COBE: Cascading Outage Bugs Elimination

Objective: Prevent service downtimes in datacenter and mobile systems caused by *cascading outage bugs* (*"CO bugs"* in short), a new class of bugs that can cause simultaneous or cascades of failures to many or all system nodes/components. In this COBE project, we will: (1) study the anatomy of CO bugs, (2) develop CO-bug detection tools to unearth CO bugs, and (3) build CO-bug containment solutions to prevent CO bugs from causing an outage in deployment.

Motivation

"No single point of failure" is the mantra for high availability. Hardware arguably is no longer a single point of failure as the philosophy of redundancies has permeated systems design. On the other hand, software redundancy such as N-version programming is deemed expensive and only adopted in mission-critical software such as in avionics. Thus, in many important systems today, software bugs are single points of failure.

Some software bugs are "benign"; they might fail some subcomponents but the whole system can tolerate the partial failure. Some other bugs however can lead to outages such as configuration bugs and state-corrupting concurrency bugs, which have been analyzed extensively in literature. However, our large-scale studies of cloud bugs and outages [1, 2] reveal a new class of outage-causing bugs. In particular, *there are bugs that can cause simultaneous or cascades of failures to each of the individual nodes in the system*, which eventually leads to a major outage. We name them *cascading outage (CO) bugs*.

Real cases of CO bugs: We present 3 real cases (among many) of CO bugs that motivate the COBE project. **(a) HBase:** HBase can run tens to thousands of region servers, each is responsible for a set of table regions. A case of CO bug surfaced when a region server crashed due to a bad handling of corrupt region files. In HBase, if a server dies, another live server will be picked to handle the orphan region files. This means that the picked server will execute the *same* buggy code (bad handling of corrupt region files), thus crashing the new server too. As the whole failover logic *repeats*, all the nodes become unavailable. **(b) DynamoDB Case:** A similar CO bug caused an outage of Amazon DynamoDB in September 2015 [3, 4]. Here, the storage servers attempted to refresh their membership data to a busy metadata service. As the storage servers observe timeouts, they disqualify themselves from accepting requests, eventually causing an outage. **(c) Safari Case:** CO bugs not only affect datacenter systems, but also client/mobile systems. Early this year, "Apple's [iPhone/iPad] Safari browser crashed *worldwide*" [5, 6], caused by a communication bug between the browser's auto-search-suggest component and the backend server (similar to the DynamoDB communication bug between the storage servers and the metadata service).

Our approach/principles: The cases above reveal single points of failure in hidden dependencies; there is a single root failure (*e.g.*, file corruption, unavailable service) that eventually affects the entire system. Our view of the problem is that in all of the cases above, the complete outage can be alleviated if we can **(a)** understand the bug pattern, **(b)** detect the potential cascading/simultaneous impacts, and **(c)** allow the system to continue in degraded mode. For example, in the HBase case, some regions can be made unavailable while most of other regions (potentially hundreds of thousands of them) are still servable. In the DynamoDB case, the storage nodes can perhaps continue in read-only mode temporarily as opposed to rejecting all requests. In the Safari case, the browser can continue to run with auto-search-suggest disabled.

We now describe the three stages of our COBE project. Our **target systems** include *distributed systems* (*e.g.*, HBase, Cassandra, Spark) and *mobile software systems* (*e.g.*, Google Chromium browser).

• **Phase 1: Study of CO bugs anatomy.** Establishing the anatomy of special bugs is a crucial step in eliminating them, which we have done successfully in the past for concurrency [7], distributed [8], performance [9], and scalability bugs [10]. Thus, our first step is to understand the patterns of CO bugs.

To give a flavor of our initial findings, below we describe some CO bug patterns we have observed. *(i) Repeated buggy logic after failover*: Here, a failover repeats the same buggy logic in other nodes (*e.g.*, the HBase case above). We also observed the same pattern in another HBase bug but the root failure is log-cleaning exception. *(ii) Positive feedback loop:* This is the case where some nodes declared dead/busy due to request overload, but the recovery introduces more load (*e.g.*, a re-mirroring storm), which causes more nodes to be marked dead/busy, which then causes more recovery. This is similar to pattern (i) but is more about busy nodes as opposed to crashing. *(iii) Simultaneous, deterministic crash paths on non-critical service errors:* Here, there is a crash-leading path executed in all nodes but the root error was caused by non-critical services/components (*e.g.*, the DynamoDB and Safari cases). *(iv) Distributed deadlock:* Here, each node is waiting for other nodes to progress (similar to the classical deadlock problem in multi-threaded process). For example, a Cassandra coordination bug caused all nodes to keep gossiping during bootstrapping but never entering a normal state.

• **Phase 2: CO Detection (Offline):** Based on the patterns we establish, our next step is to create CO bug detection tools. For example, to detect pattern (i), a program analysis needs to connect crash path and recovery path and checks whether the recovery path may lead to the same crash path. To detect pattern (iii), we need to perform path backward-slicing from crash points to the root causes and categorizes whether they are supposedly tolerable or can be replaced with a degraded mode. To reduce the number of crash points to analyze, we will devise some heuristics or leverage post-mortem failure diagnosis. Overall, we will explore and build various CO detection approaches. This phase only unearths the possibility of CO bugs but is not enough to isolate them in practice, which leads us to the next phase, CO containment.

• **Phase 3: CO Containment (Online):** In this phase, we will add to software systems the capability of containing the cascading nature of CO bugs in live deployment. Below are some of the containment principles we will develop and integrate to our target systems. First, software systems must distinguish hardware and software failures. When a "node" is dead, it is typically caused by one of them (not both). CO bugs tend to kill some machines gradually and leave some "trails" (*e.g.*, exception logs, core dumps). If the same trail appears in all nodes, the system should be suspicious of the existence of CO bugs. Second, upon detection of CO bug, the system can go in degraded mode (rather than continue and eventually shut down the entire system). For example, the three sample cases above are containable. The challenge is to develop domain-specific containment strategies (*e.g.*, stop the failover, move to read-only mode, skip auto-suggest feature). Third, as we introduce degrade mode, other components must be degrade-tolerant. Today's systems typically only accept two modes: work or fail. However, a degraded mode can cause unintended side effects to other layers (*e.g.*, skipping a buggy load balancer may cause unintended backlogs in some nodes), which must be taken care properly. Overall, we will formulate and address the challenges of building CO containment.

In conclusion, outages are providers' nightmare. Worldwide software crash undermines the company's credibility, especially as major news headlines zealously report high-profile outages. Worse, rivals always seek to capitalize on competitors' outages (*e.g.*, suggestions to switch to Chrome during Safari outage [6]). We believe COBE will provide a significant contribution in solving this problem.

References

[1] H. S. Gunawi, M. Hao, T. Leesatapornwongsa, T. Patana-anake, T. Do, J. Adityatama, K. J. Eliazar, A. Laksono, J. F. Lukman, V. Martin, and A. D. Satria, "What Bugs Live in the Cloud? A Study of 3000+ Issues in Cloud Systems," in *Proceedings of the 5th ACM Symposium on Cloud Computing (SoCC)*, 2014.

[2] H. S. Gunawi, M. Hao, R. O. Suminto, A. Laksono, A. D. Satria, J. Adityatama, and K. J. Eliazar, "Why Does the Cloud Stop Computing? Lessons from Hundreds of Service Outages," in *SUBMISSION to the Proceedings of the 7th ACM Symposium on Cloud Computing (SoCC)*, 2016.

[3] "Summary of the Amazon DynamoDB Service Disruption and Related Impacts in the US-East Region (September 20, 2015)," https://aws.amazon.com/message/5467D2.

[4] "Amazon Disruption Produces Cloud Outage Spiral (September 20, 2015)," http://www.informationweek.com/cloud/infrastructure-as-a-service/amazon-disruption-produces-cloud-outage-spiral/d/d-id/1322279.

[5] "Apple's Safari browser crashed worldwide (January 27, 2016)," http://money.cnn.com/2016/01/27/technology/iphone-safari-crash-apple.

[6] "Safari outage affects iPhone and iPad users worldwide," http://www.dailymail.co.uk/sciencetech/article-3419088/Safari-outage-affects-iPhone-iPad-users-worldwide.html.

[7] S. Lu, S. Park, E. Seo, and Y. Zhou, "Learning from Mistakes — A Comprehensive Study on Real World Concurrency Bug Characteristics," in *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2008.

[8] T. Leesatapornwongsa, J. F. Lukman, S. Lu, and H. S. Gunawi, "TaxDC: A Taxonomy of Non-Deterministic Concurrency Bugs in Datacenter Distributed Systems," in *Proceedings of the 21st International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2016.

[9] R. O. Suminto, A. Laksono, A. D. Satria, T. Do, and H. S. Gunawi, "Towards Pre-Deployment Detection of Performance Failures in Cloud Distributed Systems," in *The 7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2015.

[10] T. Leesatapornwongsa, C. Stuardo, H. Ke, J. F. Lukman, R. O. Suminto, D. H. Kurniawan, and H. S. Gunawi, "Scale-Checking Distributed Systems and Debugging Large-Scale Bugs on Just One Machine," in *SUBMISSION to the Proceedings of the 12th Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.