Coracle: Evaluating Consensus at the Internet Edge

Heidi Howard
University of Cambridge
heidi.howard@cl.cam.ac.uk

Jon Crowcroft
University of Cambridge
jon.crowcroft@cl.cam.ac.uk

ABSTRACT

Distributed consensus is fundamental in distributed systems for achieving fault-tolerance. The Paxos algorithm has long dominated this domain, although it has been recently challenged by algorithms such as Raft and Viewstamped Replication Revisited.

These algorithms rely on Paxos's original assumptions, unfortunately these assumptions are now at odds with the reality of the modern internet. Our insight is that current consensus algorithms have significant availability issues when deployed outside the well defined context of the datacenter.

To illustrate this problem, we developed Coracle, a tool for evaluating distributed consensus algorithms in settings that more accurately represent realistic deployments. We have used Coracle to test two examples of network configurations that contradict the liveness claims of the Raft algorithm. Through the process of exercising these algorithms under more realistic assumptions, we demonstrate wider availability issues faced by consensus algorithms when deployed on real world networks.

CCS Concepts

•**Computer systems organization** → **Availability;** *Redundancy;*

Keywords

Distributed consensus; Fault-tolerance; Dependable systems

1. INTRODUCTION

Modern distributed systems depend on the centralised cloud rather than deal with the complexity of the internet edge. In the wake of censorship concerns, mass-surveillance and data breaches, users are demanding viable alternatives to third-party centralised systems. Building distributed systems across hosts on the internet edge is one such alternative. Such systems have the potential to provide low latency services and the ability to operate without a full internet connection. Ensuring consensus algorithms are resilient to common failures at the internet edge is vital for building reliable systems.

Paxos [2] has been synonymous with distributed consensus for over a decade. This domain has been recently challenged by algorithms such as Raft [4] and Viewstamped Replication Revisited [3]. These algorithms (like many others) are difficult to use in practice as they are based on Lamport's original model of the internet, which is at odds with the current reality, particularly at the internet edge.

Paxos family consensus algorithms assume homogeneous, static hosts on a fixed, fully-connected network. Deployment of these algorithms requires knowledgeable sysadmins with accurate understanding of network properties and failures are assumed to be rare.

In reality, consensus algorithms are deployed beyond the datacenter. Here, we have heterogeneous hosts with various resource constraints, managed by everyday people. They are on mobile networks with unpredictable link characteristics and poorly understood middleboxes. Networks and hosts change over time and a diverse range of failures are commonplace.

We could try to modify the consensus algorithms to mitigate these issues but these changes may in turn introduce other issues or violate correctness.

Furthermore, consensus algorithms commonly depend on the correct selection of various parameters (such as timeouts for failure detection) to reach a stable state. Choosing such values is challenging enough in a datacenter context, let alone in a dynamic environment such as the internet edge.

2. EXAMPLES

Raft claims that consensus algorithms are fully functional as long as any majority of the hosts are operational and can communicate with each other and with clients.

In the two examples that follow, the majority of hosts are live and able to communicate yet the system is unavailable as the leadership algorithm is non-convergent. Although these issues are common among distributed consensus algorithms, we have chosen Raft to illustrate this point due to its popularity and understandability.

In Raft, each host stores its current mode, either *follower*, *candidate* or *leader*, and its current term, a monotonically increasing value used to order events. Nodes begin as *followers* and receive regular heartbeats from the *leader*. If a *follower* fails to hear a valid *leader's* heartbeat then it will increment its term and become a *candidate*.

SIGCOMM '15 August 17-21, 2015, London, United Kingdom

ACM ISBN 978-1-4503-3542-3/15/08.

DOI: http://dx.doi.org/10.1145/2785956.2790010

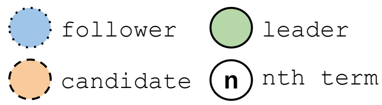

Figure 1: Legend for Figures 2 and 3

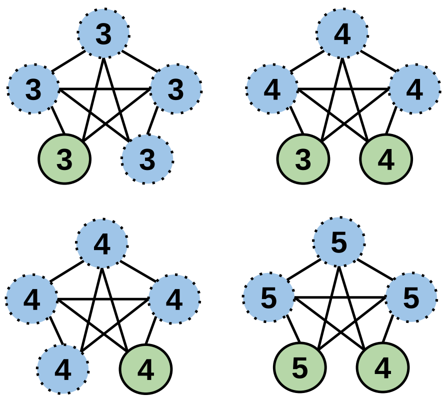

Figure 2: Four snapshots in time of Raft running across five hosts with non-transitive reachability

This host will ask all the other hosts to vote for it, including its current term in that request. A host will vote for the *candidate* if its term is the same or higher than its own and it has not yet voted in that term[1]. If the *candidate* receives votes from a strict majority of hosts, then it becomes *leader* in that term. Otherwise, it increments its term and restarts the election. If a host hears from another host with a higher term, it will step down to a *follower* in that new higher term.

Figure 2 shows the reachability between 5 hosts, all hosts are connected except the two at the bottom. Here we see leadership bouncing between the two disconnected hosts because if one of these hosts is the *leader* then the other is unable to hear the *leader's* heartbeat. This causes the host to timeout, increment its term and start a new election. This host will likely be elected *leader*. The old *leader* will step down to a *follower* when it hears about this from another host. The old *leader* will soon timeout and start a new election as it cannot hear from the new *leader*.

Figure 3 shows the reachability between 4 hosts, one of which is behind a poorly configured NAT, allowing it to transmit messages to all hosts but not hear responses. This host will never be able to hear the *leader's* heartbeats and thus will continually timeout, increment its term and start a new election. When contacting the other hosts, it will remove the current *leader* and continuously force *leader* election, rendering the system unavaliable.

3. APPROACH

We have informally argued that consensus algorithms cannot tolerate many common faults at the internet edge, but it is difficult to systematically study and thus address these issues, given the vast state space and complex experimental setups required.

[1]Raft includes extra conditions of voting, though these details are not included for simplicity.

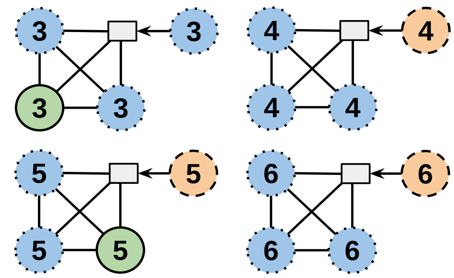

Figure 3: Four snapshots in time of Raft running across four hosts with asymmetric reachability

In response, we developed Coracle, an event-based simulator for distributed consensus algorithms on the internet edge. Building on our earlier work on reproducing Raft [1], we utilize state machine replication [5] as a common intermediate abstraction for consensus protocols like Raft and Viewstamped Replication Revisited. We provide a graphical front-end for selecting from our test suite of network environments (like the examples in Figures 2 & 3) or users can design their own network. In contrast to traditional network simulators, our pure protocol implementations allow Coracle to store a complete trace of all local state and communication. This allows it to rerun traces, run traces though a trace checker to check safety guarantees, and extract performance metrics. Coracle enables users to investigate the best algorithm parameters for a particular deployment. Likewise, users can model modifications to an existing consensus algorithm (or implement a new algorithm altogether) and evaluate it against our test suite of internet edge environments.

Acknowledgements

We appreciate constructive feedback on this abstract from Matthew Huxtable, Matthew Grosvenor, Mindy Preston, Frank McSherry, Richard Mortier and Helen Oliver. Thanks to the generous support from Microsoft.

4. REFERENCES

[1] H. Howard, M. Schwarzkopf, A. Madhavapeddy, and J. Crowcroft. Raft refloated: Do we have consensus? *ACM SIGOPS Operating Systems Review*, 49(1):12–21, 2015.

[2] L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems (TOCS)*, 16(2):133–169, 1998.

[3] B. Liskov and J. Cowling. Viewstamped replication revisited. 2012.

[4] D. Ongaro and J. Ousterhout. In search of an understandable consensus algorithm. In *USENIX Annual Technical Conference*.

[5] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.