Formal Modeling and Analysis of Distributed Systems (Finding Critical Bugs Early!)

Challenge. Distributed systems are notoriously hard to get right. Programming these systems is challenging because of the need to reason about correctness in the presence of myriad possible interleaving of messages and failures. Unsurprisingly, it is common for service teams to uncover correctness bugs after deployment. Formal methods (FM)[1] can play an important role in addressing this challenge. But **the key requirement for "success" in an industrial setting would be the ability to integrate FM in all the phases of development process** from system design, to implementation, to unit and integration testing, and even in production through runtime monitoring. Moreover, in most known applications of formal techniques in practice, analysis used during design phase (e.g., TLA+) cannot be connected to popular techniques for validation/testing of the implementation (e.g., Jepsen, QuickCheck). We would like to ensure that efforts invested in writing specifications during the design verification phase must not get wasted and should play an important role in the later phases of the software life cycle, e.g., during testing of implementation.

In this talk, we will provide an overview of the P programming framework. We will discuss how P is currently being used extensively inside Amazon (AWS) to integrate principles of FM in to the development lifecycle of service teams and deliver complex distributed services with higher assurance of correctness. **P is an open-source tool with users and contributors from industry and academia. It would be great to through this talk to engage with the distributed systems community.**

P Framework. P is a state machine-based programming language for modeling and specifying complex distributed systems. The P framework has four important parts: (1) a high-level **state machine-based programming language,** allowing programmers to specify their system design as a collection of communicating state machines (which is how they normally think about complex system design). P being a programming language (rather than a mathematical modeling language) has been one of the key reasons for its large-scale adoption; (2) it supports **scalable analysis engines** (based on automated reasoning techniques like model checking and symbolic execution) to check that the distributed system modeled in P satisfy the desired correctness specifications. P also leverages distributed compute to scale model checking to large system design and has helped find critical bugs (in AWS and Microsoft services) early on in design phase itself; (3) it supports **code conformance checking** to bridge the gap between design specifications and the actual implementation. Using runtime monitoring, we check that the system implementation traces (or logs) satisfy the P specifications checked during the design phase; (4) fourth and the final component is the ability to **integrate** these analysis engines and code conformance checks **into the CI/CD of the service teams**. For example, if P model checking fails then it can block the build-release pipelines of the service teams. In this talk, we will provide an overview of how each of these components played an important role in the adoption of P.

Experience. In our limited experience (2+ years) of using P inside AWS, we have observed that P has helped developers in three critical ways: (1) "**P as a thinking tool**": Writing formal specifications in P forces developers to think about their system design rigorously, and in turn helped in bridging gaps in their understanding of the system. A large fraction of the bugs was eliminated in the process of writing specifications itself! (2) "**P as a bug finder**": Model checking helped find corner case bugs in system design that were missed by stress and integration testing. (3) "**P helped boost developer velocity**": After the initial overhead of creating the formal models, future updates and feature additions could be rolled out faster as these non-trivial changes are rigorously validated before implementation. We are continuing to explore ways in which we can **add rigor in to our development process** using formal methods.
Links: (a) P Github: https://p-org.github.io/P/. (b) P Case studies: https://p-org.github.io/P/casestudies/ (c) P usage in Amazon S3 Strong Consistency Launch [Pi-Week Talk]: https://www.twitch.tv/videos/962963706?t=0h15m16s (d) Related Publications: https://p-org.github.io/P/publications/

[1] The term Formal Methods is used to refer to the wide area of techniques from model checking, to property-based testing, to runtime monitoring. These are approaches that can be easily integrated into development process but does require engineers to create formal models and specifications of their system.

Short Bio

Ankush Desai is a Senior Applied Scientist in the Database Services (DBS) group at AWS. He is currently working on building formal tools and techniques that help developers reason about the correctness of complex distributed services across AWS (S3, DBS, EBS,). These techniques range from lightweight approaches like model checking, to systematic testing, to more rigorous deductive verification that provides mathematical proofs. The goal of Ankush's work is to integrate the principles of formal methods in all the phases of development process from system design, to implementation, to unit and integration testing, and even in production. Before joining the DBS group, Ankush was part of the S3 team and worked on the Amazon S3's Strong Consistency project (Pi-Week Talk).

Ankush graduated with a PhD in computer science from UC, Berkeley (2019). His PhD. research had an impact both in Industry and Academia for which he was awarded the Sevin Rosen Funds Award for Innovation. Before joining graduate school, Ankush spent 2+ years working at Microsoft Research, India working on formal verification of device drivers and distributed systems. Webpage: https://ankushdesai.github.io/