Software Reliability Engineering for Resilient Cloud Operations

Michael R. Lyu(✉) and Yuxin Su

The Chinese University of Hong Kong, Hong Kong, China
{lyu,yxsu}@cse.cuhk.edu.hk

Abstract. In the last decade, cloud environments become the most sophisticated software systems. Due to the inevitable occurrences of failures, software reliability engineering is top priority for cloud developers and maintainers. In this essay, we introduce several frameworks to provide resilient cloud operations from different development phases, ranging from fault prevention before deployment and fault removal at run-time.

Keywords: Software reliability engineering · Resilient cloud operation · Fault prevention · Fault removal

1 Introduction

In the recent years, IT enterprises have drastically increased development of their applications and services on cloud computing platforms, such as search engine, instant messaging app and online shopping. As cloud systems continue to grow in terms of complexity and volume, cloud failures become inevitable and critical, which may lead to service interruptions or performance degradation. Whether cloud failures can be properly managed will greatly impact company revenue and customer satisfaction. For example, in 2017, a downtime in Amazon led to a loss of 150+ million US dollars. Thus, the reliability of modern software is of paramount importance. Consequently, we identified several critical challenges commonly seen in industrial cloud systems and provide a general road-map from fault prevention and fault removal to improve the cloud reliability by resilient operations. First, as cloud systems are actively undergoing continuous feature upgrade and system evolution, the statistical properties of system monitoring data may change from time to time. Hence, to impede the deployment of the buggy cloud service, a fast and effective fault prevention mechanism for the source code and cloud services interface is a crucial task to address. In practice, however, fault prevention is hard to offer perfect cloud services without any runtime bugs or errors. Fault removal mechanisms can come to rescue after cloud deployment.

2 Fault Prevention for Cloud Services

In this section, we introduce fault prevention before the deployment of cloud services. We attempt to detect buggy code while the service is under development and discuss the testing approaches to verify the correctness of cloud service before actual deployment.

2.1 RESTful API Testing

Most industrial scale cloud services are programmatically accessed through Representational State Transfer (REST) APIs, which are a clear trend as composable paradigm to create and integrate cloud software. One of the key benefits of involving RESTful APIs is a systematic approach to software logic modeling leveraged by a growing usage of standardized cloud software stack. In the last few years, the OpenAPI Specification (OAS) has gradually become the de-facto standard to describe RESTful APIs from a functional perspective. The adequate testing of stateful cloud services via OpenAPI is difficult and costly. Failures generated by complex stateful interactions can be of high impact to customer, but they are hard to replicate.

To address the testing problem in an automatic way would certainly increase the reliability of cloud services. Fuzzing is a widely adopted approach to find bugs in software by feeding variety of test input. RESTler [1] first performs a lightweight static analysis on the API specification of a target cloud service and detects dependencies among test input. However, the automatically-generated fake paths will limit the combinatorial explosion of the fuzzing space due to the lack of feedback about grammar. To effectively induce the fuzzers to focus on fake paths (or branches), we consider the following design aspects. We maintain a resource pool which stores a sufficient number of fake paths to affect the fuzzing policy. Typically, as the fuzzer generates various mutations from one startup input, fake paths should provide different request coverage and be directly affected by the input so that the fuzzer will keep uncovering the trap. Various mechanisms for RESTful API testing based on this direction will be investigated and evaluated.

2.2 Software Defect Prediction

To improve software reliability, software defect prediction is utilized to assist developers in finding potential bugs and allocating their testing efforts. Traditional defect prediction studies mainly focus on designing hand-crafted features, which are input into machine learning classifiers to identify defective code. However, these hand-crafted features often fail to capture the semantic and structural information of programs. Such information is important in modeling program functionality and can lead to more accurate defect prediction. Software defect prediction is a process of building classifiers to predict code areas that potentially contain defects, using information such as code complexity and change history. The prediction results (i.e., buggy code areas) can place warnings for code reviewers and allocate their efforts. The code areas could be files, changes or methods.

In this essay, we introduce a framework called Defect Prediction via Convolutional Neural Network (DP-CNN) [4], which leverages deep learning for effective feature generation. We evaluate our method on seven open source projects in terms of F-measure in defect prediction. The experimental results show that in average, DP-CNN improves the state-of-the-art method by 12%.

3 Fault Removal after Deployment

In this section, we introduce several fault removal approaches from different perspectives, e.g. log analysis, emerging incident detection and fault localization.

3.1 Automated Log Mining for Fault Management

Logs are semi-structured text generated by logging statements in software source code. In recent decades, logs generated from cloud service have become imperative in the reliability assurance mechanism of cloud systems because they are often the only data available that traces cloud runtime information.

This essay presents a general overview of log mining techniques including how to automate and assist the writing of logging statements and how to employ logs to detect anomalies, predict failures, and facilitate diagnosis [3]. Traditional log analysis that is mainly based on ad-hoc domain knowledge or manually constructed and maintained rules is inefficient and ineffective for cloud systems due to its large scale and high complex in structure. This brings three major challenges to modern log analysis for cloud services. (1) Quality of the logging statements varies to a large extent because developers from different groups usually write logging statements based on their own domain knowledge and ad-hoc designs. (2) log mining based on manual rules is prohibited due to large volume of logs generated in a short time. (3) Due to the wide adoption of the DevOps software development concept, a new software version often appears in a short-term manner. Thus, corresponding logging statements update frequently as well. To address these challenges, we introduce several work about automated rule construction and critical information extraction.

3.2 Automatic Emerging Incident Mining from Discussion

When a high-damaging incident happens in cloud system, developers and maintainers generate an incident ticket or establishes a war-room to discuss the potential reasons and possible solution to fix the incident. Detecting emerging bugs or errors timely and precisely is crucial for developers and maintainers to provide resilient cloud services. However, the tremendous quantities of discussion comments, together with their imprecise and noisy descriptions increase the difficulties in accurately identifying newly-appearing issues. In this essay, we introduce an automated framework IDEA [2] to identify any new issues based on maintainers' discussions. IDEA takes the discussion of different incident tickets or war-room about the same target as input. To track the topic variations over discussion, AOLDA (Adaptively Online Latent Dirichlet Allocation) is employed for generating discussion-sensitive topic distributions. The emerging topics are then identified based on the typical anomaly detection method. Finally, IDEA visualizes the variations of different issues along with discussions, and highlights the emerging ones for better understanding.

3.3 Fault Localization from Structural Information

A critical research direction in cloud computing has been the defense against inevitable cloud failures and their prevention from causing service interruption or service degradation. We have consequently identified two critical challenges commonly seen in industrial cloud systems. First, when diagnosing failures for large-scale cloud systems, there is currently a lack of means to incorporating expert knowledge into the training of automated detection models. Second, although the dependencies of cloud service/resource can provide rich information for tracking the cascading effect of cloud failures, they have not been explicitly considered in existing methods of root cause analysis.

To address the above challenges, we introduce a resilient cloud systems framework by incorporating structural information and knowledge about the cloud systems. Our goal is to comprehensively improve the reliability of cloud systems and services. Particularly, the framework consists of an end-to-end pipeline of software reliability engineering, i.e., *anomaly detection*, *failure diagnosis*, and *fault localization*. Anomaly detection looks for systems' anomalous patterns that do not conform to normal behaviors, such as high CPU usage, low throughput. We propose a log-based anomaly detection model which can quickly learn unprecedented log patterns in an online manner and dynamically adapt to concept drift caused by system evolution. Moreover, failure diagnosis attempts to find the most significant problems directly induced by the failures; for example, abnormal Key Performance Indicator (KPIs). We therefore introduce an adaptive failure diagnosis algorithm with a human-in-the-loop strategy for efficient model training. Lastly, fault localization locates the root cause of a failure, such as a failed microservice or device. We also develop a novel fault localization technique for microservice architecture using dependency-aware collaborative filtering. Experimental evaluations will be conducted on this end-to-end framework regarding its effectiveness in providing resilient cloud operations.

References

1. Atlidakis, V., Godefroid, P., Polishchuk, M.: RESTler: stateful REST API fuzzing. In: Proceedings of the 41st International Conference on Software Engineering, ICSE 2019 (2019)
2. Gao, C., Zeng, J., Lyu, M.R., King, I.: Online app review analysis for identifying emerging issues. In: Proceedings of the 40th International Conference on Software Engineering, ICSE (2018)

3. He, S., Zhu, J., He, P., Lyu, M.R.: Experience report: system log analysis for anomaly detection. In: Proceedings of the 27th IEEE International Symposium on Software Reliability Engineering (ISSRE) (2016)
4. Li, J., He, P., Zhu, J., Lyu, M.R.: Software defect prediction via convolutional neural network. In: Proceedings of IEEE International Conference on Software Quality, Reliability and Security (QRS)