Towards Bridging the Gap Between Scalability and Elasticity

Nicolas Ferry[1], Gunnar Brataas[2], Alessandro Rossini[1], Franck Chauvel[1] and Arnor Solberg[1]
[1]*Department of Networked Systems and Services, SINTEF, Oslo, Norway*
[2]*Department of Software Engineering, Safety and Security, SINTEF, Trondheim, Norway*

Keywords: Scalability, Elasticity, Cloud Computing, Multi-cloud, Self-adaptation, ScaleDL, CloudML.

Abstract: Scalability and elasticity are key capabilities to tackle the variable workload of a service. Cloud elasticity offers opportunities to manage dynamically the underlying resources of a service and improve its scalability. However, managing scalability of cloud-based systems may lead to a management overhead. Self-adaptive systems are a well-known approach to tame this complexity. In this position paper, we propose an approach for the continuous design and management of scalability in multi-cloud systems. Our approach is based on a three-layer architecture and relies on two existing frameworks, namely SCALEDL and CLOUDMF.

1 INTRODUCTION

Scalability and elasticity are key capabilities required to tackle the variable workload of a service. In particular, scalability is the ability of a service to sustain variable workload while fulfilling quality of service (QoS) requirements, possibly by consuming a variable amount of underlying resources. By contrast, elasticity is the ability of a service to rapidly provision and deprovision underlying resources on the fly. Note that both scalability and elasticity are required because one does not guarantee the other.

Cloud computing offers opportunities to improve the scalability of services by enabling the dynamic management of platform and infrastructure resources. Within the scope of the CloudScale[1], MODAClouds[2], and PaaSage[3] projects on cloud computing, specific work has been done to address long-term workload variations (*e.g.,* the workload increases linearly over a year) as well as short-term ones (*e.g.,* the workload has a sudden peak). The CloudScale's Scalability Description Language (SCALEDL) (Brataas et al., 2013) is a language for characterising scalability aspects of cloud-based systems in a provider-independent way. SCALEDL facilitates reasoning about long-term workload evolution in a proactive way. The MODAClouds' and PaaSage's Cloud Modelling Framework (CLOUDMF) (Ferry et al., 2013b; Ferry et al., 2013a) consists of the Cloud Modelling Language (CLOUDML) for specifying the provisioning and deployment of multi-cloud systems at design-time, as well as a models@run-time engine for enacting the provisioning, deployment, and adaptation of these systems at run-time. CLOUDMF facilitates handling short-term workload evolution in a reactive way.

Reasoning about long-term workload evolution proactively and handling short-term workload evolution reactively are typically treated as two independent activities. In order to improve management of workload evolution, these two activities must be combined. This position paper proposes an architecture inspired by self-adaptive systems for self-managing scalability of multi-cloud systems. This architecture consists of three layers: a layer for long-term adaptations that leverages upon SCALEDL, a layer for short-term adaptations that leverages upon CLOUDMF, and an intermediary layer for handling mid-term adaptations that bridges the gap between the other two layers. The proposed approach spurs to a new separation of concerns between mechanisms to handle short-, mid- and long-term scalability, enabling the continuous design and management of scalable cloud-based systems.

The remainder of the paper is organised as follows. Section 2 outlines a motivating example which is used throughout the paper. Section 3 introduces the three-layer architecture for self-managing scalability of multi-cloud systems. Section 4 and 6 outline SCALEDL and CLOUDMF as the foundation for the long-term and short-term adaptations, respectively.

[1]http://www.cloudscale-project.eu
[2]http://www.modaclouds.eu
[3]http://www.paasage.eu

Ferry N., Brataas G., Rossini A., Chauvel F. and Solberg A..
Towards Bridging the Gap Between Scalability and Elasticity.
DOI: 10.5220/0004975307460751
In *Proceedings of the 4th International Conference on Cloud Computing and Services Science* (MultiCloud-2014), pages 746-751
ISBN: 978-989-758-019-2

Section 5 presents the new, intermediary layer for handling mid-term adaptations that bridges the gap between SCALEDL and CLOUDMF. Section 7 compares the proposed approach with related works before Section 8 draws some conclusions.

2 MOTIVATING EXAMPLE

SENSAPP[4] is an open-source, service-oriented application for registering sensors, storing their data, and notifying clients when new data are pushed (Mosser et al., 2012).

SENSAPP is currently employed in the CITI-SENSE[5] EU project, which aims at providing a community-based environmental monitoring and information system. In particular, SENSAPP is adopted for collecting environmental monitoring data (air quality, noise level, *etc.*) in public spaces where the workload of SENSAPP may vary significantly throughout a year. In particular, we assume the following predicable variations at various time scales:

Daily The workload has regular peaks at mornings and evenings, when people commute daily.

Weekly The workload has regular peaks on Friday evenings, Sunday nights, and Monday mornings, when people commute for the weekend.

Yearly The workload has gradual increase and decrease before and after Summer and Christmas holiday seasons.

We also assume the following unpredictable variations:

Snow storm and cold wave The workload has an irregular peak due to a snow storm and cold wave, such as the one in Europe on 27 January 2012, which led to the disruption of European air and surface traffic for two weeks.

Volcanic eruption The workload has an irregular peak due to a volcanic eruption, such as the one of the Eyjafjallajkull volcano on 14 April 2010, which led to the stop of all European air traffic and increase of surface traffic for one week.

This scenario is going to be used as a running example throughout the paper in order to illustrate how the proposed approach handles these predictable and unpredictable variations.

[4]http://sensapp.org

[5]http://www.citi-sense.eu

3 ARCHITECTURE

The proposed architecture for self-managing scalability of cloud-based systems is based on the reference three-layer architecture for self-adaptive systems (Kramer and Magee, 2007). This architecture is organised as a three-layer stack, where each layer denotes a particular level of abstraction, complexity, and dynamic, *i.e.,* how fast the layer can react to a variation. Note that these layers retain a certain degree of independence so that each of them can pursue its own dynamic.

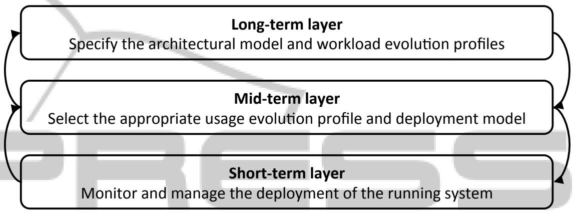
Long-term layer
Specify the architectural model and workload evolution profiles
Mid-term layer
Select the appropriate usage evolution profile and deployment model
Short-term layer
Monitor and manage the deployment of the running system

Figure 1: A three-layer architecture for self-managing scalability of multi-cloud systems.

Figure 1 outlines the proposed architecture. From the most abstract to the most concrete (*i.e.,* from the farthest to the closest to the running system), the three layers are described as follows:

Long-term layer. The designer specifies the architectural model of the system as well as usage evolution profiles within different time frames: day (*e.g.,* rush hour), week (*e.g.,* weekend), and year (*e.g.,* holiday season).

Mid-term layer. The mechanisms of this layer select the appropriate usage evolution profile and deployment model based on the current workload and context of the running system.

Short-term layer. The mechanisms of this layer monitor and manage the deployment of the running system.

For each layer, the considered time frames are up to the designer and may vary. However, these have to be organized hierarchically (*e.g.,* short-term: current, mid-term: week, long-term: year).

The independence between the layers enables the continuous evolution of the cloud-based system. The design process at the long-term layer can be done whilst the two other layers continue adapting the running system. This way, the design of the self-adaptive system does not need to be achieved beforehand, but can be done iteratively. Similarly, while the mid-term layer selects a new deployment model, the short-term layer can still manage minor workload evolution. This feature enhances the practicality of the approach as well as the design and management of the self-adaptive system.

All these layers are built upon model-driven engineering (MDE) techniques and methods. MDE is a branch of software engineering that aims at improving the productivity, quality, and cost-effectiveness of software development by leveraging upon models and model transformations. This approach, which is commonly summarised as "model once, generate anywhere", is particularly relevant when it comes to the provisioning and deployment of applications across multiple clouds, as well as their migration from one cloud to another.

4 LONG-TERM LAYER: ScaleDL

The long-term layer leverages upon SCALEDL, a family of four sub languages to characterise scalability of cloud-based systems (Brataas et al., 2013). In particular, this layer adopts SCALEDL USAGE EVOLUTION, which facilitates the specification of scalability requirements by characterising how the *workload* of a service changes over time. Note that these models are technology- and provider-agnostic so that they can be applied to multi-cloud systems.

As a first task, a designer must specify initial usage evolution profiles (Brataas et al., 2013) for all the operations provided as a service by the application. In the motivating example (see Section 2), the following initial usage evolution profiles of SENSAPP are specified: (*i*) *Browse*, where consumers are looking for some sensors registered in SENSAPP, (*ii*) *Monitor*, where consumers request data from sensors and (*iii*) *Push*, where sensors are pushing data into SENSAPP. These specifications include:

Work. The amount of data processed by the operation; *e.g.,* the work of the Push operation may be 100KB of data.

Load. The number of requests to the operation during a specified time interval; *e.g.,* the load of the Push operation may be an average of one request per second during regular work days, two requests per seconds during regular weekends, and four requests per seconds during snow storms and cold waves.

Quality metric and threshold. The measure of quality of the operation along with the corresponding threshold; *e.g.,* the quality of the Push operation is measured in terms of the average response time and the corresponding threshold is 100 ms.

The initial usage evolution profiles will then evolve over time. SCALEDL USAGE EVOLUTION supports three forms of usage evolution (Brataas et al., 2013): *stable* and *gradual* changes as well as *spikes*. In the motivating example (see Section 2), the usage evolution before and after holiday seasons is modelled using the gradual change usage evolution. Here, both the work and load will have a gradual change, but not necessarily with the same increase or decrease rate; *e.g.,* the amount of data pushed by each sensor increase faster than the number of sensor pushing data. Similarly, the usage evolution of a snow storm and cold wave or volcanic eruption is modelled with a spike. Here, the duration of the spike as well as its impact on work and load may vary.

In addition to the three forms of usage evolution above, it is possible to specify the expected usage evolution for a particular recurring time frame; *e.g.,* daily, weekly, or yearly. In the motivating example (see Section 2), the usage evolution profile of a working day specifies low load during the night, high load during rush hour, and normal load otherwise.

In addition to the usage evolution profiles, the long-term layer is used to specify architectural models. These models follow a service-oriented architecture where applications are defined as an orchestration of reusable services.

5 MID-TERM LAYER

The mid-term layer is responsible for bridging the gap between reasoning about long-term workload evolution proactively and handling short-term workload evolution reactively.

The usage evolution profiles and architectural models from the long-term layer are provided to the mid-term layer, where the responsible mechanisms select a usage evolution profile, predict the future workload, and evaluate if the quality metric and threshold are fulfilled. In particular, the mechanisms of this layer iterate the following process (see Figure 2):

1. The *Usage Evolution Manager* selects a usage evolution profile based on the current workload and context of the running system, *e.g.,* working day profile. This selection relies on event-condition-action rules, *e.g., on Monday (event), if this is a working day (condition), then select the working day profile (action)*. In case a designer wants to extend or refine the usage evolution profile for a shorter time frame, she can edit it before initiating the next task.

2. The *Workload Predictor* predicts what the workload will be at the time $t+1$ based on the workload at the current time t.
3. The *Workload Comparator* compares the current workload with the predicted one.
4. The *Architecture Selector* selects the most suitable architectural model from the the set of possible architectural models based on the current and predicted workloads.
5. The *Deployment Selector* selects the most suitable deployment model based on the corresponding architectural model.

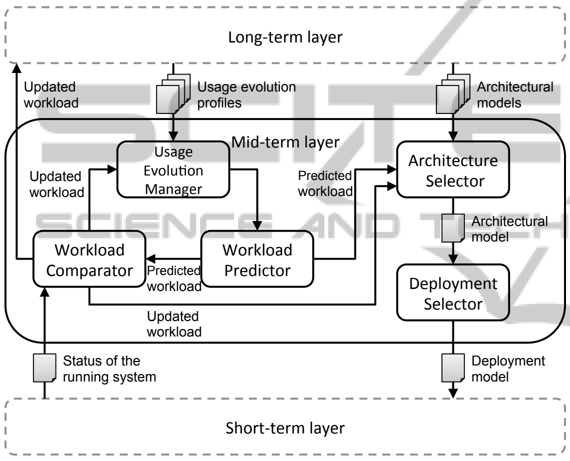

Figure 2: Architecture of the mid-term layer.

An iteration of this process can be triggered in two cases: *(i)* when the Workload Predictor anticipates that the workload is going to change; *(ii)* when the Workload Comparator detects that current workload does not correspond to the predicted one. In the latter case, a notification is raised to the long-term layer, which may alter the workload predictions.

In the motivating example (see Section 2), this process handles the following mid-term adaptations:

- In the event of snow storm and cold wave, the short-term layer reports the handling of the unpredictable workload to the mid-term layer, which analyses it and concludes it is an acceptable deviation from the prediction and hence does not report it to the long-term layer. This is because during winter one can anticipate that snow storms may occur and their average impact.
- In the event of volcanic eruption, the short-term layer reports the handling of the unpredictable workload to the mid-term layer, which analyses it and concludes it is an unacceptable deviation from the prediction and hence reports it to the long-term layer, which in turn alters the prediction. This is because, for instance, the predicted load in the usage evolution profile is two requests per second, while the current load is four. In addition, the load has been twice the predicted also during the two previous hours. Therefore, it may be reasonable to assume that the load will be twice the predicted also during the next hours, so the usage evolution profile for the next hours may be changed accordingly.

In case the duration of the workload is expected to be short, and considering the fact that adapting the deployment of a multi-cloud system may take some minutes, the deployment model may already be able to handle a range of variations of workload. For instance, this may lead to the inclusion of load-balancing mechanisms in the deployment model. These mechanisms are managed by the short-term layer.

6 SHORT-TERM LAYER: CloudMF

The short-term layer leverages upon CLOUDMF, a framework facilitating the management of multi-cloud systems. It consists of: (*i*) CLOUDML, a tool-supported, domain-specific language for specifying the provisioning and deployment of multi-cloud systems at design-time; (*ii*) a models@run-time engine for enacting the provisioning, deployment, and adaptation of these systems at run-time.

The deployment model from the mid-term layer is provided to the short-term layer, where the responsible mechanisms will adapt the running system.

CloudML

The deployments models are specified with CLOUDML. As mentioned, CLOUDML enables modelling the provisioning and deployment of multi-cloud systems. Note that CLOUDML is technology-agnostic, meaning that the multi-cloud systems can be designed and implemented based on arbitrary paradigms and technologies. In particular, CLOUDML allows expressing the following concepts:

Cloud Represents a collection of virtual machines on a particular cloud provider.

Virtual machine Represents a reusable type of virtual machine.

Application component Represents a reusable type of application component to be deployed on a virtual machine, or an external service. This element can be parameterised by elasticity requirements (*e.g.,* SENSAPP can be instantiated from one to four times).

Port Represents a required or provided interface to a feature of an application component.

Relationship Represents a communication between *ports* of two application components, or the containment of an application component by another.

The interested reader may consult (Rossini et al., 2013) for a more detailed description of CLOUDML.

In the motivating example (see Section 2), SENSAPP is deployed on three different cloud providers. Figure 3 shows a snapshot of the deployment model of SENSAPP to be used when the workload is low: the MongoDB database and the *Dispatcher* of SENSAPP are deployed behind the corresponding load balancers (depicted by LB in the figure) on a private OpenStack cloud, while the *Notifiers* (*i.e.,* the services that notify consumers about new sensor data) are deployed on the Amazon and Flexiant public clouds.

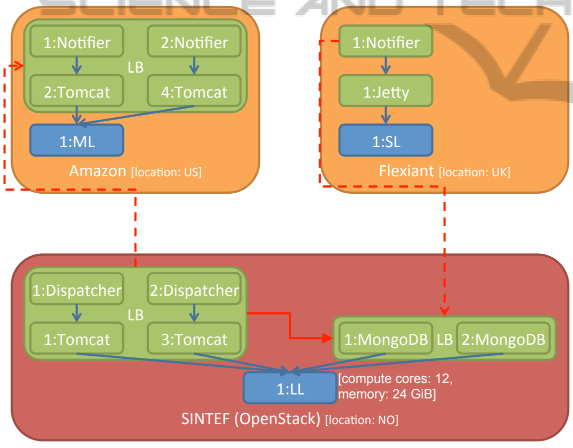

Figure 3: Snapshot of a SENSAPP deployment.

Models@run-time engine

The deployment models are enacted by the CLOUDMF models@run-time engine. Models@run-time is an architectural pattern for dynamic adaptive systems that proposes to leverage models during their execution (Morin et al., 2009). In particular, models@run-time provides an abstract representation of the underlying running system, which facilitates reasoning, simulation, and enactment of adaptation actions. A change in the running system is automatically reflected in the model of the current system. Similarly, a modification of this model can be enacted on the running system on demand.

Figure 4, inspired by (Morin et al., 2009), shows a classical architecture to realise models@run-time. The current model can be consumed by a reasoning system (*e.g.,* the mid-term layer) that produces a target model. The adaptation is then enacted by the adaptation engine and the target model becomes the current model.

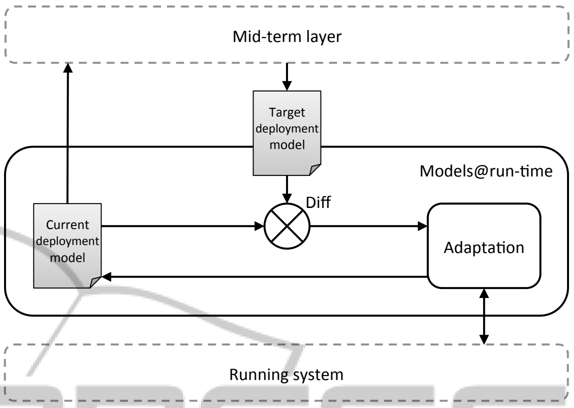

Figure 4: Models@run-time engine.

Adaptations of the model can result in: (*i*) modification of the provisioning and deployment topology or (*ii*) modification of the status of each element of the system. As part of the possible adaptation, a classical solution to handle elasticity requirements is to use load balancers. The models@run-time engine is then responsible for deploying a load balancer and configuring it to manage communications to a group of application components on a defined port.

The monitoring of the running system consists in providing the status of the current deployment together with the quality metrics (see Section 4) so that the current workload and context can be matched against the ones predicted in the long-term layer.

7 RELATED WORK

Elasticity is one of the key features to make cloud-based systems scalable and has been extensively investigated over the past few years. Elastic infrastructure are supported either by provider-specific features such as the Amazon Elastic Load Balancing, or by network level's components such as NGinx. Map/reduce platforms form another level of elasticity provided also as provider-specific services. While these techniques are now mature, they require recurrent management activities through the lifespan of a cloud-based system, as the workload evolves.

The management of cloud-based systems, including their scalability, can be facilitated by various tools. Industrial frameworks such as Cloudify[6], pup-

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

pet[7], or Chef[8], as well as research projects such as Reservoir[9] or ARTIST[10], provide capabilities for the automatic deployment and management of cloud systems, including load-balancing mechanisms. However, such frameworks do no support the long-term adaptation of elasticity policies, as does the proposed long-term layer.

Scalability management can also be addressed with performance analysis techniques. Palladio and SimuLizar (Becker et al., 2013), for instance, capture both the system and the scalability logics and produce performance predictions, which can be matched against performance requirements, leading to gradual improvements. However, this is a pure design-time activity which does not leverage run-time information, as does the model@run-time engine in the proposed short-term layer.

The Topology and Orchestration Specification for Cloud Applications (TOSCA) (Palma and Spatzier, 2013) standard is a related specification developed by the OASIS. TOSCA provides a language for specifying the components comprising the topology of cloud applications along with the processes for their orchestration. However, this standard currently lacks a models@run-time representation that enables the continuous evolution of multi-cloud systems.

8 CONCLUSION AND FUTURE WORK

In this position paper, we outlined an approach to self-managing scalability of multi-cloud systems. The proposed solution combines ScaleDL and CloudML into a three-layer architecture. In the long-term layer, the designer specifies the architectural model of the system as well as ScaleDL usage evolution profiles. In the mid-term layer, the responsible mechanisms select the appropriate usage evolution profile and deployment model based on the current workload and context of the running system. Finally, in the short-term layer, CloudMF monitors and manages the deployment of the running system.

The realisation of this three-layer architecture is an ongoing joint work between the CloudScale, MODAClouds, and PaaSage projects. Future research directions include: finalising the implementation and validation of the proposed approach, and designing a mechanism for collecting past load variations to improve the accuracy of load predictions by means of statistical analysis.

[7]https://puppetlabs.com

[8]http://www.opscode.com/chef

[9]http://www.reservoir-fp7.eu/

[10]http://www.artist-project.eu/

ACKNOWLEDGEMENT

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreements number: 318484 (MODAClouds), 317715 (PaaSage), and 317704 (CloudScale).

REFERENCES

Becker, M., Becker, S., and Meyer, J. (2013). SimuLizar: Design-Time Modeling and Performance Analysis of Self-Adaptive Systems. In *Software Engineering 2013*, volume 213 of *LNI*, pages 71–84. GI.

Brataas, G., Becker, S., Huljenic, D., Kopčak, G., Lehrig, S., Stav, E., and Walderhaug, S. (2013). D1.1 – Design support, initial version. CloudScale deliverable.

Ferry, N., Chauvel, F., Rossini, A., Morin, B., and Solberg, A. (2013a). Managing multi-cloud systems with CloudMF. In *NordiCloud 2013: 2nd Nordic Symposium on Cloud Computing and Internet Technologies*, pages 38–45. ACM.

Ferry, N., Rossini, A., Chauvel, F., Morin, B., and Solberg, A. (2013b). Towards model-driven provisioning, deployment, monitoring, and adaptation of multi-cloud systems. In *IEEE CLOUD 2013*, pages 887–894. IEEE Computer Society.

Kramer, J. and Magee, J. (2007). Self-Managed Systems: an Architectural Challenge. In *FOSE 2007*, pages 259–268.

Morin, B., Barais, O., Jézéquel, J.-M., Fleurey, F., and Solberg, A. (2009). Models@Run.time to Support Dynamic Adaptation. *IEEE Computer*, 42(10):44–51.

Mosser, S., Fleurey, F., Morin, B., Chauvel, F., Solberg, A., and Goutier, I. (2012). SENSAPP as a Reference Platform to Support Cloud Experiments: From the Internet of Things to the Internet of Services. In *SYNASC 2012*, pages 400–406. IEEE Computer Society.

Palma, D. and Spatzier, T. (2013). Topology and Orchestration Specification for Cloud Applications (TOSCA). Technical report, OASIS.

Rossini, A., Solberg, A., Romero, D., Domaschka, J., Magoutis, K., Schubert, L., Ferry, N., and Kirkham, T. (2013). D2.1.1 – CloudML Guide and Assesment Report. PaaSage deliverable.