Reproducible Fault Injection at the Operating System Level

Sebastião Amaro
IST Lisbon & INESC-ID
Lisbon, Portugal

Miguel Matos
IST Lisbon & INESC-ID
Lisbon, Portugal

Pedro Fonseca
Purdue University
West Lafayette, Indiana, U.S.A

Abstract

Distributed systems underlie many critical systems and services. To ensure their robustness and reliability, they employ a multitude of algorithms, techniques, and mechanisms. Their inherent complexity makes them prone to bugs which often are only revealed when external faults occur at specific application states. A lot of work has been done in the field of exploration, where tools inject faults to search for bugs. However, developers still need to reproduce the conditions that led to the bugs to find their root cause and deploy a fix. State-of-the-art approaches in bug reproduction are focused only on JVM-based systems and support limited faults. Our approach aims to efficiently trace a distributed system execution to obtain a trace when a bug occurs, and use it to deduce *What happened?* and *When did it happen?*. With this information, we can create a scenario where the buggy workflow occurs. Our preliminary results show that our approach is able to reproduce bugs across different systems (from KV stores to consensus algorithms) written in different languages.

1 Introduction

Distributed systems are at the core of our modern digital society and power many of the services and infrastructures we rely on a daily basis. Unsurprisingly, when those systems fail, the result is failures and losses that can lead to large losses in revenue to organizations [1]. Therefore, it is of the utmost importance to have tools and techniques to find these bugs and their root cause as soon as possible to prevent downtime and ensure they do not reoccur.

Several tools such as [2–6] leverage fault injection to find bugs in different types of distributed systems. However, finding a bug is only a step towards making the system more robust. Developers must first confirm that it is an actual bug, then reproduction is necessary to find the root cause, and finally, they can deploy a fix and push the change. In this pipeline of solving bugs, reproduction is one of the key steps — in fact a study found that *"developers spend a vast majority of the resolution time (69%) on reproducing the failure"* [7]. Reproduction is not easy for developers since there is not a lot of information available. This makes it extremely difficult to create a scenario where the buggy workflow occurs quickly and exhibits the symptom. When it comes to reproducing *fault-induced bugs* in production distributed systems more challenges emerge, since it is not only necessary to determine what fault happened, but developers also need to know at what timing it happened, since most of *fault-induced bugs* are only triggered if the fault is injected at a key timing. Anduril [8] is a fault injection tool designed to reproduce fault-induced failures in deployed distributed systems, but it only works in systems based on the Java Virtual Machine.

In this paper, we will describe our novel solution to reproducing *fault-induced bugs*, which can reproduce bugs in all types of distributed systems. Our goal is to allow developers to reproduce *fault-induced bugs*, quickly and systematically. This is accomplished by deducing *What happened* and *When it happened* from a buggy trace. This approach has multiple challenges: firstly, we have to collect a trace in production; this is only possible if the tracing has an extremely low overhead and does not interfere with the deployment. Secondly, we have to use proper techniques and insights to deduce external faults and relevant state information from the trace. Thirdly, we have to create a tool that can inject external faults in precise states in all types of systems.

Our key observation to make these challenges tractable is to rely only on externally visible effects as indications that the application transitioned to a new state. This simplifies the search space as, from this perspective, only interactions with the environment are relevant. In practice, we leverage Linux eBPF to monitor the applications' interaction with the environment (i.e. syscalls), and to inject faults at strategic points in the system execution. We created TESOR which leverages eBPF to trace systems with low overhead. With a buggy trace collected, we generate schedules where faults are defined alongside the state at which they should occur, schedules are then run with ROSE until we find a schedule that reproduces the bug. Our preliminary results show that we can obtain enough information about the system at the cost of an overhead of ≈2.5% per node.

We leveraged ROSE to reproduce multiple bugs with 100% replay accuracy and some with less in 5 different systems, written in different languages. We are finishing the automatic schedule generation therefore not all the schedules for these bugs are not automatically generated based on a trace.

2 Overview

ROSE and TESOR are tools designed to help developers reproduce bugs that occur in their systems due to unexpected external events. By deploying TESOR on their systems, when

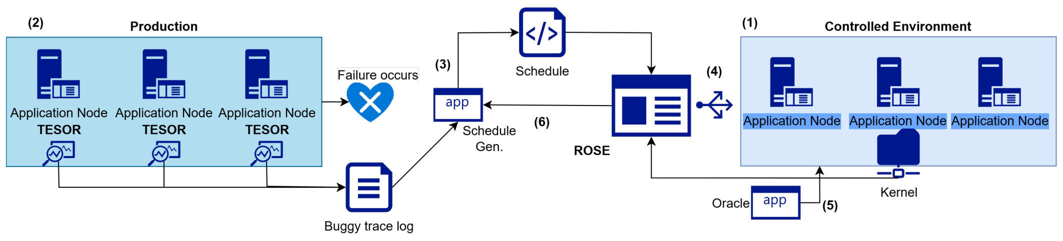

Figure 1. Workflow.

bugs occur a trace is saved so it can later be processed to generate a schedule that reproduces the bug. To run the schedules, we leverage ROSE, which allows us to run a schedule in a representative and controlled environment.

ROSE is responsible for running a schedule that represents, a bug reproduction execution. It keeps track of the state of the deployment and injects faults when the state conditions are met. ROSE supports changing the outcome of any system call, which can represent a vast amount of faults such as disk failures, memory failures, etc. Process pauses/restarts/crashes, as well as, network faults which can be packet drops and network partitions. We run ROSE in a controlled environment where the Deployment state is tracked via eBPF *kernel probes*. When an application node reaches the conditions described in the schedule the corresponding fault is injected. After a schedule finishes a user-provided oracle is run to check if the bug occurred.

While ROSE provides a way to reproduce an execution, manually writing the concrete schedulle might be impractical for complex systems. To fulfill this gap we developed TESOR which traces running production systems. When a bug occurs the trace is saved so it can later be used to generate a schedule that reproduces the bug. We trace: system call errors, which gives us key information about the external environment and the possible faults that occurred. Relevant application functions, by relevant application functions we imply application function calls that are not frequent during a standard execution. The key insight here is that when faults occur, specific code paths that do not occur in standard execution are explored. These can give helpful insights into the application node state. Network, by tracing the network and saving delays between connections, and their frequency we can deduce different network faults. Process state changes, keeping track of the changes in the processes of the application nodes gives us a way of finding if processes are stuck waiting for a certain event, or crashed.

3 Workflow

Our workflow is detailed in Figure 1. Step 1 is to distinguish between relevant function calls and frequent ones. To this end, we run TESOR in a controlled environment and separate them into these two categories. 2. TESOR is then deployed in a production environment, waiting for a bug to occur. 3. From the buggy trace we find what external faults occurred, and take a first guess at what state they occurred. 4. ROSE receives this schedule sets up the controlled environment, starts the workload, and injects the faults accordingly. 5. After the execution finishes, the user-provided oracle runs to check if the bug occurred. 6. If the bug is not shown, we increase the state details of the faults and return to step 4.

4 Research Questions

Our evaluation intends to answer three different questions: (1) Can our approach successfully reproduce fault-induced bugs (2) Can our approach from a buggy trace create a schedule that reproduces the bug with a high reproduction rate? (3) Can TESOR be used in active production systems?

We do not yet have full answers to these questions yet. Still, our approach is already able to automatically reproduce some production bugs.

Acknowledgments This work was supported by Fundação para a Ciência e a Tecnologia (FCT) under grants UIDB/50021/2020 and PTDC/CCI-COM/4485/2021 (Ainur).

References

[1] "Google lost $1.7m in ad revenue during youtube outage," accessed: 2025-1-01. [Online]. Available: https://www.foxbusiness.com/technology/google-lost-ad-revenue-during-youtube-outage-expert
[2] Mohan *et al.*, "Finding Crash-Consistency Bugs with Bounded Black-Box Crash Testing," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2018)*.
[3] R. Alagappan *et al.*, "Correlated crash vulnerabilities," in *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, GA, 2016, pp. 151–167.
[4] Meng *et al.*, "Greybox fuzzing of distributed systems," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '23.
[5] Liu *et al.*, "Dcatch: Automatically detecting distributed concurrency bugs in cloud systems," in *Proceedings of the Twenty-Second International Conference on Architectural Support for Programming Languages and Operating Systems*, 2017.
[6] Alvaro *et al.*, "Lineage-driven fault injection," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '15, 2015.
[7] Zhang *et al.*, "Pensieve: Non-intrusive failure reproduction for distributed systems using the event chaining approach," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017.
[8] Pan *et al.*, "Efficient reproduction of fault-induced failures in distributed systems with feedback-driven fault injection," in *Proceedings of the ACM SIGOPS 30th Symposium on Operating Systems Principles*, 2024.