PREFAIL: Programmable and Efficient Failure Testing Framework

Pallavi Joshi, Haryadi S. Gunawi, and Koushik Sen
University of California, Berkeley

With the arrival of the cloud computing era, large-scale distributed systems are increasingly in use. These systems are built out of tens of thousands of commodity machines that are not fully reliable and can fail from time to time [1, 2, 7, 10, 14, 15]. Thus, the software that runs on these systems has a great responsibility to correctly recover from frequent, diverse hardware failures.

Even if distributed systems are built with reliability and fault tolerance as primary goals [6, 7, 8], their recovery protocols are often buggy. For example, the developers of Hadoop File System (HDFS) [16] have dealt with 91 recovery issues over its four years of development [9]. There are many reasons for this. Sometimes developers fail to anticipate the kind of failures that a system can face in a real setting (*e.g.*, tolerate crashes but not corruption). Even if all kinds of failures are anticipated, the recovery implementation might be incorrect. There have been many serious consequences (*e.g.*, data loss, unavailability) of the presence of recovery bugs in real deployed systems [3, 4, 5, 9].

To improve the reliability of large-scale distributed systems, failure testing has become a mainstream technique to test software reliability. One major challenge is that the number of combinations of failures to explore is potentially large [9, 12]. One direct way to explore this failure space is via randomness. For example, random injection of failures is employed by the developers at Google [4], Yahoo! [17], Microsoft [18], Amazon [10], and others [11]. Random fault-injection is relatively simple to implement, but the downside is that it could easily miss corner-case failure scenarios. Thus, there is a need for systematic techniques that can smartly explore the space of failure scenarios and find bugs efficiently.

There has been some work that proposes novel techniques for smart exploration of failures [12, 13]. They primarily address single failures during program execution. However, large-scale distributed systems face frequent, multiple, and diverse failures. And thus, there is a need to advance the state-of-the-art of failure testing for large-scale distributed systems.

In this work, we address the challenges of failure testing by introducing PREFAIL, a programmable and efficient failure testing framework that can can explore failures systematically, including multiple combinations of diverse failures. More specifically, PREFAIL comes with the following features:

1. Well-defined failure optimizations: PREFAIL comes with optimizations that completely remove redundant fault-injection tests. For example, crashes are injected only before write I/Os; a naive framework would inject crashes around read and write I/Os. The optimizations bring 1 to 21 times (5 on average) of improvement depending on the workload and failure type.

2. Programmable exploration policies: PREFAIL allows testers to express failure exploration policies of different complexities so that they can use the right ones at the right times. For example, they could use some coarse policies that result in few experiments in the development mode, finer policies during nightly builds, and more elaborate policies that test a greater number of failures before a big release. In our experience, by using different policies, we only need to run one to three orders of magnitude fewer experiments compared to a brute-force approach, but still found important recovery bugs.

3. Parallelizable testing workflow: Since we target the developers of large-scale systems who tend to have many machines for testing purposes, we designed PREFAIL such that its test workflow is parallelizable. Thus, it can explore multiple failure sequences concurrently and achieve a considerable speed-up in its performance.

4. Triaging support for efficient debugging: In automated failure testing, a number of experiments can fail because of the same bug. When tens or hundreds of experiments fail, a tester could get easily overwhelmed. To reduce the debugging effort, PREFAIL automatically triages failed experiments by clustering failed experiments according to the bugs that caused them to fail. The triaging support can also sort failed experiments according to the importance of bugs that caused them.

Overall, PREFAIL is a practical and well-equipped failure testing framework that can help today's large-scale distributed systems "prevail" against failures. We have found 6 new bugs in the latest version of Hadoop File System (HDFS), and 16 bugs in an older version.

References

[1] Lakshmi N. Bairavasundaram, Garth R. Goodson, Shankar Pasupathy, and Jiri Schindler. An Analysis of Latent Sector Errors in Disk Drives. In *SIGMETRICS '07*.

[2] Lakshmi N. Bairavasundaram, Garth R. Goodson, Bianca Schroeder, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. An Analysis of Data Corruption in the Storage Stack. In *FAST '08*.

[3] Mike Burrows. The Chubby lock service for loosely-coupled distributed systems. In *OSDI '06*.

[4] Tushar Chandra, Robert Griesemer, and Joshua Redstone. Paxos Made Live - An Engineering Perspective. In *PODC '07*.

[5] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Michael Burrows, Tushar Chandra, Andrew Fikes, and Robert Gruber. Bigtable: A Distributed Storage System for Structured Data. In *OSDI '06*.

[6] Brian Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking Cloud Serving Systems with YCSB. In *SoCC '10*.

[7] Jeffrey Dean. Underneath the covers at google: Current systems and future directions. In *Google I/O*.

[8] Garth Gibson. Reliability/Resilience Panel. In *HEC FSIO '10*.

[9] Haryadi S. Gunawi, Thanh Do, Pallavi Joshi, Peter Alvaro, Joseph M. Hellerstein, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, and Koushik Sen. FATE and DESTINI: A Framework for Cloud Recovery Testing. In *NSDI '11 (to Appear; currently available as a technical report: UCB/EECS-2010-127*.

[10] Alyssa Henry. Cloud Storage FUD: Failure and Uncertainty and Durability. In *FAST '09*.

[11] Todd Hoff. Netflix: Continually Test by Failing Servers with Chaos Monkey. http://highscalability.com, December 2010.

[12] Lorenzo Keller, Paul Marinescu, and George Candea. AFEX: An Automated Fault Explorer for Faster System Testing. 2008.

[13] Paul D. Marinescu, Radu Banabic, and George Candea. An Extensible Technique for High-Precision Testing of Recovery Code. In *Usenix ATC '10*.

[14] Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz Andre Barroso. Failure Trends in a Large Disk Drive Population. In *FAST '07*.

[15] Bianca Schroeder and Garth Gibson. Disk failures in the real world: What does an MTTF of 1,000,000 hours mean to you? In *FAST '07*.

[16] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The Hadoop Distributed File System. In *MSST '10*.

[17] Hadoop Team. Fault Injection framework: How to use it, test using artificial faults, and develop new faults. http://issues.apache.org.

[18] Junfeng Yang, Tisheng Chen, Ming Wu, Zhilei Xu, Xuezheng Liu, Haoxiang Lin, Mao Yang, Fan Long, Lintao Zhang, and Lidong Zhou. MODIST: Transparent Model Checking of Unmodified Distributed Systems. In *NSDI '09*.