Mutation Testing for Microservices

Stefan Winzinger

Distributed System Group, University of Bamberg, Germany
`stefan.winzinger@uni-bamberg.de`

Abstract. The microservice architectural style is currently of great interest both to research and industry. Since applications built by this style consist of many loosely coupled services, it is necessary to test their interactions by test suites. A crucial question is to decide which test cases are necessary and able to detect errors. A method to assure the quality of test cases is mutation testing. However, there are no mutation operators available yet which would enable the application of mutation testing specifically for microservices. This paper presents preliminary ideas for the creation of possible mutation operator whose application could help assure the quality of test cases by using mutation testing and therefore improve the quality of microservice systems.

Keywords: microservices, mutation testing, mutation operator

1 Motivation

Microservices have emerged as a trend over the last years and can be defined as small, autonomous services that work together [9]. The independence of the components in a microservice architecture makes it possible to test them in isolation. But testing on a higher level can become very difficult, especially for larger systems with many connections between the services [7]. Therefore, test cases are needed that detect faults, which only emerge while using several services in combination, as these faults are not detected while testing a single service.

A method to evaluate the potential of a test case suite is mutation testing. In general, mutation testing is a fault-based testing technique which creates a faulty set of programs by seeding faults, which are often done by programmers, into the program. A faulty program is called a mutant. By running the test suite against each of the mutants, a mutant is "killed" as soon as its fault is detected. The 'mutation score' is the ratio of the detected faults over the total number of seeded faults [8]. It improves with every mutant killed.

By mutating the program many faults can be produced at low cost [6]. However, mutation operators providing the rules to create faults are required in order to create faulty programs. Using mutation operators can produce programs whose faults are similar to those of real programs [2]. Therefore, mutation operators for microservices could help to create mutants automatically which would facilitate the assessment of the test case quality. Thus, quality and speed of test execution could be improved resulting in a faster delivery for the customer.

Mutation testing can be applied at unit level, integration level and specification level [8]. Since there are no mutation operators focusing on the microservice architecture identified yet, we want to identify some mutation operators being useful for microservices and evaluate them by applying them on a running system.

2 Research Outline

For our future work, we plan to investigate several generic mutation operators on unit, integration and specification level. We will consider existing mutation operators and their suitability for microservices and define new mutation operators adapted to the microservice architectural style to apply them to a technology used in practice:

Unit level
There are already many mutation operators defined for specific programming languages (e.g. C, Java or C# [9]). E.g. considering C, these mutation operators change a statement, an operator, a variable or a constant [1]. These mutation operators will probably be applicable for testing single microservices depending on the language used. However, faults introduced by using these mutation operators are not characteristic for microservices and can be detected by testing services in isolation. Therefore, no microservice-specific mutation operators can be introduced at unit level but established mutation operators can be used to assess test suites for isolated microservices.

Integration level
Mutation operators at the integration level are more interesting since microservices can be interpreted as independent units which communicate by using interfaces. Therefore, the integration level is a more promising area to define characteristic mutation operators. As described in [4] or [5], interfaces can be changed by removing or modifying parameters. This is a promising approach, especially since a potential failure of a service can be simulated by using this approach. Additionally, it would be possible to add a delay to the delivery of messages in order to simulate a network congestion which would force an alternative service to handle this message.

Specification level
In [3] Estelle Specifications are used to specify a system. Estelle Specifications are a Formal Description Technique which describes a system hierarchically. Some mutation operators are introduced which focus on the structure of the system. Especially mutation operators modifying the control flow among components could be transferred to microservices by considering the network of services. By changing the control flow of services (e.g. parallel instead of sequential execution of services and vice versa) faults might occur which should be covered by test cases.

Finally, the most promising mutation operators shall be applied to several test case suites on a running system and be evaluated regarding their suitability to assess the quality of test cases.

References

1. Agrawal, H., DeMillo, R., Hathaway, R., Hsu, W., Hsu, W., Krauser, E., Martin, R.J., Mathur, A., Spafford, E.: Design of mutant operators for the c programming language. Tech. rep., Technical Report SERC-TR-41-P, Software Engineering Research Center, Department of Computer Science, Purdue University, Indiana (1989)
2. Andrews, J.H., Briand, L.C., Labiche, Y.: Is mutation an appropriate tool for testing experiments? In: Proceedings of the 27th international conference on Software engineering. pp. 402–411. ACM (2005)
3. De Souza, S.D.R.S., Maldonado, J.C., Fabbri, S.C.P.F., De Souza, W.L.: Mutation testing applied to estelle specifications. In: System Sciences, 2000. Proceedings of the 33rd Annual Hawaii International Conference on. pp. 10–pp. IEEE (2000)
4. Delamaro, M.E., Maldonado, J.C., Mathur, A.P.: Integration testing using interface mutation. In: Software Reliability Engineering, 1996. Proceedings., Seventh International Symposium on. pp. 112–121. IEEE (1996)
5. Delamaro, M.E., Maldonado, J.C., Pasquini, A., Mathur, A.P.: Interface mutation test adequacy criterion: An empirical evaluation. Empirical Software Engineering 6(2), 111–142 (2001)
6. Do, H., Elbaum, S., Rothermel, G.: Supporting controlled experimentation with testing techniques: An infrastructure and its potential impact. Empirical Software Engineering 10(4), 405–435 (2005)
7. Dragoni, N., Giallorenzo, S., Lafuente, A.L., Mazzara, M., Montesi, F., Mustafin, R., Safina, L.: Microservices: yesterday, today, and tomorrow. In: Present and Ulterior Software Engineering, pp. 195–216. Springer (2017)
8. Jia, Y., Harman, M.: An analysis and survey of the development of mutation testing. IEEE Transactions on Software Engineering 37(5), 649–678 (2010)
9. Newman, S.: Building microservices: designing fine-grained systems. O'Reilly Media, Inc. (2015)