Simple Testing Can Prevent Most Critical Failures

Ding Yuan et. al.

Overview

The authors of this paper analyse real-world failures reported on different types of distributed systems, by analyzing the tickets for their failures owing to their richness of detail. The analysis of this data was performed qualitatively. The authors report several findings, five of which are general, four pertain to logging, and three deal with catastrophic failures specifically, rounding up the total to twelve findings. The paper also discusses Aspirator, its code checker that analyses code for bad practices and trivial mistakes. It goes over how Aspirator works and how it translates to real-world use. It concludes that rigorous code reviews and use of good programming practices would go a long way towards preventing critical failures.

Thoughts on the Paper

Overall, I think that this is a well-written, well-researched paper with solid definitions and assumptions, a clear, tight focus on its goals, a good structure and presents some very compelling conclusions along with suggestions that seem feasible to implement and indeed, others that come across as plain common sense. In fact, from reading the abstract and introduction, I assumed that only the some of the findings about catastrophic failures and general findings would be discussed, but the paper has a wealth of information. I think the paper undersells itself in the introduction!

The paper is also extremely honest and forthright about its own shortcomings. The ‘Limitations’ section addressed essentially every major concern I had with the paper, especially the likelihood that the data the authors worked with is biased. I really appreciate that the actions proposed to address these shortcomings actually feel substantial and not merely like lip-service, e.g. not extrapolating on the distribution of failures because their data may not be representative. I also appreciate that the paper did not gloss over the mixed reception to Aspirator, where I imagine it wouldn’t have been too difficult to cherry-pick positive reviews.

The general findings are interesting because of the implications they have for testing. The fact that most of the failures are deterministic, and can be manifested on no more than three nodes in general, means that testing for these failures should be extremely feasibly. Not only that, the paper finds that just over half of all non-deterministic failures occur due to timing issues which can be worked around, bringing even more potential failures into the fold of testing.

The paper’s findings regarding catastrophic failures being caused by trivial errors are incredible to me. I understand that the authors would want to keep the companies who experienced these failures confidential, but in this case, it might have helped to have some context on the kind of companies under review here. It seems implausible that large companies like Amazon or Google would have such lapses in code reviews, and in contrast more believable that the developers in question here are from smaller companies with not enough people to do everything. In any case, it seems that to

avoid catastrophic failures, simple measures such as checking the code for errors, not shipping incomplete code and having regular code reviews go a long way.

There are also a few assorted positives about this paper that I appreciated. I liked the structure of this paper. The findings are listed based on categories, namely, general, log-related and catastrophic-failures. The organization feels intuitive and logical. Another positive I'd like to draw attention to is the definition of catastrophic failures, especially in contrast to similar definitions in the paper 'Why Does the Cloud Stop Computing? Lessons from Hundreds of Service Outages' which had vague definitions of essential and non-essential services. In this paper, by putting the focus on the quantity of users affected instead of the quality of services disrupted, the definition is more universally applicable and removes most doubts, barring the minor 'what counts as "most" users?' question raised by it. I really like that the paper not only recommended solid, actionable fixes, but went the extra mile and also developed a code checker to implement the points they made in the paper.

Minor Nitpicks

Although most shortcomings that came to my mind about this paper are addressed in the paper itself, I also have a few minor nitpicks. These aren't especially big, but are a few small suggestions. Firstly, since the concept of catastrophic failures was introduced in the introduction, it might have made more sense to also put the definition there instead of in Section 4, several pages later.

My second nitpick concerns the section on how Aspirator was received by people in the industry. Although the paper says that Aspirator garnered mixed reception, the only example of negative feedback included in the paper comes from some expressing the idea that handling every exception is unnecessary. It seems to me that after a section hammering home the point that checking code for trivial mistakes and poor logic in handing exceptions, any reader would immediately dismiss this feedback as invalid. Since this is the only example of negative feedback included in the paper, it implies that either all the criticism was in this vein, or there was valid criticism about Aspirator (e.g. it not being thorough, too many false positives) and the paper didn't include them.

Thirdly, and this is extremely subjective, but I think that some findings such as Finding 6 concerning explicit failure-related messages in logs and Finding 9 which says that most failures are reproducible by unit tests come across as rather redundant. For the first example here, the other log-related findings seem to be more relevant to the point the paper is making, and have more applications for reproducibility or scope for improvements, but also make Finding 6 seem obsolete to me. The unit test finding also seems redundant to me since so many of the general findings emphasize that it is feasible to conduct small-scale tests to simulate these failures. I believe that it should be obvious then that unit tests for the same would also be possible. But these are even more trivial nitpicks since their inclusion detracts nothing from the paper and makes things clearer. My only observation is that they feel less impactful than some other findings.

Conclusion

This paper is a well-researched, well-written collection of compelling findings and suggestions for avoiding failures in distributed systems, with a tight focus on its subject matter and suggestions that seem very feasible to implement. Aspirator is the embodiment of the suggestions made by the paper and though not without its problems, demonstrates the positives of the suggestions made in this paper.