A Model-Based Scalability Optimization Methodology for Microservices on the Cloud *

Einar Broch Johnsen, Jacopo Mauro, and Ingrid Chieh Yu

Department of Informatics, University of Oslo, Oslo, Norway
{einarj,jacopom,ingridcy}@ifi.uio.no

Cloud computing is changing the traditional business model for ICT solutions by offering on-demand delivery of infrastructure and applications over the Internet with pay-as-you-go pricing. To capitalize on these solutions, we need to develop scalable, adaptable, modular, and quickly accessible cloud-based applications. Microservice architectures provide a way to develop an application as a collection of fine-grained services running as independent processes which typically communicate asynchronously. By means of container technology, distributed applications can then be constructed from independently deployable services taking advantage of the properties of the microservice architecture (e.g., flexibility, maintainability, reusability, compositionality, and scalability) as well as the elasticity of cloud infrastructure by adding or removing virtual machine instances to respond to workload changes.

Deciding on the proper thresholds for scaling for a complex architecture is, however, a challenging task: many possible settings exist which may have big consequences in terms of system performance and cost. In previous work [3], we started to explore how a system can be reconfigured by developing JRO (Jolie Redeployment Optimiser), a tool for the automatic and optimized deployment of microservices. The tool uses the configuration optimizer Zephyrus [1] to automatically generate a target configuration that the system needs to reach. Unfortunately, since cloud instances may have random decreases in performance and take minutes to be deployed, knowing the final configuration is not enough: metric violations may happen during the steps performed to reach the target configuration, or even later in case of failures. To overcome this limitation, to decide better scaling strategies we propose to combine the strengths of optimization and modeling. While optimization techniques can be used directly on real systems, working with a model provides a more abstract and succinct representation of the composition and scaling problem which has several advantages: it allows (1) to explore the search space of different possible parameter settings faster (running a simulation is quicker than running the full system) and in a cheaper way (using cloud instances is much more expensive in terms of money and electricity) and (2) to explore possible but not frequent events such as failures or instance degradation.

The Actor paradigm provides an interesting model for microservices, in which asynchronous message passing between actors spawns concurrent activities without the need for explicit threads, leading to an inherent compositionality. The ABS [5] modeling language combines actors with object-oriented structuring concepts and with cooperative concurrency, which allows complex synchronization between concurrent activities to be expressed. Real-Time ABS, a recent extension of ABS, supports the explicit modeling of deployment decisions for timed, resource-restricted models [6]. Real-Time ABS has been used successfully to model services deployed on the cloud [2].

We have started to explore how the interplay of modeling in ABS and optimization can be exploited to find better (and possibly optimal) scaling strategies for microservices deployed on

*Supported by the EU projects H2020-644298 *HyVar: Scalable Hybrid Variability for Distributed Evolving Software Systems* (http://www.hyvar-project.eu).

the cloud, taking into account the unreliability of resources, complex service level agreement metrics, and the changing behavior of incoming requests. Our experience from an initial case study suggests a methodology to optimize the deployment of complex microservice systems, actively using executable models. Starting with an existing microservice architecture developed for dispatching car software updates, we create a simple executable model which exposes scaling decisions as configurable parameters (e.g., threshold values that trigger the scaling up or down of a microservice, instances to add or remove when a scaling action is performed, cooling-down time before another scaling action can be performed). Based on this model, we search for good scaling settings by using an automatic parameter configurator (e.g., [4]) that relies on machine learning techniques to explore possible configurations in a smarter and more systematic way in order to come up with good parameter settings. In this talk, we will present this work and report on our experiences from the case study.

References

[1] E. Ábrahám, F. Corzilius, E. B. Johnsen, G. Kremer, and J. Mauro. Zephyrus2: On the fly deployment optimization using SMT and CP technologies. In *SETTA*, 2016.

[2] E. Albert, F. S. de Boer, R. Hähnle, E. B. Johnsen, R. Schlatte, S. L. Tapia Tarifa, and P. Y. H. Wong. Formal modeling and analysis of resource management for cloud architectures: An industrial case study using Real-Time ABS. *Journal of Service-Oriented Computing and Applications*, (4), 2014.

[3] M. Gabbrielli, S. Giallorenzo, C. Guidi, J. Mauro, and F. Montesi. Self-reconfiguring microservices. In *Theory and Practice of Formal Methods - Essays Dedicated to Frank de Boer on the Occasion of His 60th Birthday*. Springer, 2016.

[4] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential Model-Based Optimization for General Algorithm Configuration. In *LION*. Springer, 2011.

[5] E. B. Johnsen, R. Hähnle, J. Schäfer, R. Schlatte, and M. Steffen. ABS: A core language for abstract behavioral specification. In B. Aichernig, F. S. de Boer, and M. M. Bonsangue, editors, *Proc. 9th International Symposium on Formal Methods for Components and Objects (FMCO 2010)*. Springer, 2011.

[6] E. B. Johnsen, R. Schlatte, and S. L. Tapia Tarifa. Integrating deployment architectures and resource consumption in timed object-oriented models. *Journal of Logical and Algebraic Methods in Programming*, (1), 2015.