Formal Modeling and Analysis of Distributed Systems

ABSTRACT

Large scale distributed systems are difficult to design and test. Hence, it is not uncommon for design bugs to escape guard rails of design reviews, functional and stress testing, and get uncovered by customers in production. Formal methods can play an important role in addressing these challenges and help catch these bugs early on in the development process.

In this tutorial, we present formal methods tools and techniques used at Amazon Web Services (AWS) to reason about the correctness and performance of distributed systems. We will provide an introduction to the P framework and also share our experience of integrating P in all the phases of the development process from design, to testing, to after deployment. One of our goals with this tutorial is to ignite a discussion between the formal methods and systems community on challenges faced by practitioners when building complex distributed systems.

ACM Reference Format:

. 2023. Formal Modeling and Analysis of Distributed Systems. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Distributed systems are notoriously hard to get right. Programming these systems is challenging because of the need to reason about both correctness and performance in the presence of myriad possible interleaving of messages, failures, and variation in workloads. Unsurprisingly, it is common for service teams to uncover design-level bugs after deployment. Formal methods (FM) can play an important role in addressing this challenge. Formal methods have proven to be extremely valuable to the builders of large scale distributed systems [5, 8, 9], and to the researchers designing distributed protocols [11]. Increasingly, formal verification papers are getting published in top systems conferences [3, 4, 6, 7, 12], but, most of these papers involve application of formal methods by experts in the domain and not by developers in industry. On the other hand, the systems community have been building complex, novel system and protocols, but they are rarely backed by formal methods tools. We seek to ignite a discussion between the two communities and discuss the state-of-the-art, along with open challenges that needs attention.

The goal of our tutorial is to provide an overview of *how practitioners are leveraging formal methods when designing complex distributed systems.* We would like to share our experience applying formal methods to distributed systems at Amazon Web Services (AWS). We will present P, a state machine-based programming language for formally modeling and specifying complex distributed systems (P-Github [10]). P is a unified framework that not only allows developers to reason about correctness (using model checking) but also conduct performance analysis (using probabilistic simulation) of the system design. P also enables integrating FM in all the phases of development process from system design, to implementation, to unit and integration testing, and even in production. In this tutorial, we will get the audience started on P using well known distributed protocols as examples, demonstrate all the different aspects of the framework, especially, highlight the basic principles of formal methods. We will use examples from AWS to highlight our learning.

P Framework

The P framework has four important parts:
(1) a high-level **state machine-based programming language** that allows 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 in industry;
(2) P 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 leverages distributed compute to scale *model checking* to large system design and has helped find critical bugs inside AWS early on in design phase itself. P also supports a backend explorer that performs *random probabilistic simulations* of the system for reasoning about performance quantitatively.
(3) P provides **code conformance checking** to bridge the gap between design specifications and the actual implementation. Using runtime monitoring, we check that the system 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.

Impact of P inside AWS. Teams across AWS from storage (e.g., S3), to databases (e.g., Aurora, DynamoDB), to compute (e.g., EC2) have been using P to reason about the correctness of complex distributed protocols driving these services. P has helped find and eliminate several critical bugs in the design, early on, these bugs could not be found using the traditional testing approaches. From our experience (3+ 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

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

creating the formal models, future updates and feature additions could be rolled out faster as these non-trivial changes are rigorously validated before implementation.

Thinking beyond correctness is critical. Typically, formal methods is focused on checking correctness properties (safety and liveness). This makes sense: safety violations cause issues like data corruption and loss which are correctly considered to be among the most serious issues with distributed systems. But, when making design decision for distributed systems (e.g. what protocols to use for committing transactions), in addition to correctness, the following questions are often debated: What latency can customers expect, on an average and in an outlier cases? How sensitive is the design to network latency or packet loss? How do availability and durability scale with the number of replicas? Traditionally, these questions have been addressed by prototyping, closed-form modelling, and simulations. In particular, database community have traditionally used performance simulation to evaluate and compare protocols [1, 2], however, there is definitely a need for approaches that can reduce the barrier to entry and make performance analysis a part of the design process itself. To this end, we extended the P analysis backend with capabilities to perform probabilistic simulation of the system design and collect metrics from these simulations. The key point to note is that for conducting performance simulations, we are reusing the same formal models that were created for checking the correctness of the system.

2 TUTORIAL OVERVIEW

We will provide a **1.5 hour tutorial** (can also do a 3 hour tutorial if there is interest), broken down into 3 modules.

Module 1: Introduction to Formal Methods and P (30 min). In this module, we will introduce the basic principles of formal methods by modeling two popular protocols (two phase commit and optimistic concurrency control) in P.

(**10 min**) Enumerate existing approaches and highlight the benefits of light-weight formal methods.

(**10 min**) Use the well-known two-phase-commit (2PC) protocol to introduce the P language. This will provide an overview of the language primitives and highlight the key aspect of modeling distributed system design as communicating state machines.

(**10 min**) Now that the audience have basic understanding of the P language, we will give a code walk-through over the formal models of the complex optimistic concurrency control (OCC) protocol. The objective is to demonstrate how creating abstract formal models of complex protocols can help in thinking about these protocols abstractly and reason about their correctness.

Module 2: Checking Correctness of Design and Implementation (30 min). In this module, we will introduce the process of writing correctness specifications (global invariants) and checking them on both the formal models and the implementation of the system.

(**5 min**) We will revisit the formal models of the 2PC and OCC protocols. We will discuss the correctness properties that the system must satisfy, (for e.g., atomicity for 2PC and seriazability for OCC).

(**10 min**) We will do a code walk-through over the atomicity and serialiability safety properties in P.

(**10 min**) We will demonstrate how the P model checker can check these correctness properties on the formal models.

(**5 min**) We will finally demonstrate how using the P runtime monitoring framework, we can check these specifications on logs generated from the implementation. The goal is to show that the P specifications can be connected to the implementation and keep the design specifications in sync with it.

Module 3: Going beyond Correctness (30 min). In this module, we will first introduce the importance of thinking about performance trade-offs when designing complex systems and not just correctness. We will describe how we can leverage the formal models used for checking correctness to also answer some basic performance questions using simulations.

(**10 min**) Using anecdotal evidence from our experience at AWS, we will describe how design decisions can have implications on performance of the system. The goal is to convey to the audience that *performance is correctness as well*!

(**10 min**) Using the example of 2PC and OCC, we will demonstrate how random simulations of the formal models can help answer simple questions about the behavior of these protocols under different failure and workload scenarios.

(**10 min**) We will finally conclude by summarizing how we can cross pollinate ideas between the database and formal methods community to build tools and techniques that can help practitioners build reliable systems.

2.1 Target Audience

The target audience for this tutorial are practitioners building systems or designing protocols and would like to reason about their system, these include graduate students, researchers, and industry developers. The audience will be exposed to challenges and open problems that needs attention from the systems and formal methods communities.

2.2 Presenters:

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. Before joining the DBS group, Ankush was part of the S3 team and worked on the Amazon S3's Strong Consistency project. Ankush graduated with a PhD in computer science from UC, Berkeley in 2019.

Marc Brooker is a Senior Principal Engineer at AWS. He is currently focused on scalable, managed, and serverless transactional database systems. Before joining the databases organization, he was part of the team that launched AWS Lambda, and worked on virtualization, storage, and placement optimization systems at AWS. Marc graduated with a PhD in electrical engineering from the University of Cape Town in 2008.

REFERENCES

[1] Rakesh Agrawal, Michael J. Carey, and Miron Livny. 1987. Concurrency Control Performance Modeling: Alternatives and Implications. *ACM Trans. Database Syst.* 12, 4 (nov 1987), 609–654. https://doi.org/10.1145/32204.32220

[2] Philip A. Bernstein and Nathan Goodman. 1981. Concurrency Control in Distributed Database Systems. *ACM Comput. Surv.* 13, 2 (jun 1981), 185–221. https://doi.org/10.1145/356842.356846

[3] James Bornholt, Rajeev Joshi, Vytautas Astrauskas, Brendan Cully, Bernhard Kragl, Seth Markle, Kyle Sauri, Drew Schleit, Grant Slatton, Serdar Tasiran, Jacob Van Geffen, and Andrew Warfield. 2021. Using lightweight formal methods to validate a key-value storage node in Amazon S3. In *SOSP 2021*. https://www.amazon.science/publications/using-lightweight-formal-methods-to-validate-a-key-value-storage-node-in-amazon-s3

[4] Tej Chajed, Joseph Tassarotti, M. Frans Kaashoek, and Nickolai Zeldovich. 2019. Verifying Concurrent, Crash-Safe Systems with Perennial. In *Proceedings of the 27th ACM Symposium on Operating Systems Principles* (Huntsville, Ontario, Canada) *(SOSP '19)*. Association for Computing Machinery, New York, NY, USA, 243–258. https://doi.org/10.1145/3341301.3359632

[5] Ankush Desai, Vivek Gupta, Ethan K. Jackson, Shaz Qadeer, Sriram K. Rajamani, and Damien Zufferey. 2013. P: safe asynchronous event-driven programming. In *PLDI*. ACM, 321–332.

[6] Chris Hawblitzel, Jon Howell, Manos Kapritsos, Jay Lorch, Bryan Parno, Justine Stephenson, Srinath Setty, and Brian Zill. 2015. IronFleet: Proving Practical Distributed Systems Correct. In *Proceedings of the ACM Symposium on Operating Systems Principles (SOSP)* (proceedings of the acm symposium on operating systems principles (sosp) ed.). ACM - Association for Computing Machinery. https://www.microsoft.com/en-us/research/publication/ironfleet-proving-practical-distributed-systems-correct/

[7] Jacob R. Lorch, Yixuan Chen, Manos Kapritsos, Bryan Parno, Shaz Qadeer, Upamanyu Sharma, James R. Wilcox, and Xueyuan Zhao. 2020. Armada: Low-Effort Verification of High-Performance Concurrent Programs. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation* (London, UK) *(PLDI 2020)*. Association for Computing Machinery, New York, NY, USA, 197–210. https://doi.org/10.1145/3385412.3385971

[8] Chris Newcombe. 2014. Why amazon chose TLA+. In *International Conference on Abstract State Machines, Alloy, B, TLA, VDM, and Z*. Springer, 25–39.

[9] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. 2015. How Amazon Web Services uses formal methods. *Commun. ACM* (2015). https://www.amazon.science/publications/how-amazon-web-services-uses-formal-methods

[10] P-GitHub. 2021. The P Programming Langugage. https://github.com/p-org/P.

[11] TLA+-Examples-GitHub. [n. d.]. https://github.com/tlaplus/Examples

[12] Jianan Yao, Runzhou Tao, Ronghui Gu, and Jason Nieh. 2022. {DuoAI}: Fast, Automated Inference of Inductive Invariants for Verifying Distributed Protocols. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 485–501.