Dara: Hybrid Model Checking of Distributed Systems

Vaastav Anand
University of British Columbia
Canada
vaastav@cs.ubc.ca

ABSTRACT

Building correct implementations of distributed systems continues to elude us. Solutions consist of abstract modeling languages such as TLA+, PLusCal, which specify models of systems and tools like Coq, and SPIN which verify correctness of models but require considerable amount of effort, or transparent model checkers like MODIST, CMC and CHESS which suffer from state space explosion, rendering them impractical to use as they are too slow.

We propose Dara, a novel hybrid technique that combines the speed of abstract model checkers with the correctness and ease-of-use of transparent model checkers. Dara utilizes tests as well as a transparent model checker to generate logs from real executions of the system. The generated logs are analyzed to infer a model of the system which is model-checked by SPIN to verify user-provided invariants. Invariant violations are reported as likely bug traces. These traces are then passed to a replay engine which tries to replay the traces as real executions of the system to remove false positives. We are currently evaluating Dara's efficiency and usability.

CCS CONCEPTS

• **Software and its engineering → Formal software verification**;

KEYWORDS

Model Checking, Distributed Systems

ACM Reference Format:
Vaastav Anand. 2018. Dara: Hybrid Model Checking of Distributed Systems. In *Proceedings of the 26th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '18), November 4–9, 2018, Lake Buena Vista, FL, USA.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3236024.3275438

1 BACKGROUND & MOTIVATION

Distributed systems are difficult to debug and understand. Despite the difficulty, developers of distributed systems strive for correctness as these systems have become a key part of modern data center environments. Additionally, failures caused by bugs in distributed systems can be costly [2, 22].

Modeling languages like TLA+, PlusCal, are used to model protocols and system behavior, which are used by tools like Coq, SPIN to verify safety and liveness properties. [20]. A limitation of these tools is that they are applied on a model of the system and not the actual system. There is still a gap between the specification and implementation of systems leaving the system prone to bugs.

Verdi is a framework for implementing and formally verifying fault tolerant distributed systems in Coq [23]. It requires serious developer effort as Verdi requires developers to provide a specification of the system in OCaml and write proofs of correctness. Similarly, IronFleet is a methodology for automated verification of distributed systems [12]. It also requires significant developer effort as the developers must write a formal specification, a distributed protocol layer, and proof annotations.

Transparent model checking and model-based testing have been widely used in the past to find bugsin actual implementations of distributed systems [1, 6, 10, 11, 14, 16, 17, 19, 24, 25]. As these model checkers focus on testing, unmodified, distributed and concurrent systems, they are easy to use as developers need to provide minimal input. However, this approach leads to massive state space explosion. DEMETER [11], built on top of MODIST [25], reduces the state space explosion by separating out the system's global state from the local states of the system's components during exploration. SAMC [17] incorporates application-specific information in state-space reduction policies to alleviate redundant interleavings of messages, crashes and reboots. In Dara, we try to offer a pragmatic approach of exploring states by searching for error traces in the abstract state space and trying to map these error traces back to the concrete state space to find bugs in systems.

In this work, I describe Dara, a hybrid technique that combines the speed of abstract model checkers with the ease-of-use of transparent model checkers, aimed at reducing developer effort to find bugs in distributed systems.

2 APPROACH

Our approach for finding bugs only requires system source code, names of important system variables, and the safety and liveness invariants as input. The Dara pipeline is shown in Figure 1.

1.Log Generation — Dara infers a model of the system to find bugs. Dara makes use of a transparent model checker that generates logs of the system that capture the state of the system, as well as the state of sent and received messages. Our transparent model checker, GoDist, leverages the Go runtime as an interposition layer between the application and OS to capture all system calls and thread scheduling information. Specifically, GoDist logs the state of processes during execution. For the logs to capture the state of the system, we make use of automatic instrumentation, in which, all in scope variables are logged at the entry and exit of each function. Upon executing a sending, receiving, or local event, vector clock timestamps [18] are attached to logged values.

2.Model Inference — Logs have been used to infer specifications of distributed systems [3, 15]. Dara aggregates logs of multiple

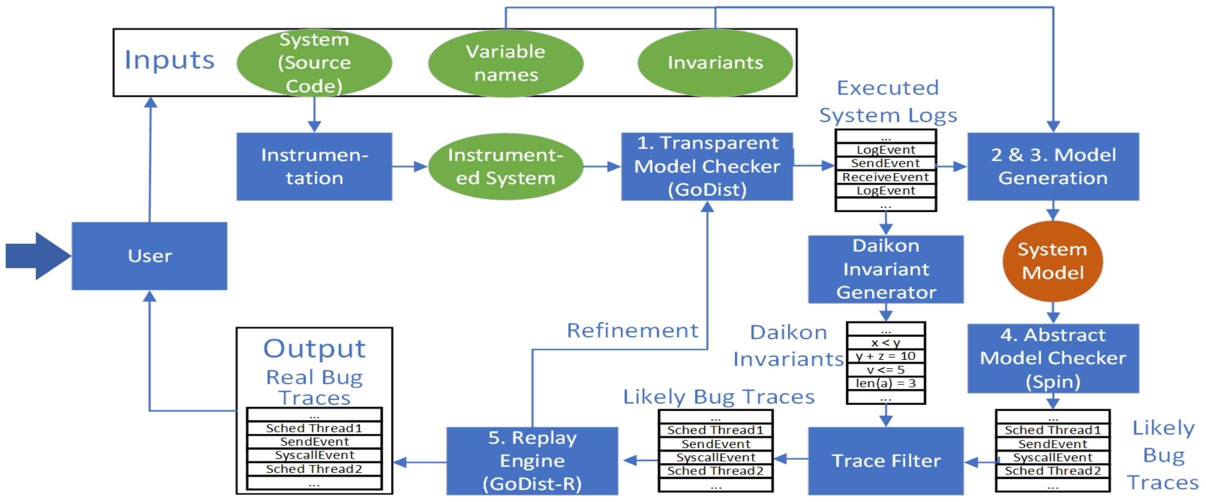

Figure 1: Dara pipeline: An instrumented system and its tests are used to produce state traces. A model is inferred from these traces and is coarsened using the user-provided variable IDs. SPIN checks the model for violations of user-provided invariants to produce likely bug traces which are then verified by a replay engine as real bug traces.

executions together for Finite State Machine (FSM) generation. Each state in the FSM corresponds to a unique combination of concrete variable values at a point in the execution of the program. State transitions (edges) are defined by the sent or received messages, or important local events (syscalls, timers). However, the FSM constructed from the logs is too large and suffers from state space explosion. Dara also uses Daikon [7] to infer invariants from these logs which are used later in the pipeline to rank likely bug traces.

3.Model Coarsening — Dara uses a coarsening abstraction procedure in which the model is collapsed by matching FSM states on a subset of variables which are specified by the user. By default all the variables are used, as this gives us a whole view of the system. However, performing state matching on a subset of the variables, allows Dara to mitigate the effect of variables like timestamps, buffers, IDs, and ports whose values have a low probability of matching. However, matching on a subset of state allows false positives in property checking. To filter out false positives, Dara uses a model accuracy heuristic which uses Daikon invariant violations.

4.Abstract Model Checking — The abstract model is converted to a Promela [13] model and provided as input to the SPIN model checker which generates error traces based on the provided safety and liveness invariants. These error traces are then verified against the previously inferred Daikon invariants for filtering. The key idea is that fewer the number of Daikon violations, it is more likely for that error trace to correspond to a real trace. The top n error traces with the fewest invaraint violations are passed to the replay engine to be verified as real bug traces.

5.Bug Trace Verification — The likely bug traces are passed to our replay engine to be verified as real traces to weed out false positive traces. The replay engine, GoDist-R, just like GoDist uses the Go runtime as an interposition layer between the application and OS. GoDist-R converts the the likely bug trace into a concrete trace that is schedulable by the Go runtime. If the engine fails to replay a trace, Dara reverts to system exploration by looping back to Log Generation phase with GoDist.

3 EVALUATION & FUTURE WORK

Our Dara prototype consists of 6K lines of Go. We used Dara on a 200 line implementation of Dining Philosophers and successfully identified fairness violations not present in existing logs.

We are actively developing Dara. Here we overview some of our ongoing work.

State Space Exploration We will equip Dara's transparent model checker with state-of-the-art efficient schedule exploration algorithms such as DPOR [8].

Extensive Evaluation We will evaluate Dara on real systems in production by checking important invariants. The candidate systems include ETCD [5], a distributed key-value store which uses Raft [21], and BTCD [4], an alternative full node Bitcoin implementation written in Go. We will build on our prior work in injecting bugs in ETCD [9] to find violations of strong leadership, log matching, and leadership agreement invariants.

4 CONCLUSION

We have introduced Dara, a hybrid model checker for distributed systems. Dara can be used to find *valid*, complex bugs *faster* as it combines the speed of abstract model checking with the correctness of transparent model checkers. We hope to demonstrate that Dara reduces the developer burden in verifying system correctness.

ACKNOWLEDGEMENT

This research is being performed with Stewart Grant under the supervision of Ivan Beschastnikh, and is sponsored by an NSERC Discovery Grant and a Work Learn International Undergraduate Research Award (WLIURA).

REFERENCES

[1] Cyrille Artho, Quentin Gros, Guillaume Rousset, Kazuaki Banzai, Lei Ma, Takashi Kitamura, Masami Hagiya, Yoshinori Tanabe, and Mitsuharu Yamamoto. 2017. Model-based API testing of Apache ZooKeeper. In *Software Testing, Verification and Validation (ICST), 2017 IEEE International Conference on*. IEEE, 288–298.

[2] Fletcher Babb. 2015. AmazonâĂŹs AWS DynamoDB Experiences Outage, Affecting Netflix, Reddit, Medium, and More. en-US. https://venturebeat.com/2015/09/20/amazons-aws-outage-takes-down-netflix-reddit-medium-and-more/.

[3] Ivan Beschastnikh, Yuriy Brun, Michael D Ernst, and Arvind Krishnamurthy. 2014. Inferring models of concurrent systems from logs of their behavior with CSight. In *Proceedings of the 36th International Conference on Software Engineering*. ACM, 468–479.

[4] btcsuite. 2013. an alternative full node bitcoin implementation. https://github.com/btcsuite/btcd.

[5] Coreos. 2013. Distributed reliable key-value store. https://github.com/coreos/etcd.

[6] Pantazis Deligiannis, Matt McCutchen, Paul Thomson, Shuo Chen, Alastair F Donaldson, John Erickson, Cheng Huang, Akash Lal, Rashmi Mudduluru, Shaz Qadeer, et al. 2016. Uncovering Bugs in Distributed Storage Systems during Testing (Not in Production!).. In *FAST*. 249–262.

[7] Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, and Chen Xiao. 2007. The Daikon system for dynamic detection of likely invariants. *Sci. Comput. Program.* (2007).

[8] Cormac Flanagan and Patrice Godefroid. 2005. Dynamic partial-order reduction for model checking software. In *ACM Sigplan Notices*, Vol. 40. ACM, 110–121.

[9] Stewart Grant, Hendrik Cech, and Ivan Beschastnikh. 2018. Inferring and Asserting Distributed System Invariants. In *Proceedings of the 40th International Conference on Software Engineering (ICSE '18)*. ACM, New York, NY, USA, 1149–1159. https://doi.org/10.1145/3180155.3180199

[10] Rachid Guerraoui and Maysam Yabandeh. 2011. Model Checking a Networked System Without the Network. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation (NSDI'11)*. USENIX Association, Berkeley, CA, USA, 225–238. http://dl.acm.org/citation.cfm?id=1972457.1972481

[11] Huayang Guo, Ming Wu, Lidong Zhou, Gang Hu, Junfeng Yang, and Lintao Zhang. 2011. Practical Software Model Checking via Dynamic Interface Reduction. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles (SOSP '11)*. ACM, New York, NY, USA, 265–278. https://doi.org/10.1145/2043556.2043582

[12] Chris Hawblitzel, Jon Howell, Manos Kapritsos, Jacob R. Lorch, Bryan Parno, Michael L. Roberts, Srinath Setty, and Brian Zill. [n. d.]. IronFleet: Proving Practical Distributed Systems Correct. In *SOSP 2015*. 17. https://doi.org/10.1145/2815400.2815428

[13] Gerard J. Holzmann. 1997. The Model Checker SPIN. *IEEE Trans. Softw. Eng.* 23, 5 (May 1997), 279–295. https://doi.org/10.1109/32.588521

[14] Charles Killian, James W. Anderson, Ranjit Jhala, and Amin Vahdat. 2007. Life, death, and the critical transition: finding liveness bugs in systems code. In *Networked Systems Design and Implementation (NSDI)*. Cambridge, MA, USA.

[15] Sandeep Kumar, Siau-Cheng Khoo, Abhik Roychoudhury, and David Lo. 2012. Inferring class level specifications for distributed systems. In *Proceedings of the 34th International Conference on Software Engineering*. IEEE Press, 914–924.

[16] François Laroussinie and Kim G Larsen. 1998. CMC: A tool for compositional model-checking of real-time systems. In *Formal Description Techniques and Protocol Specification, Testing and Verification*. Springer, 439–456.

[17] Tanakorn Leesatapornwongsa, Mingzhe Hao, Pallavi Joshi, Jeffrey F Lukman, and Haryadi S Gunawi. 2014. SAMC: Semantic-Aware Model Checking for Fast Discovery of Deep Bugs in Cloud Systems.. In *OSDI*. 399–414.

[18] Friedemann Mattern. 1989. Virtual Time and Global States of Distributed Systems. In *Parallel and Distributed Algorithms*. 215–226. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.47.7435

[19] Madanlal Musuvathi, Shaz Qadeer, Thomas Ball, Gerard Basler, Piramanayagam Arumuga Nainar, and Iulian Neamtiu. 2008. Finding and Reproducing Heisenbugs in Concurrent Programs.. In *OSDI*, Vol. 8. 267–280.

[20] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. 2015. How Amazon Web Services Uses Formal Methods. *Commun. ACM* 58, 4 (March 2015), 66–73. https://doi.org/10.1145/2699417

[21] Diego Ongaro and John K Ousterhout. 2014. In search of an understandable consensus algorithm.. In *USENIX Annual Technical Conference*. 305–319.

[22] Shannon Vavra. 2017. Amazon outage cost S&P 500 companies $150M. https://www.axios.com/amazon-outage-cost-sp-500-companies-150m-1513300728-aaff3a9e-d5de-4700-aded-55614cf7852c.html.

[23] James R. Wilcox, Doug Woos, Pavel Panchekha, Zachary Tatlock, Xi Wang, Michael D. Ernst, and Thomas Anderson. [n. d.]. Verdi: A Framework for Implementing and Formally Verifying Distributed Systems. In *PLDI 2015*. https://doi.org/10.1145/2737924.2737958

[24] Maysam Yabandeh, Nikola Knezevic, Dejan Kostic, and Viktor Kuncak. 2009. CrystalBall: Predicting and preventing inconsistencies in deployed distributed systems. In *The 6th USENIX Symposium on Networked Systems Design and Implementation (NSDIâĂŹ09)*.

[25] Junfeng Yang, Tisheng Chen, Ming Wu, Zhilei Xu, Xuezheng Liu, Haoxiang Lin, Mao Yang, Fan Long, Lintao Zhang, and Lidong Zhou. [n. d.]. MODIST: Transparent Model Checking of Unmodified Distributed Systems. In *NSDI 2009*. USENIX Association, 16. http://dl.acm.org/citation.cfm?id=1558977.1558992