Programming Technologies for Highly-Scalable Data-Intensive Applications

Gowtham Kaki
Purdue University gkaki@purdue.edu

Introduction

The evolution of computational systems has always trended in the direction of increasing complexity. While early machines were designed for simple, deterministic tasks, modern computational systems routinely operate in distributed environments connected by adversarial and unreliable networks. Ensuring safety and security of computational systems in these environments requires novel programming abstractions and formal (reasoning) methods that enable the programmers overcome the overwhelming complexity of reasoning about these systems. My research attempts to make foundational advances in programming languages and formal methods with aim of making provably-safe distributed systems development accessible to mainstream software developers. Formal methods have seen increasing adoption in mission-critical systems in the recent years. DARPA's HACMS program, for example, demonstrated the effectiveness of formal verification in securing unmanned aerial vehicles. However, their widespread adoption remains a challenge due to the steep learning curve, integration complexity, and scalability concerns associated with formal methods. Bridging the gap between theoretical advances and practical deployment in industry requires research into more accessible tools, automated reasoning techniques, and programming abstractions that can seamlessly integrate with existing development workflows. My work is motivated by these challenges and aims to make formal methods more practical and impactful for real-world software systems. My research program involves three major thrusts to address scalability, generality, and usability challenges associated with formal methods. I will next summarize each research thrust, highlight significant contributions, and present their broader impacts and applications.

Thrust 1. Scalability

Programming safe and reliable distributed systems is extremely hard [5, 4, 2]. The complexity stems from having to simultaneously reason about several failure modes, such as reordered and dropped messages, network partitions, and node crashes. Complex combinatorial reasoning involving low-level events and high-level application logic is hard for humans and machines alike, making it challenging to apply formal methods at scale. Consequently, formal certification of safety and reliability guarantees of distributed systems remains out of reach for most application developers. Any formal verification that is currently done in the industry is carried out at the level of distributed protocol specifications, e.g., using TLA+ [3] or P [1], with no way to translate the guarantees to implementations.

Several decades of research into programming languages has shown that modularity and abstraction is often the key to overcoming programming complexity. The abstractions typically used to program distributed systems today expose asynchronous communication primitives and require the explicit reasoning about the dynamics of the network state in relation to faults. To scale formal verification to real-world systems implementations, we need novel programming abstractions that factor out the complexity of fault tolerance into separate modules, and let programmers focus on the high-level protocol logic.

What is the ideal interface to RSM that optimizes for programmability and efficiency?

References

[1] Ankush Desai, Vivek Gupta, Ethan Jackson, Shaz Qadeer, Sriram Rajamani, and Damien Zufferey. P: Safe asynchronous event-driven programming. In *Proceedings of the 34th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '13, page 321–332, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450320146. doi: 10.1145/2491956.2462184. URL https://doi.org/10.1145/2491956.2462184.

[2] Yu Gao, Wensheng Dou, Feng Qin, Chushu Gao, Dong Wang, Jun Wei, Ruirui Huang, Li Zhou, and Yongming Wu. An empirical study on crash recovery bugs in large-scale distributed systems. In *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2018, page 539–550, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450355735. doi: 10.1145/3236024.3236030. URL https://doi.org/10.1145/3236024.3236030.

[3] Leslie Lamport. The temporal logic of actions. *ACM Trans. Program. Lang. Syst.*, 16(3):872–923, May 1994. ISSN 0164-0925. doi: 10.1145/177492.177726. URL https://doi.org/10.1145/177492.177726.

[4] Tanakorn Leesatapornwongsa, Mingzhe Hao, Pallavi Joshi, Jeffrey F. Lukman, and Haryadi S. Gunawi. Samc: Semantic-aware model checking for fast discovery of deep bugs in cloud systems. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, page 399–414, USA, 2014. USENIX Association. ISBN 9781931971164.

[5] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. How amazon web services uses formal methods. *Commun. ACM*, 58(4):66–73, mar 2015. ISSN 0001-0782. doi: 10.1145/2699417. URL https://doi.org/10.1145/2699417.