Benchmark Suite for Evaluating Chaos Engineering Tools

15-300, Fall 2020

Yiwen(Victor) Song

http://www.andrew.cmu.edu/user/yiwenson/ProjectWebpage.html

November 4, 2020

1 Project Description

For my proposed research project, I will be working in the Composable Systems Lab led by Prof. Heather Miller in CMU Institute for Software Research. Her research mainly focuses on distributed systems. I will be working closely with her Ph.D. student Christopher Meiklejohn, who has previously worked on a related project involving chaos engineering tools. Prof. Rohan Padhye is also advising on this project, and he is an expert in software testing.

To provide some background, distributed systems are now ubiquitous, and they need to be built resilient to partial failures. It is usually difficult to reason about or formally prove the correctness of error-handling and fallback behaviors for large-scale distributed systems. An emerging software engineering method to tackle this problem is called chaos engineering, which is detailed by "1) injecting faults in a production system, 2) observing the impact of such faults, and 3) building new knowledge about the strengths and weaknesses of the resilience of the system"[6].

Although most chaos engineering tests are conducted in production, some fault injection tests are not particular to the production environment and can be done earlier in the development process, which can dramatically expedite debugging. Some researchers are proposing to do fault injection as integration tests, and various tools have been developed. For example, Zhang et al.[6] presented ChaosMachine, which does chaos engineering tests on exception-handling in the JVM. Christopher has also built a tool that can help catch bugs in distributed applications.

The problem that this project is trying to solve is to come up with a benchmark for such chaos engineering tools. The idea is similar to YCSB [3] for cloud services, or DaCapo [2] for Java applications. Currently, there is no existing benchmark for chaos engineering or fault injection tools that are used to find bugs in distributed applications. If we can successfully build such a benchmark suite, it can be used to test and evaluate existing and future chaos engineering tools, which can have a profound impact in this field.

In this project, we plan to first collect some real-world fault-tolerance bugs by watching industry talks and reading bug reports. To build our benchmark suite, we will extend an existing open-source distributed application by adding these predetermined failures. The challenge of this project is to make this test bed accessible to different chaos engineering tools. We need to support some variation of a plugin mechanism, but details are yet to be finalized.

2 Project Goals

2.1 75% Project Goal

- Collect one or two fault-tolerance bugs from real-world applications
- Extend an existing open-source distributed application by adding these bugs
- Support for chaos engineering tools written in Python
- Test the benchmark on one chaos engineering tool
- Provide a feedback report for developers
- Write a research summary

2.2 100% Project Goal

- Collect a number of fault-tolerance bugs from real-world applications
- Extend an existing open-source distributed application by adding these bugs
- Good support for variations of chaos engineering tools written in Python
- Test the benchmark on more than one chaos engineering tool

- Support toggling each bug on or off
- Provide a detailed feedback report for developers
- Write a complete research paper

2.3 125% Project Goal

- Collect a large number of quality fault-tolerance bugs from real-world applications
- Extend an existing open-source distributed application by adding these bugs
- Good support for variations of chaos engineering tools written in a variety of languages
- Test the benchmark on multiple chaos engineering tools and receive feedback from researchers
- Support toggling each bug on or off
- Support for easily adding new bugs or customizing bugs
- Provide a detailed feedback report for developers
- Write and publish a complete research paper

3 Project Milestones

3.1 1st Technical Milestone

By the end of this semester, I anticipate to have watched several industry talks on chaos engineering and collected some fault-tolerance bugs. I should have also read several papers on building benchmarks and have some idea of how we can proceed.

3.2 1st Biweekly Milestone: February 15th

By the 1st biweekly milestone, I should have started working on understanding the open-source distributed application and thinking about how we can add predetermined failures to it.

3.3 2nd Biweekly Milestone: March 1st

By the 2nd biweekly milestone, I hope to have have added some predetermined failures to the application and be able to inject the failure.

3.4 3rd Biweekly Milestone: March 15th

By the 3rd biweekly milestone, I anticipate to be still working on extending the application and, at the same time, start thinking about how to support evaluating chaos engineering tools.

3.5 4th Biweekly Milestone: March 29th

By the 4th biweekly milestone, I hope to have some preliminary results on using the benchmark to evaluate Christopher's chaos engineering tool.

3.6 5th Biweekly Milestone: April 12th

By the 5th biweekly milestone, if things go well, I will have collected some results and start working on a research paper.

3.7 6th Biweekly Milestone: April 26th

By the 6th biweekly milestone, I anticipate to be working on making the benchmark more extensible to support different chaos engineering tools and collect more results, while continue working on my research paper.

3.8 7th Biweekly Milestone: May 10th

By the end of semester, I hopefully have a somewhat complete framework suite and a research paper.

4 Literature Search

As of this writing, I have read several latest papers on chaos engineering cited below [1][5][6]. I have also glanced through some papers on testing and frameworks [2][3][4], but I will need to do more literature search on that.

5 Resources Needed

We will be using Python and building upon an open-source distributed application. I will be using my personal computers and I don't anticipate any need for further computation power.

References

[1] Peter Alvaro, Kolton Andrus, Chris Sanden, Casey Rosenthal, Ali Basiri, and Lorin Hochstein. Automating failure testing research at internet scale. In *Proceedings of the Seventh ACM Symposium on Cloud Computing*, pages 17–28, 2016.

[2] Stephen M. Blackburn, Robin Garner, Chris Hoffmann, Asjad M. Khang, Kathryn S. McKinley, Rotem Bentzur, Amer Diwan, Daniel Feinberg, Daniel Frampton, Samuel Z. Guyer, Martin Hirzel, Antony Hosking, Maria Jump, Han Lee, J. Eliot B. Moss, Aashish Phansalkar, Darko Stefanović, Thomas VanDrunen, Daniel von Dincklage, and Ben Wiedermann. The dacapo benchmarks: Java benchmarking development and analysis. *SIGPLAN Not.*, 41(10):169–190, October 2006.

[3] Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM Symposium on Cloud Computing*, SoCC '10, page 143–154, New York, NY, USA, 2010. Association for Computing Machinery.

[4] René Just, Darioush Jalali, and Michael D Ernst. Defects4j: A database of existing faults to enable controlled testing studies for java programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, pages 437–440, 2014.

[5] Kennedy A Torkura, Muhammad IH Sukmana, Feng Cheng, and Christoph Meinel. Cloudstrike: Chaos engineering for security and resiliency in cloud infrastructure. *IEEE Access*, 8:123044–123060, 2020.

[6] Long Zhang, Brice Morin, Philipp Haller, Benoit Baudry, and Martin Monperrus. A chaos engineering system for live analysis and falsification of exception-handling in the jvm. *IEEE Transactions on Software Engineering*, 2019.