ModelFuzz: Model guided fuzzing of distributed systems.

Srinidhi Nagendra*

Max Planck Institute for Software Systems

February 12, 2025

Large-scale distributed systems form the core infrastructure for many software applications. It is well-known that designing such systems is difficult due to interactions between concurrency and faults, and subtle bugs often show up in production. Thus, designing testing techniques that cover diverse and interesting program behaviors to find subtle bugs has been an important research challenge.

Coverage-guided fuzzing, which guides test generation toward more coverage, has been effective in exploring diverse executions, mainly in the sequential setting, using structural coverage criteria as a feedback mechanism [1, 2]. However, adopting coverage-guided fuzzing for testing distributed system implementations is nontrivial since there is no common notion of *coverage* for distributed system executions. Unfortunately, structural code coverage criteria such as line coverage can ignore the orderings of message interactions in a system, thus missing interesting schedules. On the other hand, more detailed criteria, such as traces of messages, may provide too many coverage goals and thus consider each random trace a new behavior, giving up the advantages of coverage-guided exploration.

In this talk, I will present a new approach to use the *state coverage* in an *abstract formal model* of the system as a coverage criterion and present *model-guided fuzzing* of distributed systems. Abstract formal models are often developed in the design phase of distributed systems to model and formally analyze the underlying protocols [3, 4, 5, 6, 7, 8]. We show that these artifacts are also beneficial in the continuous testing infrastructure of the implementations themselves. Our experiments show that a formal model can serve as a good "guide" for a random testing engine—this is because the formal model often captures the important scenarios of the protocol, and coverage of states in the model correlates well with coverage of interesting behaviors in the implementation.

At a high level, the abstract models recognize semantically interesting behavior; the use of abstract states is a way to provide coverage criteria that capture

*Joint work with Ege Berkay Gulcan, Burcu Kulahcioglu Ozkan, Rupak Majumdar

program semantics. Of course, the use of abstract models is not a panacea: the abstraction may not cover certain implementation details where bugs may lurk. However, lack of structural coverage after model-guided exploration can indicate where additional testing effort should focus, as well as point out aspects of implementation behavior that are not covered by the model.

We have implemented our algorithm for testing distributed systems implementations using TLA+ models [3] of protocols. In a nutshell, our testing algorithm proceeds as follows. We start by exploring random schedules of messages, but feed the same sequence of messages to the TLA+ model. We modify the TLC model checker [9] to obtain the set of reachable model states corresponding to the explored schedule. We mark a schedule as "interesting" if it covers a new state of the abstract model. We perform, as in coverage-guided fuzzing, a *mutation* of an interesting schedule by swapping the receipt order of two randomly chosen messages or changing the processes to crash. Applying a mutation to an event schedule gives a new schedule to explore that is similar to the original schedule but likely to exercise new system behavior.

We applied our algorithm to test the implementations Etcd-raft and RedisRaft. Our evaluation shows that model-guided fuzzing leads to higher coverage and can detect bugs faster than pure (unguided) random testing, structural code-coverage guided fuzzing, and trace-based coverage-guided fuzzing. Besides reproducing known bugs, we discovered 13 previously unknown bugs in the implementations of Etcd-raft and RedisRaft. Moreover, four of the new bugs could only be detected by model-guided fuzzing.

References

[1] Marc Heuse, Heiko Eißfeldt, Andrea Fioraldi, and Dominik Maier. AFL++, January 2022.

[2] Andreas Zeller, Rahul Gopinath, Marcel Böhme, Gordon Fraser, and Christian Holler. The fuzzing book, 2019.

[3] Leslie Lamport. *Specifying Systems, The TLA+ Language and Tools for Hardware and Software Engineers*. Addison-Wesley, 2002.

[4] Ankush Desai, Vivek Gupta, Ethan K. Jackson, Shaz Qadeer, Sriram K. Rajamani, and Damien Zufferey. P: safe asynchronous event-driven programming. In Hans-Juergen Boehm and Cormac Flanagan, editors, *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13, Seattle, WA, USA, June 16-19, 2013*, pages 321–332. ACM, 2013.

[5] Pantazis Deligiannis, Alastair F. Donaldson, Jeroen Ketema, Akash Lal, and Paul Thomson. Asynchronous programming, analysis and testing with state machines. In David Grove and Stephen M. Blackburn, editors, *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation, Portland, OR, USA, June 15-17, 2015*, pages 154–164. ACM, 2015.

[6] Pantazis Deligiannis, Narayanan Ganapathy, Akash Lal, and Shaz Qadeer. Building reliable cloud services using coyote actors. In Carlo Curino, Georgia Koutrika, and Ravi Netravali, editors, *SoCC '21: ACM Symposium on Cloud Computing, Seattle, WA, USA, November 1 - 4, 2021*, pages 108–121. ACM, 2021.

[7] James Bornholt, Rajeev Joshi, Vytautas Astrauskas, Brendan Cully, Bernhard Kragl, Seth Markle, Kyle Sauri, Drew Schleit, Grant Slatton, Serdar Tasiran, Jacob Van Geffen, and Andrew Warfield. Using lightweight formal methods to validate a key-value storage node in amazon S3. In Robbert van Renesse and Nickolai Zeldovich, editors, *SOSP '21: ACM SIGOPS 28th Symposium on Operating Systems Principles, Virtual Event / Koblenz, Germany, October 26-29, 2021*, pages 836–850. ACM, 2021.

[8] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. How amazon web services uses formal methods. *Commun. ACM*, 58(4):66–73, 2015.

[9] Yuan Yu, Panagiotis Manolios, and Leslie Lamport. Model checking tla$^+$ specifications. In Laurence Pierre and Thomas Kropf, editors, *Correct Hardware Design and Verification Methods, 10th IFIP WG 10.5 Advanced Research Working Conference, CHARME '99, Bad Herrenalb, Germany, September 27-29, 1999, Proceedings*, volume 1703 of *Lecture Notes in Computer Science*, pages 54–66. Springer, 1999.