Simulation-based resilience prediction of microservice architectures

Samuel Beck, Johannes Günthör, Christoph Zorn

University of Stuttgart
Reliable Software Systems (RSS)
70569 Stuttgart, Germany

Abstract. Current software simulators are tailored towards one specific purpose of conservative software simulation. Given the success of these tools it would be useful to run these tools on microservice architectures. This paper will focus on the development of a simulator that can be used for microservice architectures.

1 Introduction

Was rein muss: Was wir wollen (unser ziel), ...

2 Research

2.1 Tools in Comparison

Spigo During our research on existing tools for microservice simulation we discovered a tool called spigo. It was written by Adrian Cockcroft a Amazon Web Services employee in the programming language go. Therefore the name spigo comes from Simulate Protocol Interactions in Go.
On the first view the tool looked very promising. Spigo contains a fairly simple JSON input and the structure of the parameters is intuitive. Each microservice architecture consists of multiple microservices. Each microservice has a name, a package inheritance a counter of the instances and dependencies to other microservices. The reason we like this tool is because it can simulate the occurrence of an error. Spigo uses the error monkeys from the simian army. But here lies already on of the biggest disadvantages. One can only simulate the failure of a single microservice during the execution. Another point against spigo is the weak documentation of the actual implementation as well as minor bugs that remain from 12 months ago.
In conclusion it is quite sure that spigo is not an option for us to use for an extension module. The restrictive and static code design makes it hard to follow the workflow and append another module.

Palladio Palladio is a software component model for business information systems to enable model-driven predictions on throughput, response time and resource utilization. [3] Our motivation to get information on the PCM was primarily to get knowledge from the inheritance of SimuLizar. A tool extended from the PCM and described later in this article. Component models provide many advanteages over object-oriented development approaches i.e. higher usability, quality and better test potential. [3]

Simulizar Simulizar is an extension of the Palladio Component Model. This tool was especially developed for systems that change at runtime i.e. cloudcomputing and virtualized infrastructure environments. Such dynamic system adapt to the environment in order to meet the quality-of-service requirements. In the most cases software systems rely on static and fixed resource management so only steady states can be predicted.[1] Simulizar uses the so called MAPE-K feedback loop (Monitor, Analyze, Plan, Execute, Knowledge) which is used to react to changes that are done during runtime.
Palladio and especially Simulizer seem to be to complex and big to get an understanding of the tool in depth. The sheer amount of code and resources that is used for collecting data and making prediction would be just an overhead on the problems that we currently face.

GreenCloud With this tool we tried a different approach. Greencloud was designed to calculate the energy consumption of datacenters[2, P.1]. Knowing that this was kind of a long shot we had the idea to take a simulator that simulates distributed objects and map microservice abilities and requirements to these objects. Since the simulator was written to overlook datacenters and their components, a mapping would mean that the entire microservice system would be mapped to a datacenter in the current GreenCloud simulator. Instances of microservices would compare to a server[2, P.2] that gathers metrics. These metrics are currently power consumption, CPU- utilization and workload [2, P.3] and should be changed or replaced to throughput,workload and whatever metrics we require from a microservice. GreenCloud also includes connections[2, P.3] between servers. The connection could possibly used to be a mapping of microservice connections, that we can interfere to implement chaos monkeys.
Sounding good in theory but taking a closer look at the tool some problems lead to a problems. The focus of GreenCloud is obviously the simulation of power consumption. The previous mentioned connection between servers is not actually modeled but just taken into consideration regarding the power consumption of switches that connect servers together. Additionally there is no sign of the possibility to scale Instances (servers) which is a major part in a microservice system. Lastly workloads are only specified as the computing power they require (e.g. 3 Million Instructions per Second)[2, P.3]. Communication or a differentiation between machines(DB, Workstation, ...) takes additional resources and is currently not taken into account. This would make mapping to a microservice architecture incredibly hard. In hindsight it seems to be not very likely that using

GreenCloud would be a good idea. The differences between provided capabilities by GreenCloud and required capabilities by the new tool is just to large.

CloudSim Developed by the CLOUDS Laboratory of the University of Melbourne, CloudSim is a framework for modeling and simulation of cloud computing infrastructures and services. It allows modeling and simulation of large scale cloud computing data centers aswell as modeling and simulation of application containers and virtualized server hosts.[**?**] The platform is written in Java and extensible, in fact various extensions for the framework exist. Its architecture consists of multiple layers. The fundamental layer provides management of applications, hosts of virtual machines, etc. while the top layer represents the basic entities for hosts and enables the generation of requests in a variation of approaches, configurations and cloud scenarios.[**?**]

3 Simulatorspecification

Expected Input properties We want our Simulator to execute the following tasks. In order to simulate a system we need to know how the actual system is build. Therefore we will get information form an input file. This file will contain information about the infrastructure of the system. Each microservice(MS) that is part of the system has the following properties. It has a unique name, the number of instances that are used in the simulation, dependencies on other MS's, throughput and anti-patterns that it implements.
During runtime we want to be able to modify the system by deleting some instances of microservices. We will achieve this by implementing a chaosmonkey which is inspired by netflix´s toolbox of cloud tools. There can be any number of chaosmonkey´s that need the following parameters to intercept the system. It has a Ms as a target, the number of instances it will delete and the time period in which the error will occur. Finally the input has a number of message objects that travel through the system. We expect the input to look like Figure 1(!!!Instert figzure 1!!!)

Expected system runtime behavior The basic idea of our system is that is runs on a clock. For each time period every MS will be able to send, receive and work on message objects. If a number of MS instances will go down it is expected to have larger message queues in front of a MS. As a result the system will probably slow down. To guarantee fail-safety the user will have the option to include anti-patterns in the input file. This will make a MS resilient against high workloads and failures of dependent MS. During simulation time we already gather information about performance in each timeframe to generate metrics. We also collect system failures in form of lost message objects if certain MS aren?t available anymore.

Expected output metrics We collect the following metrics for all MS´s. The workload, queue utilisation (size) and potential system failures due to missing instances of services. As a result there should be highlighting of eventual problematic data and an analyzation of bottlenecks and performance problems. This information will then be written into a user-friendly output file to deliver a big picture of the system.

(!!TODO!! INSERT FIGURE2 SYSTEM)

---

was muss das tool können?
was ist der input
was ist der output
grafik wie system ablaufen soll
welche metriken müssen gesammelt werden

Execution Validate input
Executable system
With reliability patterns - circuit breaker
Without reliability patterns
Failure
Output

Input: JSON/XML/YAML
Graph?

Types of failures:
Failure of one service
Failure of multiple services

Failure modes:
Static
Dynamic
Event driven/Triggered

Types of reliability patterns:
Circuit breaker
Client-side load balancing

Metrics How does the failure propagate
Did the fail-safe measures (circuit breaker, etc.) work
Systemrun was successful

TODO - RESEARCH: Occurring failure types Reliability patterns Metrics Input structure and file format

Available simulators for cloud systems: Spigo Palladio/Simulizar cloudsim

4 Simulatordocumentation

- struktur des programms
- probleme in der implementierung
-

5 Conclusions

These are my conclusions.

References

[1] M. Becker, S. Becker, and J. Meyer. Simulizar: Design-time modeling and performance analysis of self-adaptive systems, 2013. URL `http://www.performance-symposium.org/fileadmin/user_upload/palladio-conference/2014/papers/paper12.pdf`. Last visited June 23, 2017.

[2] D. Kliazovich, P. Bouvry, Y. Audzevich, and S. U. Khan. Greencloud: A packet-level simulator of energy-aware cloud computing data centers. In *2010 IEEE Global Telecommunications Conference GLOBECOM 2010*, pages 1–5, Dec 2010. doi: 10.1109/GLOCOM.2010.5683561.

[3] R. Reussner, S. Becker, J. Happe, H. Koziolek, K. Krogmann, and M. Kuperberg. The palladio component model, 2007. URL `https://www.researchgate.net/profile/Jens_Happe/publication/36452772_The_Palladio_component_model/links/0912f50f66f5a02a5e000000.pdf`. Last visited June 23, 2017.