Inferring likely data invariants of distributed systems

Stewart Grant[0], Sam Creed[0], Ivan Beschastnikh
Department of Computer Science, University of British Columbia
stewbertgrant@gmail.com, samcreed@cs.ubc.ca, bestchai@cs.ubc.ca

Overview

Distributed systems are difficult to debug and understand. One key reasons for this is distributed state, which is not easily accessible and must be pieced together from the state at the individual nodes. Developers have few tools to help them extract and reason about distributed state. By contrast, the state of a sequential program is well defined (e.g., stack and heap), easy to inspect (e.g., with breakpoints) and can be checked for correctness (e.g., by asserting invariants).

In this poster and demo[1] we describe a systems analysis tool called DInv[2]. DInv use a combination of static and dynamic analyses to identify and record at runtime the sets of concrete values for node variables that make up distributed state in a distributed system. It then determines consistent snapshots of distributed state in the system and uses the Daikon tool [2] to infer likely data relations, or invariants, over the tracked variables.

The inferred data invariants relate variables at different nodes in the system and can be used in a variety of ways. For example, they can improve developer understanding of their systems and can be used by test-case generation techniques to more effectively drive the system towards invalid states (i.e., states that violate a mined invariant). They can be also used for debugging: a mined invariant that violates the system's specifications can be tracked back to observed concrete values, which may help in reproducing an execution that led to an incorrect state of the system.

For example, consider a two phase commit protocol in which the coordinator first queries other nodes for their vote and if all nodes, including the coordinator, voted "Commit" then the coordinator broadcasts a "TX Commit", otherwise it broadcasts a "TX Abort". At the end of this protocol all nodes should either commit or abort. To check if the algorithm is correct, developers can examine the DInv-inferred distributed state invariants for a set of executions. In this case a developer can check whether DInv mines the invariant $coordinator.commit = replica_i.commit$ for each replica i in the system (i.e., commit state at all nodes should be identical).

The DInv tool is implemented in Go[3] and works on systems written in Go. We have applied DInv to a Go implementation of the Raft consensus algorithm [4] by HashiCorp[4]. With minimal instrumentation of the source code we are able to detect basic invariants about the system regarding whether or not multiple leaders are active at the same time and the number of active nodes in the system.

[0]Student authors. Ivan (faculty) will probably present the poster.
[1]We will demo a preliminary version of our tool on a laptop.
[2]Prototype at https://bitbucket.org/bestchai/dinv
[3]http://golang.org
[4]https://github.com/hashicorp/raft

Related work

Modeling, management, and capture of state in distributed systems has a long history [1, 6, 3]. However, the concrete state of a system is only useful to tools that can evaluate it against known properties of the system. To be more readily useful, distributed state needs a more usable, and more abstract, representation.

DInv tackles the problem of mining and abstracting distributed state. Other work in this domain focused on detecting dependencies, anomalies [7], and performance debugging [5]. However, this prior work considers events, and not the state of the system. The closest prior work in the sequential domain is the Daikon tool [2], which cannot be applied to distributed systems.

Yabandeh et al. [8] infer almost-invariants in distributed systems: invariants that are true in most cases and assume these invariants only are violated due to bugs. They require the user to provide a list of variables and functions for invariant inference. DInv infers distributed state variables automatically. Moreover, they assume that an external module generates a trace of globally consistent cuts with distributed state for their algorithm while our approach actually generates these consistent cuts.

1. REFERENCES

[1] K. M. Chandy and L. Lamport. Distributed snapshots: determining global states of distributed systems. *ACM Trans. Comput. Syst.*, 3(1):63–75, Feb. 1985.

[2] M. D. Ernst, J. Cockrell, W. G. Griswold, and D. Notkin. Dynamically Discovering Likely Program Invariants to Support Program Evolution. *IEEE Transactions on Software Engineering*, 27(2):99–123, Feb. 2001.

[3] D. Geels, G. Altekar, P. Maniatis, T. Roscoe, and I. Stoica. Friday: Global Comprehension for Distributed Replay. In *NSDI*, 2007.

[4] D. Ongaro and J. Ousterhout. In Search of an Understandable Consensus Algorithm. In *ATC*, 2014.

[5] R. R. Sambasivan, A. X. Zheng, M. De Rosa, E. Krevat, S. Whitman, M. Stroucken, W. Wang, L. Xu, and G. R. Ganger. Diagnosing Performance Changes by Comparing Request Flows. In *NSDI*, 2011.

[6] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Comput. Surv.*, 22(4):299–319, Dec. 1990.

[7] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan. Detecting Large-Scale System Problems by Mining Console Logs. In *SOSP*, 2009.

[8] M. Yabandeh, A. Anand, M. Canini, and D. Kostic. Finding Almost-Invariants in Distributed Systems. In *SRDS*, 2011.