an interesting/influential/important paper from the world of CS every weekday morning, as selected by Adrian Colyer

Use of Formal Methods at Amazon Web Services

NOVEMBER 24, 2014

tags: Amazon, Distributed Systems, Formal methods

Use of Formal Methods at Amazon Web Services (http://research.microsoft.com/en-us/um/people/lamport/tla/formal-methods-amazon.pdf) – Newcombe et al 2014

Leslie Lamport recently gave a talk at the **React conference (http://reactconf.com/)** on the specification language TLA. I wasn't there to hear the talk, but I was intrigued enough to dig in and find out a little more. Especially since I have some experience with Z and CSP. My journey led me to today's paper choice, a wonderful find and something that I hope spreads further into accepted industry practice.

Something a lot of people believe…

> *In industry, formal methods have a reputation of requiring a huge amount of training and effort to verify a tiny piece of relatively straightforward code, so the return on investment is only justified in safety- critical domains such as medical systems and avionics.*

That isn't necessarily so:

> *Our experience with TLA+ has shown that perception to be quite wrong. So far we have used TLA+ on 10 large complex real-world systems. In every case TLA+ has added significant value, either finding subtle bugs that we are sure we would not have found by other means, or giving us enough understanding and confidence to make aggressive performance optimizations without sacrificing correctness. We now have 7 teams using TLA+, with encouragement from senior management and technical leadership. Engineers from entry level to Principal have been able to learn TLA+ from scratch and get useful results in 2 to 3 weeks, in some cases just in their personal time on weekends and evenings, and without help or training.*

The AWS teams have been using formal specification since 2011, and the paper tells the story of how this came to be and how their use has spread. It's highly readable so I encourage you to check it out. I will focus on the rationale and benefits for this short summary.

Building distributed systems is hard

> *S3 is just one of tens of AWS services that store and process data that our customers have entrusted to us. To safeguard that data, the core of each service relies on fault-tolerant distributed algorithms for replication, consistency, concurrency control, auto-scaling, load balancing, and other coordination tasks.*

When it comes to building real production systems, there's more to it than just coding up algorithms from the literature:

> *There are many such algorithms in the literature, but combining them into a cohesive system is a major challenge, as the algorithms must usually be modified in order to interact properly in a real-world system. In addition, we have found it necessary to invent algorithms of our own. We work hard to avoid unnecessary complexity, but the essential complexity of the task remains high.*

How can you gain confidence that your system is correct? AWS used multiple verification methods (short of formal methods), but still weren't finding all the subtle bugs that may lurk.

> *before launching such a service, we need to reach extremely high confidence that the core of the system is correct. We have found that the standard verification techniques in industry are necessary but not sufficient. We use deep design reviews, code reviews, static code analysis, stress testing, fault-injection testing, and many other techniques, but we still find that subtle bugs can hide in complex concurrent fault-tolerant systems.*

Some of the most difficult to detect bugs are not in the code, but in the design…

> *some of the more subtle, dangerous bugs turn out to be errors in design; the code faithfully implements the intended design, but the design fails to correctly handle a particular 'rare' scenario.*

Finding a precise design language

> *In order to find subtle bugs in a system design, it is necessary to have a precise description of that design. There are at least two major benefits to writing a precise design; the author is forced to think more clearly, which helps eliminate 'plausible hand-waving', and tools can be applied to check for errors in the design, even while it is being written.*

AWS needed a precise design language, but also a pragmatic one:

> *As our designs are unavoidably complex, we needed a highly expressive language, far above the level of code, but with precise semantics. That expressivity must cover real-world concurrency and fault-tolerance. And, as we wish to build services quickly, we wanted a language that is simple to learn and apply, avoiding esoteric concepts. We also very much wanted an existing ecosystem of tools.*

They 'found what we were looking for in TLA+, a formal specification language.'

Benefits

TLA+ helped AWS find subtle bugs in S3, DynamoDB, EBS, and an internal distributed lock manager, as well as verifying several optimizations. The effects go deeper than just finding bugs though.

> *TLA+ has been helping us shift to a better way of designing systems.*

The process begins by stating clearly 'what needs to go right?' by defining correctness properties. Safety properties define what the system is allowed to do, and liveness properties define what the system must eventually do.

The benefits are not just in the initial system design, but over the lifetime of the system.

> *We have found that writing a formal specification pays several dividends over the lifetime of the system. All production services at Amazon are under constant development, even those released years ago; we add new features that customers have requested, we re-design components to handle massive increases in scale, and we improve performance by removing bottlenecks. Many of these changes are complex, and they must be made to the running system with no downtime. Our first priority is always to avoid causing bugs in a production system, so we often need to answer the question, "is this change safe?" We have found that a major benefit of having a precise, testable model of the core system is that we can rapidly verify that even deep changes are safe, or learn that they are unsafe without doing any harm. In several cases we have prevented subtle, serious bugs from reaching production. In other cases we have been able to make innovative performance optimizations – e.g. removing or narrowing locks, or weakening constraints on message ordering – which we would not have dared to do without having model checked those changes.*

Spreading the word

Success breeds success, but it's best not to scare people off too early:

> *…we have found that software engineers more readily grasp the concept and practical value of TLA+ if we dub it: exhaustively testable pseudo-code. We initially avoid the words 'formal', 'verification', and 'proof', due to the widespread view that formal methods are impractical.*

Formal methods seem to be becoming an important part of the AWS processes:

> *At AWS, formal methods have been a big success. They have helped us prevent subtle, serious bugs from reaching production, bugs that we would not have found via any other technique. They have helped us to make aggressive optimizations to complex algorithms without sacrificing quality. So far, seven teams have used TLA+, and all have found high value in doing so. At the time of writing, more teams are starting to use TLA+. We believe that use of TLA+ will accelerate both time-to-market and quality of these projects. Executive management is now proactively encouraging teams to write TLA+ specs for new features and other significant design changes. In annual planning, managers are now allocating engineering time to use TLA+.*

Summary

It's great to see the value of formal methods being recognised once more. This paper is also a good insight into the distributed systems design methodology at AWS. It adds another dimension to AWS's challenge to the rest of the industry: keep up if you can!

from → Uncategorized

14 Comments leave one →

Trackbacks

1. Photon: Fault-tolerant and scalable joining of continuous data streams | the morning paper
2. Blazes: Coordination analysis for distributed programs | the morning paper
3. Online, Aysnchronous Schema Change in F1 | the morning paper
4. How to write a 21st Century Proof | the morning paper
5. ZooKeeper's Atomic Broadcast Protocol: Theory and Practice | the morning paper
6. Combining static model checking with dynamic enforcement using the Statecall Policy Language | the morning paper
7. IronFleet: Proving Practical Distributed Systems Correct | the morning paper
8. A Year in Papers | the morning paper
9. Uncovering bugs in Distributed Storage Systems during Testing (not in production!) | the morning paper
10. An empirical study on the correctness of formally verified distributed systems | the morning paper
11. Linux 基金会透露未来 Linux 内核可能会引入形式验证 | Linux Story
12. Linux 基金会透露未来 Linux 内核可能会引入形式验证 | News Pod
13. Linux 基金会透露未来 Linux 内核可能会引入形式验证
14. Growing a protocol | the morning paper