2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS) | 978-1-7281-7002-2/20/$31.00 ©2020 IEEE | DOI: 10.1109/ICDCS47774.2020.00137

A Hierarchical Model for Fast Distributed Consensus in Dynamic Networks

Timothy Castiglia, Colin Goldberg, and Stacy Patterson

I. Introduction

State machine replication is a foundational tool that is used to provide availability and fault tolerance in distributed systems. Safe replication requires a consensus algorithm as a method to achieve agreement on the order of system updates. Designing algorithms that are safe while retaining high throughput is imperative for today's systems. Two of the most widely adopted consensus algorithms, Paxos [1] and Raft [2], have received much attention and use in industry. While both Paxos and Raft provide safe specification for maintaining a replicated log of system updates, Raft aims for ease of understandability and implementation.

Modern distributed systems are often large-scale, globally distributed, and dynamic. Both Paxos and Raft require several rounds of messaging between a leader and a majority of sites, and message latency in a global system is too high to support these message rounds while maintaining high throughput. To mitigate this problem, variations on Paxos, such as Fast Paxos [3], reduce the number of message rounds if certain conditions, such as non-concurrent proposals, are met. These algorithms will suffer additional rounds if conditions are not met. To address the dynamic nature of networks, Raft [4] and Vertical Paxos [5] provide protocols for membership reconfiguration. However, both rely on a system administrator to ensure safe reconfiguration. In many distributed systems, membership changes may be sudden, and may occur silently.

To address the limitations of previous works, we design Fast Raft. Fast Raft reduces message rounds similar to Fast Paxos, while maintaining Raft's understandability. Fast Raft also handles dynamic membership without the need for an external administrator. However, reducing message rounds is often not enough to achieve high throughput in systems with high message latency. Here, a hierarchical model where sites are grouped into low latency clusters is beneficial. The bulk of the computation is performed within each cluster, with results combined globally at a lower frequency. For such a model, we present C-Raft, a new algorithm that provides multi-level consensus and improves throughput of replication in globally distributed systems. We provide experimental results of the performance of Fast Raft and C-Raft against classic Raft. We find Fast Raft on average is twice as fast as classic Raft when message loss is below 5%. C-Raft achieves up to a 5x throughput increase over Raft in a globally distributed system. For more details on the algorithms, see our technical report [6].

T. Castiglia, C. Goldberg, and S. Patterson are with the Department of Computer Science, Rensselaer Polytechnic Institute, 110 8th St, Troy, NY 12180, castit@rpi.edu, goldbc@rpi.edu, sep@cs.rpi.edu. This work was supported in part by NSF grants CNS-1553340 and CNS-1816307.

II. Fast Raft

In classic Raft, time is split into *terms* numbered in a monotonically increasing manner. Sites take on one or more roles in a term. In a typical term, one site is elected as the *leader*. *Proposers* propose new entries to the leader. The leader gathers proposals and appends the entries to its log. The leader sends the entries to the *followers*, who append the entries and send back an acknowledgement. Once the leader receives a majority of acknowledgements, the entry is considered committed, taking a total of 3 message rounds.

In Fast Raft it takes 2 message rounds to commit entries when proposals are non-concurrent. Proposers send new entries to followers first, then followers send proposal votes to the leader. The leader gathers proposal votes for each log index, and determines if an entry can be committed early. This *fast track* can be taken if enough followers, a *fast quorum*, has voted for it. Otherwise, the leader reverts to classic Raft, the *slow track*.

With the addition of the fast track, Raft's leader election must be modified to maintain safety. In classic Raft, heartbeat messages are used to determine if a leader has failed. If a follower suspects the leader of failing, it will increment the term number and request election votes from other sites. A site will only vote for the follower if the follower's log is up-to-date, ensuring leader-completeness. In Fast Raft, the definition of up-to-date is modified to only include slow track entries to maintain safety. Once the most up-to-date candidate is elected, Fast Raft runs a recovery algorithm to evaluate whether it is safe to commit fast track entries.

To support dynamic membership, Raft defines a configuration log entry that contains a list of all sites taking part in consensus. Classic Raft assumes a system administrator proposes configuration changes to the leader, and that only one site is added or removed from the configuration at a time to ensure safety. In Fast Raft, sites send join or leave requests to the leader. It is the role of the leader to ensure that only one site joins at a time. Unlike classic Raft, the Fast Raft algorithm supports sites leaving the system silently, as sites may not propose a leave request before leaving the system.

III. C-Raft

The goal of C-Raft is to improve the throughput of consensus in systems with high message latency between distant sites. In our system model, sites form a set of clusters whose

2575-8411/20/$31.00 ©2020 IEEE
DOI 10.1109/ICDCS47774.2020.00137

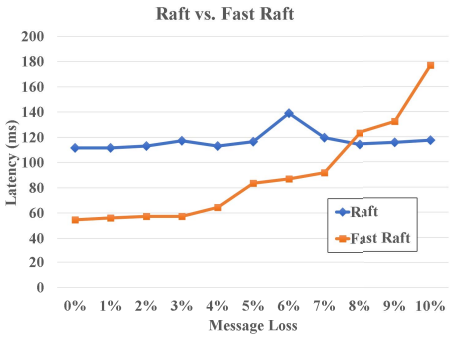

Fig. 1: Average latency of committing entries in Raft and Fast Raft with different percentages of message loss.

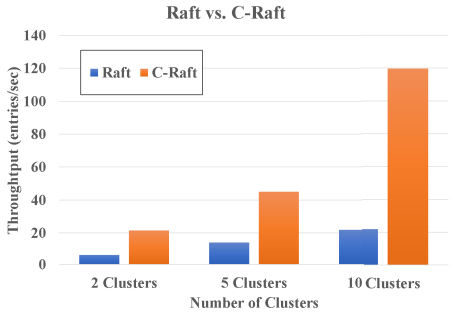

Fig. 2: Average throughput of 20 sites in classic Raft and C-Raft. Each cluster is in a different AWS region.

number and membership can change over time. We separate the means of communication within clusters, *intra-cluster*, and across clusters, *inter-cluster*. We assume that intra-cluster communication is lower latency than inter-cluster.

Within each cluster, sites commit entries using Fast Raft to a *local log* of that cluster. Periodically, a cluster leader proposes a batch of local entries to other cluster leaders in a global level of Fast Raft. These entries are committed to a *global log* which maintains the total order of entries over all sites and clusters. The hierarchical structure allows proposers to have their entries replicated locally, with the guarantee that the entries will eventually be replicated to other clusters and totally-ordered in the global log. C-Raft utilizes this model to improve throughput of consensus at the global scale.

During inter-cluster consensus, it is possible for a cluster leader to fail and a new one be elected. The local log of the cluster is used here to ensure a leader's state for the global log is passed on to new leaders. This is achieved through intra-cluster consensus on a *global state entry* for inter-cluster state replication.

IV. Experiments

To showcase the performance of Fast Raft and C-Raft, we performed experiments on Amazon Web Services (AWS). To simulate clusters, instances were started in different regions around the world. Sites in the same AWS region formed a cluster. We implemented classic Raft, Fast Raft, and C-Raft and compared their performance.

Classic Raft vs. Fast Raft: First, we compared the commit latency of classic Raft and Fast Raft in a single cluster. We chose a single site at random to be the proposer, and measured the average latency for entries committed over 100 trials when using classic Raft and Fast Raft. In the experiments, we had five sites in one region and varied the message loss between 0% and 10%.

Figure 1 shows the results of the experiment. When message loss is low, Fast Raft achieves about half the latency as classic Raft. However, as message loss increased, Fast Raft started to degrade in performance while classic Raft maintained similar latency. As more messages are dropped, the classic-track is used more in Fast Raft, causing it to face an extra message round more often.

Classic Raft vs. C-Raft: We compared the throughput of Raft and C-Raft. We chose one proposer at random per cluster. Each proposer waited until its last proposed entry was committed before proposing another. We compared throughput based on how many entries were committed to the global log in classic Raft and C-Raft, averaged over five 3-minute trials.

For the C-Raft implementation, a cluster proposes a batch of entries to the global log after ten entries are committed in the local log. We tested with 20 sites total, split evenly over varying number of clusters. Note, there were more proposers in the system as the number of clusters increased for both C-Raft and Raft. The results are shown in Figure 2. C-Raft shows significant improvements over classic Raft, with a $5x$ throughput increase for 10 clusters.

V. Takeaway

Fast Raft is a variation on the Raft consensus algorithm that speeds up consensus in typical operation, and C-Raft defines a hierarchical model of Fast Raft consensus. Both algorithms deal with membership changes in dynamic networks. Our experiments show that Fast Raft can achieve half the latency of classic Raft when message loss is low, and C-Raft can achieve a $5x$ throughput improvement over classic Raft in a globally distributed scenario. In future work, we plan to explore extending Fast Raft to support partially-ordered log entries, similar to Generalized Paxos [7].

References

[1] L. Lamport, "Paxos made simple," *ACM Sigact News*, vol. 32, no. 4, pp. 18–25, 2001.
[2] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *2014 USENIX Annual Technical Conference*, 2014.
[3] L. Lamport, "Fast paxos," *Distributed Computing*, vol. 19, no. 2, pp. 79–103, 2006.
[4] D. Ongaro, "Consensus: Bridging theory and practice," Ph.D. dissertation, Stanford University, 2014.
[5] L. Lamport, D. Malkhi, and L. Zhou, "Vertical paxos and primary-backup replication," in *Proc. 28th ACM Symp. on Principles of Distributed Computing*, 2009, pp. 312–313.
[6] T. Castiglia, C. Goldberg, and S. Patterson, "A hierarchical model for fast distributed consensus in dynamic networks," *arXiv preprint arXiv:2004.06215*, 2020.
[7] L. Lamport, "Generalized consensus and paxos," *Technical Report MSR-TR-2005-33, Microsoft Research*, 2005.