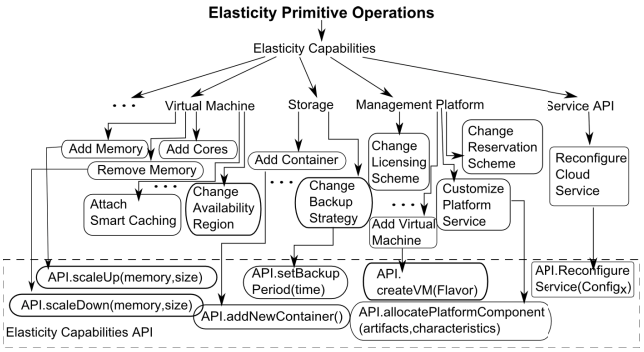

Fig. 2. Elasticity primitive operations

2.2. Programming Software-defined Elastic Systems

Given the conceptual model of elastic service units, we see that cloud service providers should offer their services as elastic service units that facilitates PaaS to support the development of native SES. Furthermore, any developer can also provide his/her elastic service units. To support the programming of SES, we capture available elastic service units under ecosystems of elastic service units. An ecosystem will include elastic service units, which have certain dependency relationships. For example, an ecosystem can include cloud services offered by a cloud provider or a set of service units offered by different providers that can be employed together in a single system. Regarding ecosystems, we can support service unit composition and selection based on elasticity requirements. Based on that, a new SES can be developed. Besides elastic service units supporting required functionality (such as computation, storage, or analytics), SES will also include software-defined management APIs through which the elasticity capabilities of SES and its service units can be programmed at runtime. To represent such SES and their requirements, we devise a model capturing the hierarchical view of SES and its associated elasticity (see Figure 3).

2.3. Controlling Elasticity

For SES, we need to have a high-level view of elasticity specifications that are suitable for the user but the elasticity controller can support. Our general concept is to have a high-level, directive-based language for specifying elasticity named SYBL [2] that supports three fundamental features: (i) specifying which elasticity metrics should be monitored, (ii) establishing constraints on elasticity metrics, and (iii)

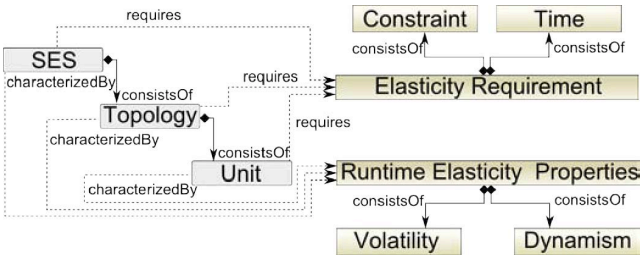

Fig. 3. Software-defined elastic systems

defining strategies for controlling elasticity. At runtime, we utilize monitoring information and the structure of SES to generate suitable elasticity control actions, each control action is mapped to a set of suitable elasticity primitive operations. Based on elasticity control actions, we can support diverse tradeoffs of elasticity, for example, the user might just need to specify the cost and quality constraints and the elasticity controller will have to deal with both computational resources and changes of SES service units to meet these constraints.

2.4. Monitoring and Analyzing Elasticity

Given the elasticity requirement from the user, we need to monitor and analyze elasticity of SES at multiple levels of abstraction, such as individual service units, a set of service units structured into a logical topology, or the whole SES. Furthermore, as elasticity is multi-dimensional, we need to support different types of metrics. Therefore, we develop concepts of elasticity space and elasticity pathway [3]. Elasticity space allows us to capture all relevant elasticity metrics that are required by the user (specified in the user's elasticity requirement) when the SES is in an elastic mode (e.g., increasing the quality due to the increasing paid costs). Elasticity pathway allows us to determine how the elasticity was evolved (based on historical elasticity space) as well as how the elasticity would evolve (through the prediction based on the evolution of elasticity space).

2.5. Deploying Software-defined Elastic Systems

Deploying SES will require the deployment of different elastic service units at different levels of abstraction, such as VM, executables, Web execution environment, and Web services. Generally, we need to work on deployment of the whole SES, topologies of SES and service units in SES. This requires the deployment to integrate elasticity monitoring and analysis features as well as to offer interfaces for elasticity controllers and users to carry out certain elasticity operations, such as deploying new VM and new service units.

2.6. Testing Software-defined Elastic Systems

Since SES will be deployed and executed in an elastic manner, testing SES will require a fundamental change in

designing and executing tests. In our concept of testing elastic systems [4], we see the need to have elasticity testing-as-a-service which can generate test cases suitable for native SES and deploy these test cases in multiple clouds to test the SES. Furthermore, as controlling elasticity is an important issue and part of SES, testing of SES should also test elasticity controllers and monitors to find out if the elasticity capabilities of SES can be properly programmed.

3. CoMoT – PaaS for Controlling, Monitoring, and Testing Elasticity

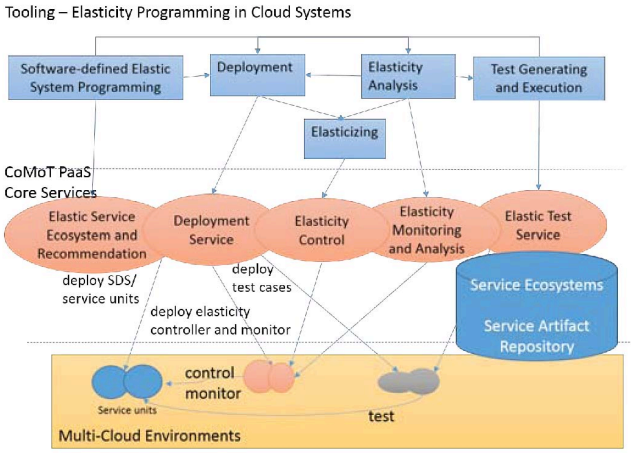

Fig. 4. Overview architecture of CoMoT

Based on building blocks mentioned in Section 2, Figure 4 describes our CoMoT (Control, Monitoring, and Testing) that implements elasticity programming, management and execution for SES. CoMoT is divided into three layers, namely *Tooling*, *CoMoT PaaS Core Services*, and *Multi-cloud Environments*.

- *Tooling*: includes different development and end-user tools for software-defined service programming, deployment, monitoring and testing.
- *Elastic PaaS Core Services*: includes core services for PaaS, such as *Elastic Service Ecosystem and Recommendation*, *Deployment Service*, *Elasticity Control*, *Elasticity Monitoring and Analysis*, and *Elastic Test Service*.
- *Multi-cloud Environments*: includes multiple cloud systems where the developed services will be deployed and executed and where other components of CoMoT will be deployed.

CoMoT PaaS Core Services can be used individually or in combination. The core services also invoke each other. For example, typically, the *Deployment Service* – called Salsa – deploys not only SES and its service units but also elasticity controllers and monitors, and then passes the deployment information to the *Elasticity Control* – based on SYBL [2] – which obtains elasticity monitoring information from the *Elasticity Monitoring and Analysis* – based on MELA [3] – to control the elasticity. However, in order to control the elasticity, *Elasticity Control* can invoke *Deployment Service* to deploy SES service units/topologies. Similarly, *Elastic Test Service* can generate test cases and require the *Deployment Service* to deploy test cases in multiple cloud systems. This way of interactions among core services enables us to support iterative, interactive and interwoven design, deployment, execution, monitoring and testing SESs, completely on the cloud.

4. Illustrating Example

We illustrate some aspects in our current prototype with the development, deployment, and execution of a machine-to-machine (M2M) Data-as-a-Service (DaaS) which includes several service units for event processing, load balancing, and NoSQL-based storage. Using our PaaS, a customer describes his/her requirements and he/she uses our tool to iteratively design the M2M DaaS. After this phase, the system will present a SES design that the user can refine. The user also annotates elasticity controls with the M2M DaaS using SYBL. Based on that, we generate a TOSCA description [5] for the M2M DaaS with elasticity requirements. This description is sent to Salsa, which determines deployment strategies and enriches the TOSCA with deployment information. Figure 5 presents a simple example of the enriched TOSCA with elasticity requirements as well as deployment information.

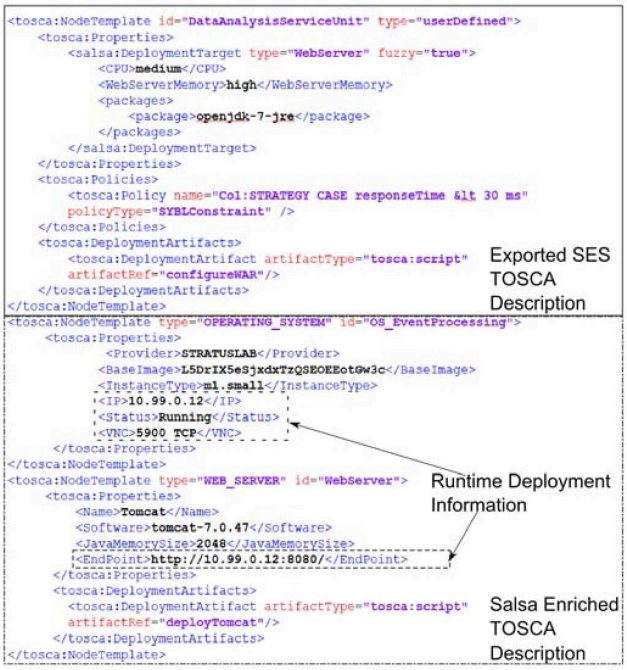

Fig. 5. Examples of SES structure, deployment and runtime description

Salsa deploys the M2M DaaS to multiple cloud environments. It also deploys other required units, if they have not been deployed, such as SYBL for controlling elasticity and MELA for monitoring elasticity. As the M2M DaaS is running,

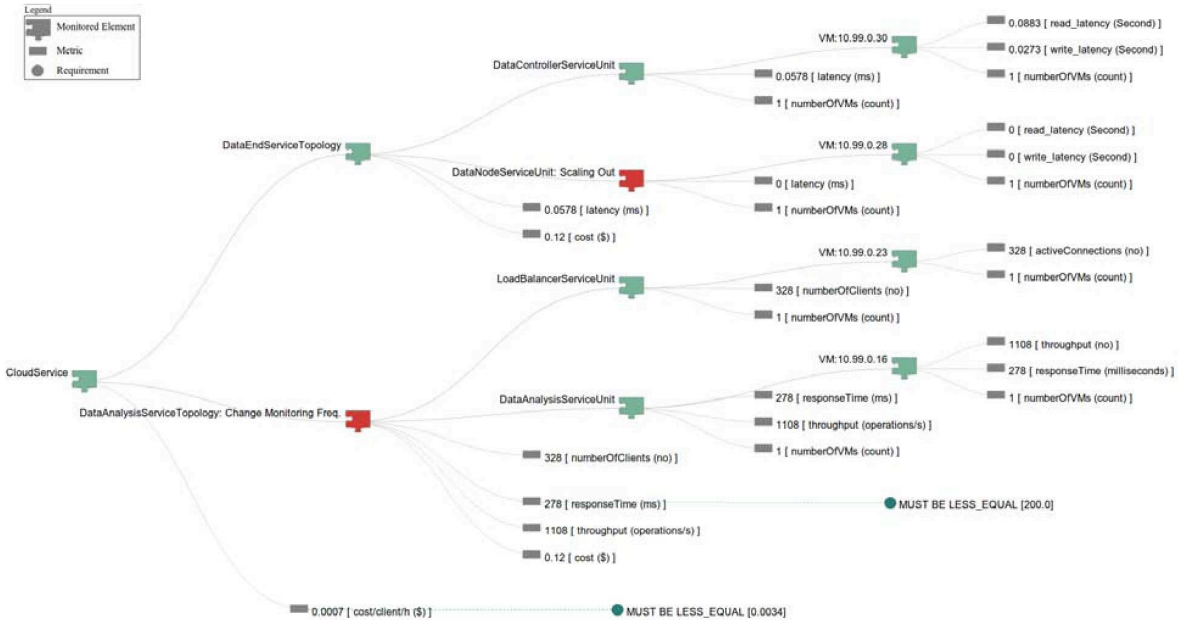

Fig. 6. Elasticity space monitoring integrated with control, deployment and testing events/metrics

the user can observe elasticity metrics, such as shown in Figure 6. Test cases can be defined and deployed to test running service units. The results of testing and elastic metrics from testing can also be shown using MELA.

5. Related Work

Several industrial and academic PaaS have been proposed, developed and provided, such as Appistry CloudIQ [6], Apprenda SaaSGrid [7], Boomi AtomSphere [8], Gigaspaces [9], Google App Engine [10], Microsoft Azure [11], Parabon Frontier [12]. On the one hand, we see many of them supporting programming cloud services based on well-known application models with little/without considering multi-dimensional elasticity nature. Mostly they support resource elasticity. On the other hand, we also observed PaaS, such as OpenTOSCA [13], Azure's Octopus Deploy [14] or ModaClouds [15], supporting resource elasticity. In contrast to existing ones, our PaaS is novel by focusing on programming, deployment, monitoring, control, and testing based on multi-dimensional elasticity in an integrated view.

6. Conclusions and Future Work

In this paper, we discuss issues that PaaS should support to meet elasticity requirements of (future) native software-defined elastic systems. We show that by conceptualizing elastic service units with explicit elasticity capabilities programmed via software-defined APIs, we can facilitate interwoven design, deployment, testing, execution and control activities for elastic cloud systems. We are currently carrying out our integration and further development of these building blocks in CoMoT.

Acknowledgment: The work mentioned in this paper is partially supported by the Pacific Control Cloud Computing Lab and by the European Commission in terms of the CELAR FP7 project (FP7-ICT-2011-8 #317790).

References

[1] S. Dustdar, Y. Guo, B. Satzger, and H. L. Truong, "Principles of elastic processes," *IEEE Internet Computing*, vol. 15, no. 5, pp. 66–71, 2011.
[2] G. Copil, D. Moldovan, H. L. Truong, and S. Dustdar, "Sybl: An extensible language for controlling elasticity in cloud applications," in *CCGRID*. IEEE Computer Society, 2013, pp. 112–119.
[3] D. Moldovan, G. Copil, H.-L. Truong, and S. Dustdar, "Mela: Monitoring and analyzing elasticity of cloud services," in *International Conference on Cloud Computing Technology and Science*, ser. CloudCom, 2013, p. to appear.
[4] A. Gambi, W. Hummer, H.-L. Truong, and S. Dustdar, "Testing elastic computing systems," *Internet Computing, IEEE*, vol. 17, no. 6, pp. 76–82, 2013.
[5] T. Binz, G. Breiter, F. Leymann, and T. Spatzier, "Portable cloud services using tosca," *IEEE Internet Computing*, vol. 16, no. 3, pp. 80–85, 2012.
[6] Appistry CloudIQ Platform, http://www.appistry.com/products.
[7] Apprenda SaaSGrid, http://apprenda.com/.
[8] Boomi AtomSphere, http://www.boomi.com/.
[9] GigaSpaces, http://www.gigaspaces.com/.
[10] Google App Engine, http://code.google.com/appengine/.
[11] Microsoft Azure Services Platform, http://www.microsoft.com/azure/default.mspx.
[12] Parabon Frontier, http://www.parabon.com/.
[13] A. Nowak, T. Binz, U. Breitenbcher, F. Haupt, O. Kopp, F. Leymann, and S. Wagner, "OpenTOSCA - A Runtime for TOSCA-based Cloud Applications," in *Service-Oriented Computing*, ser. Lecture Notes in Computer Science, S. Basu, C. Pautasso, L. Zhang, and X. Fu, Eds. Springer Berlin Heidelberg, 2013.
[14] "Azure's octopus deploy, http://octopusdeploy.com."
[15] E. D. Nitto, "Supporting the development and operation of multi-cloud applications: The modaclouds approach," in *International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, ser. SYNASC. IEEE, 2013.