Jeremy Cooley

5/6/2019

***Simple Testing Can Prevent Most Critical Failures An Analysis of Production Failures in Distributed Data-intensive Systems***

Ding Yuan, Yu Luo, Xin Zhuang, Guilherme Renna Rodrigues, Xu Zhao, Yongle Zhang, Pranay U. Jain, Michael Stumm

University of Toronto

Large, complex software systems often have so many individual moving parts that it’s incredible that these systems don’t fail, and fail catastrophically, much more often than they actually do. According to the fault study discussed in the article, “almost all failures require only 3 or fewer nodes to reproduce, which is good news considering that these services typically run on a very large number of nodes. However, multiple inputs are needed to trigger the failures with the order between them being important.” It seems to me that developers are extremely proficient at coding a module, component, or subsystem, but when the interaction or order of subsystems come into play, developers are prone to miss faults that could end up being catastrophic for the entire system. Because the authors of this particular article discovered that “the majority of catastrophic failures could easily have been prevented by performing simple testing on error handling code,” the authors developed their own static checker, called Aspirator, to locate bugs, failures, and bad practices in code. According to the paper, “Running Aspirator on the code of 9 distributed systems located 143 bugs and bad practices that have been fixed or confirmed by the developers.”

Outages and failures in widespread, highly used websites and services such as Amazon, Google, and Reddit are extremely noticeable, and are almost inevitable, even with the intense testing these systems undergo. To help answer the questions of “why these systems still experience failures and what can be done to increase their resiliency,” the authors “studied 198 randomly sampled, user-reported failures of five dataintensive distributed systems that were designed to tolerate component failures and are widely used in production environments. The specific systems we considered were Cassandra, HBase, Hadoop Distributed File System (HDFS), Hadoop MapReduce, and Redis.”