The CloudScale Method for Software Scalability, Elasticity, and Efficiency Engineering: a Tutorial*

Sebastian Lehrig Steffen Becker
{sebastian.lehrig|steffen.becker}@informatik.tu-chemnitz.de
Software Engineering Chair
Chemnitz University of Technology, Chemnitz, Germany

ABSTRACT

In cloud computing, software engineers design systems for virtually unlimited resources that cloud providers account on a pay-per-use basis. Elasticity management systems provision these resource autonomously to deal with changing workloads. Such workloads call for new objective metrics allowing engineers to quantify quality properties like scalability, elasticity, and efficiency. However, software engineers currently lack engineering methods that aid them in engineering their software regarding such properties.

Therefore, the CloudScale project developed tools for such engineering tasks. These tools cover reverse engineering of architectural models from source code, editors for manual design/adaption of such models, as well as tools for the analysis of modeled and operating software regarding scalability, elasticity, and efficiency. All tools are interconnected via ScaleDL, a common architectural language, and the CloudScale Method that leads through the engineering process. In this tutorial, we execute our method step-by-step such that every tool and ScaleDL are briefly introduced.

Categories and Subject Descriptors

D.2.8 [**Software Engineering**]: Metrics—*Scalability, Elasticity, Efficiency*; D.2.9 [**Software Engineering**]: Management—*Software quality assurance (SQA)*; D.2.11 [**Software Engineering**]: Software Architectures—*Architectural analysis*

Keywords

Tutorial; CloudScale; Cloud Computing; Software; Analysis; Scalability; Elasticity; Efficiency; Metrics; Method; Engineering

*The research leading to these results has received funding from the European Seventh Framework Programme (FP7/2007-2013) under grant no 317704 (CloudScale).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
ICPE'15, Jan. 31–Feb. 4, 2015, Austin, Texas, USA.
Copyright © 2015 ACM 978-1-4503-3248-4/15/01 ...$15.00.
http://dx.doi.org/10.1145/2668930.2688818.

1. MOTIVATION

In cloud computing, software engineers develop applications on top of compute environments being offered by cloud providers. For these applications, the amount of offered resources is virtually unlimited while elasticity management systems provision resources autonomously to deal with changing workloads. Furthermore, providers bill provisioned resources on a per-use basis [1]. As a consequence of these characteristics, engineers want their applications to use as few resources as possible in order to save money while still maintaining the quality requirements of the system. Quality properties that focus directly on these aspects are scalability, elasticity, and efficiency [2].

These quality properties need to be quantified for software engineering by means of suitable metrics. For instance, cloud consumers and cloud providers need to negotiate service level objectives (SLOs), i.e., metrics and associated thresholds [5]. Such SLOs have to consider characteristics like changing workloads ("how fast can an application adapt to a higher workload?") and pay-per-use pricing ("how expensive is serving an additional consumer?"). However, no established engineering methods and tools supporting metrics for the mentioned quality properties exist. Current methods assume knowledge of implementation details as they focus on the application at run-time [6].

In literature, methods and tools with support for classical performance-oriented metrics [3] like response time and throughput are insufficient for situations relevant for cloud computing applications. First, they do not take changing workloads into account, e.g., metrics to describe reaction times to system adaptations are missing. Second, the degree to which systems match resource demands to changing workloads cannot be quantified. More recent work [6] proposes initial metrics for such characteristics that assume knowledge of implementation details like resource handling. Therefore, these metrics are inapplicable when such knowledge is unavailable, e.g., in early software engineering phases such as the software design phase. Accordingly, no existing method/tool has an appropriate support for software engineers that want to analyze scalability, elasticity, and efficiency properties at the design time of software applications.

To cope with this lack, we developed tools for such engineering tasks in the context of the CloudScale project [4]. These tools cover reverse engineering of architectural models from source code, editors for manual design/adaption of such models, as well as tools for the analysis of modeled and operating software regarding scalability, elasticity, and efficiency. We derived metrics for such properties in our pre-

vious work [2]. All tools are interconnected via ScaleDL, a common architectural language, and the CloudScale Method that leads through the engineering process.

The contribution of this tutorial paper is an example-based, step-by-step execution of our method such that every tool and ScaleDL are briefly introduced. Our method and tools eventually help software engineers in engineering cloud computing applications. Most interesting, we go beyond classical performance metrics like response time and highlight challenges in cloud computing settings where engineers have to plan for changing workloads and dynamic resource allocation.

This paper is organized as follows. In Sec. 2, we introduce our running example. Afterwards, we use this example to walkthrough our CloudScale Method in Sec. 3. Sec. 4 concludes the paper and gives an outlook on future work.

2. EXAMPLE SCENARIO

A software engineer of a company offers a book store as a Software-as-a-Service (SaaS) solution. The shop has been used for 15 years. An increase in load is now expected as a consequence of a new business strategy of selling novel cloud scalability books. The manager wants the engineer to ensure that the system – after modernization – can sustain this increased load.

Based on the general business strategy of the company, the engineer suggests the manager a modern cloud computing solution. However, the manager has heard of others that were disappointed by migrating to the cloud. The manager is concerned about short term issues such as unacceptable response times. In addition, he wants to know what the new system's operation will cost in the long term (considering the expected increase in users).

The engineer has heard that CloudScale [4] has some useful tools to analyze systems regarding such issues. Therefore, he plans to apply these tools and to provide the manager with detailed analysis results, allowing the manager to get rid of unpleasant surprises.

The engineer starts with the following. There is already an existing book store implementation which is implemented in the classical three layer architectural style (we show a screenshot of the client UI). This implementation has currently certain service level objectives (SLOs; e.g., 2 seconds response time limit). He plans to move this implementation to Amazon EC2 as one of the popular cloud computing infrastructures. However, he does not know whether features like autoscaling will eventually provide the needed user capacity (because he does not know whether his SaaS layer scales). As he also has no experience in building cloud computing systems, he is also unable to tell without either implementing the system and doing costly tests or without applying an engineering method like CloudScale's.

In the next section, we examine each step the engineer conducted to achieve his goals in detail. After the execution of these steps, the engineer is able to build a guaranteed scalable, elastic, and efficient system while answering all other questions of the manager. Source code and models of the book store and for all steps are available at our web page[1]. (Note: Our so-called "Cloud Book Store" is based on a legacy implementation of the TPC-W benchmark.)

[1] `http://www.cloudscale-project.eu/results/showcase/`; Last accessed at 2014/11/29

3. USING THE CLOUDSCALE METHOD

The engineer learns about the CloudScale method[2]. He also learns that this method is configurable to support multiple use cases. His use case is a modernization task. For this task, the following sequence of activities is recommended:

1. extract a model of the existing application,
2. refine the extracted model with resource demands,
3. analyze the model,
4. spot HowNotTos & resolve with HowTos,
5. reanalyze, and
6. implement, test, and operate when OK.

The following subsections now lead the engineer through applying these steps; potentially guided by a dedicated CloudScale tool. All of these tools are included within the CloudScale Environment[3].

3.1 Extractor: Extract Model from Code

The Extractor is a tool for reverse-engineering (partial) ScaleDL models from Java source code. ScaleDL allows to specify inter-connected components and their behavior, hardware resources, user behavior, and special annotations needed for scalability, elasticity, and efficiency analyses. Extractor supports the first part of ScaleDL (inter-connected components and their behavior).

Based on its clustering algorithm, the Extractor summarizes Java classes in such components and their interfaces (ScaleDL Repository model). For example, the book store source code involves ∼50 classes that Extractor summarizes to ∼20 software components. Extractor particularly links such components via connectors (ScaleDL System model). Thanks to such models, engineers can get a good overview of existing software, even if documentation is unavailable.

The engineer follows our screencast for extraction[4]. In our experience, such extractions take only a few minutes, even for larger systems (> 1 million LOC). For extracting ScaleDL models that provide a good overview, engineers commonly have to try several extractions with varying parameters for the clustering algorithm. These parameters are explained in our screen cast as well.

3.2 ScaleDL Editors: Refine & Calibrate

Because the Extractor only provides a partly specified ScaleDL model, the engineer has to refine the previously extracted model.

First, he needs to annotate resource demands using our ScaleDL editors. Therefore, the engineer takes an example behavior specification of a component, measures the needed resource demands for that behavior, and puts them into the model. Our Analyzer series of screen casts explains resource demand measurement in detail (see same page as the Extractor screen cast).

[2] `http://www.cloudscale-project.eu/results/method/`; Last accessed at 2014/11/29

[3] `http://www.cloudscale-project.eu/results/environment/`; Last accessed at 2014/11/29

[4] `http://www.cloudscale-project.eu/results/screencasts/`; Last accessed at 2014/11/29

The engineer repeats resource demand measurement for all quality-relevant component behaviors. In our experience, this process takes a maximum of 2 weeks for half a million lines of code; for the complete cloud book store it took 1 week for an inexperienced engineer.

Second, the engineer adds usage scenarios to the model. For getting the information he needs for these models, he asks the manager for the expected load evolution. He models this evolution using the ScaleDL Usage Evolution editor. The editor comes with a self-explanatory wizard.

3.3 Analyzer: Analyze

Once finished with ScaleDL's Usage Evolution model, the engineer runs CloudScale's analysis tool – the Analyzer. On our screen casts page, we demonstrate how and which results can be obtained using the Analyzer. For example, we show predicted response times and connect them to previously stated SLOs.

In our running example, the engineer may observe scalability issues: the system capacity (in terms of the maximum number of users the system can cope with) is insufficient. He may also observe too many SLO violations in Analyzer's results.

3.4 Spotter: Spot HowNotTos & Resolve with HowTos

Because the engineer observed too many SLO violations, he wants to investigate the root causes for these using CloudScale's Spotter. Spotter detects scalability, elasticity, and efficiency anti-patterns utilizing information from the source code, extracted models, and/or the system in operation. Such anti-patterns are provided in CloudScale's catalogue of HowNotTos[5]. Our screen casts about CloudScale's Spotter exemplify the detection of the one lane bridge anti-pattern. This anti-pattern is also detected by our engineer.

Therefore, the engineer looks up the Simplified SPOSAD HowTo (in the CloudScale catalogue for best practices[6]). Based on this HowTo, he resolve the found issue resulting in a new implementation and model.

3.5 Analyzer: Reanalyze

The architect reanalyzes the new model created based on the Simplified SPOSAD HowTo. He particularly investigates the differences to the previous analysis.

The analysis results indicate a success, which means that the engineer now can also pay attention to other metrics generated, e.g., operational costs (checking this long-term concern was asked for by the manager). In our recent works, we describe supported metrics [2] and their integration into CloudScale's Analyzer [7].

3.6 Implement, Test & Operate

After the engineer achieved satisfying analysis results, he implements the planned system according to the modified model. In our scenario, the implementation is fine, testing does not show any issues any more, and operation really costs what has been predicted. The manager and the engineer are happy now.

[5] `http://cloudscale.xlab.si/wiki/index.php/HowNotTos:_Anti-Patterns`; Last accessed at 2014/11/29

[6] `http://cloudscale.xlab.si/wiki/index.php/HowTos`; Last accessed at 2014/11/29

4. CONCLUSIONS

In this tutorial paper, we apply the CloudScale method step-by-step on a running example. For each step, we point to relevant resources allowing to reproduce our descriptions. In particular, the running example is available at our web page[7].

This tutorial helps software engineers to learn our method and to get familiar to our tools and languages for engineering cloud computing applications. Our engineering method goes beyond classical performance metrics like response time and supports challenges in cloud computing settings where engineers have to plan for changing workloads and dynamic resource allocation.

In our future work, we will polish our tools and reduce some remaining manual effort in using them. Where feasible, we will extend our screen casts such that every step will finally be exemplified in detail. Regarding HowTos and HowNotTos, we plan to extend our catalogues as we now have the infrastructure in place. We also recently moved core parts of our tools to GitHub to make it easier for others to contribute[8]. The CloudScale project ends September 2015 – until then, we plan to finalize these tasks.

5. REFERENCES

[1] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, Apr. 2010.

[2] M. Becker, S. Lehrig, and S. Becker. Systematically deriving quality metrics for cloud computing systems. In *Proceedings of the 6th ACM/SPEC International Conference on Performance Engineering*, ICPE '15, New York, NY, USA, 2015. ACM. Accepted for publication.

[3] G. Bolch, S. Greiner, K. S. Trivedi, and H. de Meer. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation With Computer Science Applications*. 1998.

[4] G. Brataas, E. Stav, S. Lehrig, S. Becker, G. Kopcak, and D. Huljenic. CloudScale: Scalability Management for Cloud Systems. In *4th Int. Conf. on Performance Engineering*. ACM, Apr. 2013.

[5] T. Erl, Z. Mahmood, and R. Puttini. *Cloud Computing: Concepts, Technology & Architecture*. Prentice Hall, 2013.

[6] N. R. Herbst, S. Kounev, and R. Reussner. Elasticity: What it is, and What it is Not. In *Proceedings of the 10th International Conference on Autonomic Computing (ICAC 2013), San Jose, CA, June 24–28*, 2013.

[7] S. Lehrig and M. Becker. Approaching the Cloud: Using Palladio for Scalability, Elasticity, and Efficiency Analyses. Technical Report 2014/05, Proceedings of the Symposium on Software Performance 2014, University of Stuttgart, Faculty of Computer Science, Electrical Engineering, and Information Technology, Nov. 2014.

[7] `http://www.cloudscale-project.eu/results/showcase/`; Last accessed at 2014/11/29

[8] `https://github.com/CloudScale-Project`; Last accessed at 2014/11/29