Improved Majority Quorums for Raft

Ensar Basri Kahveci   Fatma Kahveci

2020-07-27 (Revision 1)

Abstract

Raft has gained widespread adoption in the industry as a newcomer, thanks to its pragmatic approach to the distributed consensus problem. Also, its resemblance to Paxos allows it to benefit from decades of research on Paxos. In this work, we study simple quorums and improved majority quorums of FPaxos in the context of Raft consensus algorithm. Even-sized clusters are prone to a greater risk of unavailability compared to odd-sized clusters because of the majority quorum requirement. Our main finding, which applies equally to Raft and Multi-Paxos, shows that even-sized clusters can employ improved majority quorums to obtain the same degree of availability as odd-sized ones with one less server. We also use improved majority quorums to simplify Raft cluster membership changes. Our solution minimizes availability gaps during cluster membership changes without any extra phase or voting state. We expect existing Raft implementations to utilize our findings with quite small code changes.

1 Introduction

Consensus algorithms play a key role in building large-scale and reliable distributed systems. Paxos [1, 2] and Raft [3] are two such algorithms. They are used to replicate the data across multiple servers and access it in a highly available and strongly consistent manner. They basically work in two phases. First, they elect a leader among servers. The leader takes charge of the system for the second phase, log replication. In this phase, the leader replicates operations to other servers and decides when it is safe to commit them. Paxos and Raft guarantee that an operation committed by a leader is never overwritten by another operation or a future leader. To make this guarantee work, they make use of majority quorums where both leader elections and replication of operations involve more than half of the servers.

Paxos is the traditional and probably the best-studied consensus algorithm. For instance, Flexible Paxos (FPaxos) [4] weakens the *"all quorums must intersect"* requirement for Paxos. According to FPaxos, it is enough that every leader election (phase 1) quorum intersects with every log replication (phase 2) quorum.

Raft, as a newcomer, has gradually overtaken Paxos in the industry. The resemblance between Raft and Paxos creates several opportunities for porting optimizations of Paxos to Raft. The observations of FPaxos are proven to be applicable to Raft [5].

In this work we study the application of the simple quorums and improved majority quorums settings of FPaxos to Raft clusters. There are two contributions:

1. Even-sized clusters have so far been considered at a disadvantage compared to odd-sized ones because they are less available. The finding in this paper, which applies equally to Raft and Multi-Paxos [2, 6], removes that disadvantage through the application of the improved majority quorum setting of FPaxos, and sets the even-sized cluster on an equal footing with the odd-sized cluster with one less server.
2. We simplify the transition of a running Raft cluster to a larger size. Our cluster membership change solution uses improved majority quorums to minimize availability gaps without any extra phase or voting state.

We expect existing Raft implementations to be able to make use of our findings with quite small code changes.

2 Weakening the Quorum Requirement

The main observation of FPaxos enables a number of different quorum settings. One example is *simple quorums*. We can write down the simple quorums setting as $|Q_{LE}| + |Q_{LR}| > N$, where Q_{LE}, Q_{LR} are leader election and log replication quorums respectively, and N is the number of servers. When they are both $\lfloor N/2 \rfloor + 1$, FPaxos is equivalent to Paxos.

Log replication is typically more costly and frequent than leader election. Therefore we can shrink Q_{LR} to reduce latency and improve throughput of log replication. For instance, in a 5-server setup, a Paxos leader can replicate and commit new operations with another server with $|Q_{LR}| = 2$. Log replication continues as long as the leader remains up, even when 3 servers fail. The price we pay is the increased risk of unavailability in case we need a new leader. Since $|Q_{LE}| = 4$ for this setup, if a leader fails along with another server, a new leader cannot be elected until one of the failed servers recovers.

The improved majority quorums setting is a specialization of the *simple quorums* setting. It improves performance of even-sized clusters without hurting availability. In this setting, we can reduce $|Q_{LR}|$ by one and keep $|Q_{LR}|$ the same. For instance, we can have $|Q_{LE}| = 3$ and $|Q_{LR}| = 2$ for a 4-server cluster. Leader elections still require majority quorums, but once a leader is elected, it no longer needs a majority quorum to commit log entries.

3 Simple Quorums for Raft

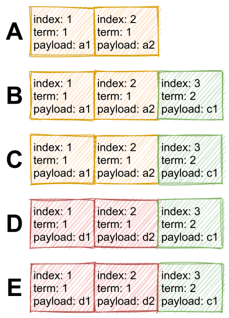

Figure 1: Raft logs of 5 Raft servers with $|Q_{LE}| = 2$ and $|Q_{LR}| = 4$

Raft's safety properties require that a server can become a leader only if it has all committed log entries and gets the majority votes. Otherwise, as shown in Figure 1, multiple Raft leaders can be elected in a single term and those leaders can append log entries with different payloads to the same indices (i.e., split-brain). This breaks Raft's *log matching property* and causes conflicting log entries to be committed by a leader of a higher term. To prevent this problem, Raft requires that $|Q_{LE}| \geq \lfloor N/2 \rfloor + 1$ always holds.

If $|Q_{LE}|$ becomes greater than $\lfloor N/2 \rfloor + 1$ and $|Q_{LR}|$ is decreased while maintaining $|Q_{LE}| + |Q_{LR}| > N$, Raft leader elections still guarantee that a leader contains all committed log entries. However, a Raft leader sends log entries to all servers. It means that log entries will be eventually replicated to the whole cluster even though they are committed with a minority quorum. Therefore we should carefully consider the tradeoffs, such as replica and network capacities, and failure rates, to utilize this optimization in a Raft cluster. We expect majority quorums to be the optimal setting for most Raft deployments when there is no significant difference between replica and network capacities.

4 Improved Majority Quorums for Raft

Raft clusters typically consist of an odd number of servers. It is because an even-sized cluster tolerates the same number of failures as the odd-sized cluster with one less server. However, the risk of unavailability of the former one is greater than the latter one. To improve the degree of availability of an odd-sized Raft cluster, 2 more servers must be added. In this regard, an even-sized Raft cluster is usually a temporary state, with a greater risk of unavailability, while the cluster is being extended. We shall now show that, with improved majority quorums, this is no longer true.

4.1 Improving the Availability of Even-Sized Clusters

Here we first analyze the probability of the loss of availability for Raft clusters. Then, we show how improved majority quorums enable even-sized Raft clusters to have the same degree of availability as odd-sized clusters with one less server.

We assume that servers fail independently from each other with the same failure probability. Servers do not act maliciously (i.e., no Byzantine failures) and a failed server stops executing the algorithm. Then the probability of unavailability of a Raft cluster can be written as follows:

(1)

$$Pr(unavailability) = \sum_{k=\lceil \frac{N}{2} \rceil}^{N} \binom{N}{k} {p_f}^{k} (1-p_f)^{N-k}$$

where N is the number of servers, k is the number of failed servers, and p_f is the probability of a single server failure.

For instance, both 3- and 4-server Raft clusters become unavailable in case of failure of 2 or more servers. If we apply Equation (1) to clusters of 4 and 3 servers where $p_f = 0.20$, the probability of unavailability will be ~ 0.18 and ~ 0.10, respectively. Namely, a 4-server Raft cluster is more prone to unavailability compared to a 3-server Raft cluster because of the majority quorum requirement. $p_f = 0.20$ has no special meaning for this example and we could choose any probability value.

Improved majority quorums reduce the risk of unavailability for even-sized clusters. It is because when $N/2$ servers fail, the loss of availability depends on whether the leader is among the failed servers. The leader is absent in exactly half of the combinations of size $N/2$. Suppose that 2 servers out of *[A, B, C, D]* fail when A is the leader. The failure combinations are *(A, B)*, *(A, C)*, *(A, D)*, *(B, C)*, *(B, D)*, and *(C, D)*. With improved majority quorums, the cluster can still make progress during the failures of *(B, C)*, *(B, D)*, and *(C, D)*. Thus, we can update our equation for even-sized clusters to remove those combinations.

When N is even and improved majority quorums are in place, we can write down the probability of unavailability as follows:

(2)

$$Pr(unavailability) = \left(\sum_{k=\frac{N}{2}+1}^{N} \binom{N}{k} {p_f}^{k} (1-p_f)^{N-k} \right) + \frac{1}{2} \binom{N}{\frac{N}{2}} {p_f}^{\frac{N}{2}} (1-p_f)^{\frac{N}{2}}$$

If we apply Equation (2) to a 4-server Raft cluster for $p_f = 0.20$, the probability of unavailability is reduced to ~ 0.10.

We realize that, if we use Equation (2) for an even-sized Raft cluster and Equation (1) for an odd-sized Raft cluster with one less server, both clusters obtain the

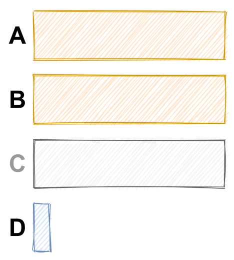

Figure 2: C crashes before D catches up with the leader

same degree of availability. In other words, improved majority quorums enable even-sized Raft clusters to have the same degree of availability as odd-sized clusters with one less server. We provide proof of this equation in the Appendix.

This realization also applies to Multi-Paxos when proposers and acceptors are colocated.

Thanks to this improvement, even-sized Raft clusters become more useful. First, they don't increase the risk of unavailability while clusters are being extended for better resilience. We rely on this realization to build an alternative cluster membership change solution for Raft in the next section. Second, even-sized Raft clusters can be used when network capacities between servers differ. Consider a scenario where an even-sized Raft cluster is homogeneously deployed into 2 availability zones. A leader performs log replication with the followers in its availability zone without paying the latency cost between the availability zones. In the meantime, the followers on the other availability zone serve can read requests with relaxed consistency guarantees. The leader continues log replication when the other availability zone fails or the network connection between the availability zones drops. If multiple Raft clusters are deployed, leaders can be balanced between availability zones to handle workloads more efficiently. The leadership transfer capability of Raft [7] can be used for this.

4.2 Cluster Membership Changes

Raft implementations usually realize cluster membership changes by adding or removing one server at a time [7]. For instance, in order to extend a 3-server Raft cluster to 5 servers to improve its resilience, we first add the 4th server. At this point, the majority quorum size goes up from 2 to 3. However, a new server typically starts with an empty state, and it could take quite a while until it catches up with the leader. If one of the original servers fails during this process, we cannot commit any new log entry. In short, we can encounter availability gaps if minority failures occur before a new server catches up in an even-sized cluster. Figure 2 depicts this scenario.

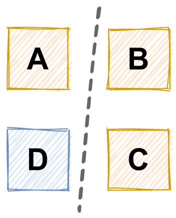

Figure 3: D is joining while there is a network partition

To prevent this problem, Raft introduces an additional phase to cluster membership changes. A new server joins the cluster as a non-voting member, hence the majority quorum size does not change. In this phase, the leader replicates log entries to the new server, but excludes it from majority quorum calculations. Once the new server catches up with the leader, the leader proceeds the membership change process, and the new server turns into a voting member.

We use improved majority quorums to simplify Raft cluster membership changes. In our solution, a new server joins the cluster directly as a voting member, and leader election and log replication quorum sizes are updated based on improved majority quorums. In the example above, after the 4^{th} server joins the cluster, $|Q_{LE}|$ is incremented to 3, but $|Q_{LR}|$ is still 2. As a consequence, the cluster remains available if one of the original servers fails before the new server has caught up with the leader. Once the 4^{th} server completes the sync process, the 5^{th} server joins, and both $|Q_{LE}|$ and $|Q_{LR}|$ becomes 3.

There is a subtle point in this solution. Recall that in Raft, a new member list becomes effective when it is appended to the Raft log and committed via the regular log replication mechanism. However, when using improved majority quorums, the cluster membership log entry becomes a special case and must be committed using the leader election quorum. In Figure 3, we add a new server while there is a network partition in the cluster. Suppose A is the leader and the current Raft log is not very long. If A uses $|Q_{LR}| = 2$ to commit this cluster membership change, it can perform the commit with D, then continue to commit new log entries with the same log replication quorum size. In the meantime, B and C still have the original member list, so they have $|Q_{LE}| = 2$. Once they notice that A is not reachable, they can elect a new leader among themselves and commit log entries independently from A and D.

To prevent this split-brain situation, the log entry indicating the cluster membership change must be committed using the majority quorum of the new cluster. After this commit, log replication can continue with log replication quorum of the original cluster.

5 Implementation Advice

Raft implementations can make use of the improved majority quorums setting with a minimal amount of code change. The only required change is to split the quorum variable into two variables, one for leader election and the second one for log replication:

- The leader election quorum variable is the same as the majority quorum size, that is $\lfloor N/2 \rfloor + 1$.
- The log replication quorum variable is $N/2$ if the currently effective cluster member list is committed and consists of an even number of servers. Otherwise, it is the same as the leader election quorum.

6 Conclusion

In this work, we show that the improved majority quorums setting of FPaxos enables even-sized Raft clusters to obtain the same degree of availability as odd sized clusters with one less server. This finding applies to Multi-Paxos as well. We also simplify Raft cluster membership changes with improved majority quorums. Our cluster membership solution minimizes availability gaps without any extra phase or voting state. Existing Raft implementations can make use of our findings with quite small code changes.

7 Acknowledgements

I worked on this project in my free time. It is not affiliated, associated, endorsed by, or in any way officially connected with my current employer Facebook, or any of its subsidiaries or its affiliates.

My dear wife, Fatma Kahveci, contributed the proof shown in the Appendix.

I would like to thank to my friends, Mehmet Dogan and Milan Savic for reviewing multiple drafts of this paper, and Marko Topolnik for his valuable feedback to improve the readability of the paper.

All errors belong to me.

8 References

- [1] Leslie Lamport. “The part-time parliament.” ACM Trans. Comput. Syst., 16(2):133–169, 1998.

- [2] Leslie Lamport. “Paxos made simple”. ACM SIGACT News (Distributed Computing Column), 2001.
- [3] Diego Ongaro and John Ousterhout. “In search of an understandable consensus algorithm.” USENIX Annual Technical Conference, 2014.
- [4] Heidi Howard, Dahlia Malkhi, Alexander Spiegelman.“Flexible Paxos: Quorum intersection revisited.” 20th International Conference on Principles of Distributed Systems (OPODIS), 2016.
- [5] https://github.com/fpaxos/raft.tla (Latest access: July, 2020)
- [6] Van Renesse, Robbert, and Deniz Altinbuken. “Paxos made moderately complex.” ACM Computing Surveys (CSUR) 47.3 (2015): 1-36.
- [7] Diego Ongaro. “Consensus: Bridging theory and practice”. Diss. Stanford University, 2014.

9 Appendix

Theorem: For all $0 < k \leq N$, where $N \in \mathbb{Z}^+$ is odd and p_f is a probability,

$$\sum_{k=\frac{N+1}{2}}^{N} \binom{N}{k} {p_f}^k (1-p_f)^{N-k} = \sum_{k=\frac{N+3}{2}}^{N+1} \left[\binom{N+1}{k} {p_f}^k (1-p_f)^{N+1-k}\right] + \frac{1}{2}\binom{N+1}{\frac{N+1}{2}} {p_f}^{\frac{N+1}{2}} (1-p_f)^{\frac{N+1}{2}}$$

Proof:

Both sides represent the probability of unavailability in case of server failures. N, k, and p_f are the number of servers, the number of failed servers, and the probability of a single server failure, respectively.

$$\sum_{k=\frac{N+3}{2}}^{N+1} \left[\binom{N+1}{k} {p_f}^k (1-p_f)^{N+1-k}\right] + \frac{1}{2}\binom{N+1}{\frac{N+1}{2}} {p_f}^{\frac{N+1}{2}} (1-p_f)^{\frac{N+1}{2}}, \text{where } (N+1) \in \mathbb{Z}^+ \text{ is even.}$$

$$= \left[\binom{N+1}{N+1} {p_f}^{N+1} (1-p_f)^0 + \sum_{k=\frac{N+3}{2}}^{N} \binom{N+1}{k} {p_f}^k (1-p_f)^{N+1-k}\right] + \frac{1}{2}\binom{N+1}{\frac{N+1}{2}} {p_f}^{\frac{N+1}{2}} (1-p_f)^{\frac{N+1}{2}}$$

$$= {p_f}^{N+1} + \sum_{k=\frac{N+3}{2}}^{N} \binom{N+1}{k} {p_f}^k (1-p_f)^{N+1-k} + \frac{1}{2} 2 \binom{N}{\frac{N+1}{2}} {p_f}^{\frac{N+1}{2}} (1-p_f)^{\frac{N+1}{2}} \;**$$

$$= {p_f}^{N+1} + \sum_{k=\frac{N+3}{2}}^{N} \left[\left[\binom{N}{k} + \binom{N}{k-1}\right] {p_f}^k (1-p_f)^{N+1-k}\right] + \binom{N}{\frac{N+1}{2}} {p_f}^{\frac{N+1}{2}} (1-p_f)^{\frac{N+1}{2}} \;*$$

$$= {p_f}^{N+1} + \sum_{k=\frac{N+3}{2}}^{N} \binom{N}{k} {p_f}^{k}(1-p_f)^{N+1-k} + \sum_{k=\frac{N+3}{2}}^{N} \binom{N}{k-1} {p_f}^{k}(1-p_f)^{N+1-k} + \binom{N}{\frac{N+1}{2}} {p_f}^{\frac{N+1}{2}}(1-p_f)^{\frac{N+1}{2}}$$

$$= {p_f}^{N+1} + \left[\sum_{k=\frac{N+3}{2}}^{N} \binom{N}{k} {p_f}^{k}(1-p_f)^{N+1-k} + \binom{N}{\frac{N+1}{2}} {p_f}^{\frac{N+1}{2}}(1-p_f)^{\frac{N+1}{2}} \right] + \sum_{k=\frac{N+3}{2}}^{N} \binom{N}{k-1} {p_f}^{k}(1-p_f)^{N+1-k}$$

$$= {p_f}^{N+1} + \sum_{k=\frac{N+1}{2}}^{N} \binom{N}{k} {p_f}^{k}(1-p_f)^{N+1-k} + \sum_{k=\frac{N+3}{2}}^{N} \binom{N}{k-1} {p_f}^{k}(1-p_f)^{N+1-k}$$

$$= {p_f}^{N+1} + \sum_{k=\frac{N+1}{2}}^{N} \binom{N}{k} {p_f}^{k}(1-p_f)^{N+1-k} + \sum_{k=\frac{N+1}{2}}^{N-1} \binom{N}{k} {p_f}^{k+1}(1-p_f)^{N-k}$$

$$= {p_f}^{N+1} + \sum_{k=\frac{N+1}{2}}^{N} \binom{N}{k} {p_f}^{k}(1-p_f)^{N+1-k} + \left[\sum_{k=\frac{N+1}{2}}^{N} \binom{N}{k} {p_f}^{k+1}(1-p_f)^{N-k} - \binom{N}{N} {p_f}^{N+1}(1-p_f)^{N-N} \right]$$

$$= {p_f}^{N+1} + \sum_{k=\frac{N+1}{2}}^{N} \binom{N}{k} {p_f}^{k}(1-p_f)^{N+1-k} + \sum_{k=\frac{N+1}{2}}^{N} \binom{N}{k} {p_f}^{k+1}(1-p_f)^{N-k} - {p_f}^{N+1}$$

$$= (1-p_f) \sum_{k=\frac{N+1}{2}}^{N} \binom{N}{k} {p_f}^{k}(1-p_f)^{N-k} + p_f \sum_{k=\frac{N+1}{2}}^{N} \binom{N}{k} {p_f}^{k}(1-p_f)^{N-k}$$

$$= (1-p_f+p_f) \sum_{k=\frac{N+1}{2}}^{N} \binom{N}{k} {p_f}^{k}(1-p_f)^{N-k}$$

$$= \sum_{k=\frac{N+1}{2}}^{N} \binom{N}{k} {p_f}^{k}(1-p_f)^{N-k}$$

*** Pascal's Rule:** $\binom{N}{k} = \binom{N-1}{k-1} + \binom{N-1}{k}$

** $\binom{N+1}{\frac{N+1}{2}} = \binom{N}{\frac{N-1}{2}} + \binom{N}{\frac{N+1}{2}} = \binom{N}{N-\frac{N+1}{2}} + \binom{N}{\frac{N+1}{2}} = 2\binom{N}{\frac{N+1}{2}}$