Improving scalability of permissioned blockchains by making Raft Orderer to understand the underlying network topology

Kiran Kumar Kondru[1], Saranya Rajiakodi[2]

{ kirankondru.tech@gmail.com[1], saranya@acad.cutn.ac.in[2] }

Department of Computer Science, Central University of Tamil Nadu, Thiruvarur, India[1,2]

Abstract. Blockchains like Hyperledger Fabric are comparably faster than public blockchains like Ethereum. These Permissioned Blockchains do not need to consider for in-built security like Bitcoin. The core part of Hyperledger Fabric, the orderer, replicates blocks across the blockchain network. The orderer follows the raft consensus protocol, which is already used in many distributed applications, from distributed databases to cloud based systems. However, strong leader based consensus algorithms like Raft have a bottleneck and this limits scalability. Here, we propose an improved raft consensus protocol which is made to perform better, even with an increased number of nodes, by making the leader delegate the responsibility of broadcasting to proxies and just focus on sequencing the clients' commands.

Keywords: permissioned blockchain, hyperledger fabric, consensus, raft algorithm, scalability.

1 Introduction

Blockchain technology has been becoming more and more adopted especially in modern large-scale enterprises. Both public and permissioned/consortium blockchains have found utilization in various fields and not just restricted to cryptocurrencies. Enterprises are more taken to consortium blockchains like Hyperledger Fabric which offer services including Smart contracts with various associated trade partners.

Raft consensus [1]algorithm is a strong leader-based protocol where the leader takes in clients' commands and transmits and makes the followers commit the transactions. Raft is designed to make it easy to understand and implement as opposed to Paxos consensus, which is very hard to understand though it is mathematically proven. Hyperledger Fabric, using Raft consensus instead of Paxos, is a proof of its ease of understanding [2], [3]. Raft has been formally proved with TLA+[1] and NLT [4]. Attempts are made to incorporate forensics [5] to better understand the protocol.

IACIDS 2023, November 23-25, Lavasa, India
Copyright © 2024 EAI
DOI 10.4108/eai.23-11-2023.2343202

With wider adoption of permissioned blockchains like Hyperledger Fabric, the degradation of performance is observed when there are a lot of participating nodes in the blockchain. This issue of scalability has been present but is not addressed until recently. Traditional distributed consensus algorithms are targeted at databases, especially for those distributed databases/datastore [6], and they usually don't need those numerous nodes to serve clients. However, with the advent of blockchains, the possibility of adding more nodes to the network heavily increased. Beyond a threshold, the traditional consensus algorithm degrades faster.

In the following sections, we discuss the attempts made to increase throughput and scalability.

2 Related Works

Various attempts are made to either mitigate or minimize degradation of performance in the traditional consensus algorithms [7]–[14], especially extensions of Paxos. They achieved reasonably good results. Most of the improvements are optimizations of the original consensus algorithms. Here, in this paper, we discuss some of these consensus mechanisms and how they try to solve the problem of scalability.

Most of these algorithms haven't been designed taking into consideration the underlying physical network topology. For example, in the paper [15], Raft didn't take into consideration that there could be a chance of partial link failure and only one-way communication is possible. We choose the Raft consensus algorithm[16] to improve this, as it's being widely used, extensively cited, and has proof of correctness through formal verification. It is being used in one of the more famous permissioned blockchains, Hyperledger Fabric, as an order .
In our proposed improvements to Raft algorithm, we aim to make 2 fundamental improvements. (1) To reduce the number of broadcasts the leader has to make for every request from the client; this we achieve by delegating to some nodes to pass on the messages from the leader. And (2) to make the followers return a read request directly to the connected clients instead of routing through the leader. A recent protocol, Canopus [17] has a similar way of solving the problem.

3 RAFT Consensus Algorithm

Raft consensus algorithm is specifically designed to make it more understandable in contrast to the famous Paxos consensus. It is designed as a strong leader-based consensus protocol where the leader actively participates in every decision. Though the performance of Raft is comparable to that of Paxos, there are still issues with scalability. When a greater number of nodes is added to the cluster, the performance degrades faster. The leader is in charge of receiving all the client requests and sequencing those commands and broadcasting them to the follower nodes. When the majority of the nodes accept and replicate the commands, the Leader sends a response to the clients.

3.1 Working of Raft Algorithm

Raft algorithm is a strong leader-based consensus mechanism but is not a centralized one, as any follower can become a leader in the cluster. When the cluster begins, it's operations for the first time, all the nodes in the group start off as followers. One of the followers wakes up from a random timer and realizes there is no leader in the group to co-ordinate clients' commands and becomes a candidate. It will increment the term number and send requests votes to other followers. The followers send a Yes vote to the candidate on a first come first serve basis. The Candidate, after receiving a majority of the votes (n/2 + 1), becomes a Leader and sends broadcasting heartbeat messages indicating to all the followers who the leader is and what the term is. The Leader handles all the client commands, sequences them and broadcasts them to the followers. The Leader then awaits a reply from the majority of the followers and sends back a confirmation message to the client. This is the general working of the Raft consensus algorithm in short. The below illustration depicts the same.

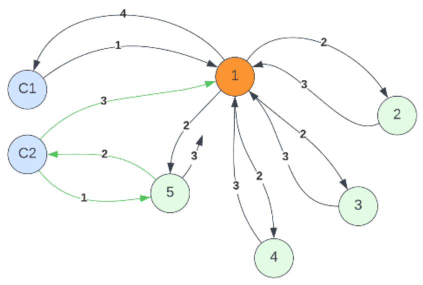


Figure 1 Functioning of a Raft cluster after a client's request

In the Fig 1, C1 and C2 and clients. C1 requests the leader '1' to set a value, which '1' broadcasts to nodes 2,3,4 and 5. In the reply to nodes 2,3 and 4 respond to '1' positively, but the response from the follower '5' got lost. This doesn't stop the leader as it has received a response from the majority of the followers (2,3,4) apart from itself and hence it considers the request to be committed by the cluster and in the last step (4), it responds to the client C1. In the same figure, client C2 tries to make a read request to node 5, which is a follower, but it responds by saying go to leader '1'. Then C2 contacts the leader '1'. This is a brief description of how a raft cluster works.

3.2 Issues with Throughput and Scalability

- There is only node, the leader that (1) caters to all the client requests, (2) sequences them in the order of commands received, (3) broadcasts the commands to followers to replicate, (4) receive majority feedback from the followers for successfully committing to their log, and (5) Finally replying to the client of the successful completion of the request. This makes the leader a lot slower and puts a theoretical limit to how many requests it can handle with a specified period of time.
- As the number of nodes in the cluster increases, the broadcasts also increase, in turn greatly increasing the leader's responsibility to await the majority of nodes to reply. This also puts a theoretical limit to how much the size of the cluster can grow without significantly degrading the performance of the cluster as a whole.

As stated above, to tackle these issues we propose a novel solution to improve throughput and scalability of the Raft cluster.

4 System Model

As mentioned in the above section, the issues with throughput and scalability are approached systematically here and we propose a solution that would greatly reduce degradation of performance as the nodes grow and increase throughput.

4.1 Proposal Description

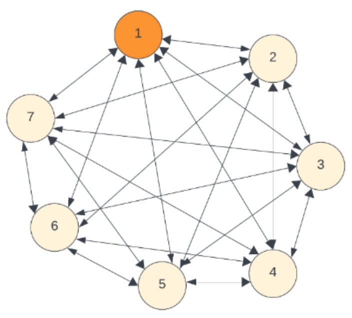
For every client command, the replication happens with all the broadcasting from the Leader and appropriate response back. This virtual broadcasting assumes every node is directly connected to the leader, but in reality, the underlying network devices such as switches take care of the routing process. Sometimes some of the switches are overused and some under used. Raft, by design (like Paxos) theoretically assumes one-to-one connection of all the nodes. As there is no one physical network setup, the assumption is a useful abstraction. Here too, in our proposal, we don't assume a specific network topology (as there are many) but we question the non-feasible one-to-one complete graph approach. For solving this we assume more of a tree like message passing between the servers of the cluster. Layered Paxos [18] also follows a similar design.

Our proposal has two sections to it as described as follows:

a) **Delegating broadcasting responsibilities**: Instead of broadcasting to every node in the cluster, the Leader broadcasts only to a select couple of nodes in the network which in turn propagate the broadcast to a select couple of nodes. Only 2 levels of propagation is required if we properly choose that number of selected nodes. We elaborate more below.

b) **Delayed local read response**: While in the original raft algorithm, only the leader is allowed to reply to the clients after a command is append to the Log of the cluster. We relax this condition by allowing any follower node connected to a client to respond to a read request.

The following illustration Figure 3, shows how a 7 node Raft cluster looks like



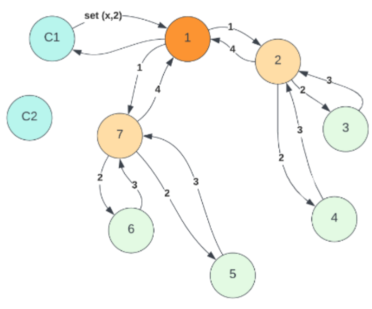


Figure 2: Raft assumes a one-to-one connection; Figure 3: Proposed two-level replication broadcast

We are going to elaborate on our above proposed design changes in the following sub-section.

4.2 Two level broadcast propagation

In the first part of the proposal, we delegate the responsibility of broadcasting the replication command (aka AppendEntries) to some pre-listed nodes in the cluster. Those nodes in turn pass on the broadcast to others in the cluster. The response of this AppendEntries command is sent back to the Leader who inturn replies to the client. This is illustrated in Fig 5 with a 7 node Raft cluster.

Clients C1 **.**sends a set(x,2) message to set the value of 'x' to '2' with the raft cluster. The leader no. 1, then instead of broadcasting to all the 6 other follower nodes, sends it to only two other nodes numbered 7 and 2. Node 7 follows up with nodes 6 and 5 and gives back their replies to the leader '1'. This multi level broadcast message propagation might seem to contain more rounds, but it frees the leader '1' to accept more clients' requests and do the same all over again. If we take the structure of the underlying physical network, this multi-level propagation takes off the load on some overworked switches and balances the broadcasts across all the network.

As the network size grows, this 2-level broadcast can be maintained just by increasing the number of nodes that the leader communicates to. If there are 5 or 7 nodes, then delegating to 2 proxy nodes is enough. If the network is 9 or 11 or 13, propagating to 3 nodes is required. All this delegating has been done, taking into account not to change the safety property of the Raft protocol. For this, each server is given a responsibility for an 'm' number of nodes. If there are 'n' number of nodes, then we should choose a number 'm' such that the following equation is satisfied.

$$Log_{m}(n-1) \leq 2. \tag{1}$$

4.3 Delayed Local Read Request

Stale reads are a real problem, and Raft strictly guarantees consistency, and no stale reads are allowed. This is enforced by the Raft leader responding to even read requests of clients. Usually, for most databases, there are more read requests than "write" requests and most are optimized for read requests. We propose to improve the throughput of a raft cluster by making follower nodes respond to the read requests of the connected clients.

4.4 Simulation

Raft's performance didn't change much even when simulated in a container[19] .We have chosen Mininet for the simulation and chosen 'etcd' a well known Key-Value data store implementing the raft consensus protocol. Mininet is an emulator where we can run a real-world application in a virtualized environment where we can create a virtual network of our choice using python scripts.

5 Analysis

Given that various overlay network topologies are available, like Hub-and-Spoke, Full Mesh, Partial Mesh, Lead-Spine topology, Multi-tier architecture, End-to-End overlays etc, we choose Hub-and-Spoke topology to illustrate the actual working of the raft replication process.In this section, we delve into the actual working of the raft consensus protocol with each step separately illustrated showing how long it would take the original raft consensus protocol to complete one round of replication in the cluster.

Table 1. comparison of series of events of original raft with modified Raft

Steps	**Raft(Original)**	**Description**	**Raft(Modified)**	**Description**
1	C1 → 1	Client C1 send first request to the leader node ‘1’	C1 → 1	Client C1 send first request to the leader node ‘1’
2	1 → H	Leader ‘1’ sends request to the Hub ‘H’	1 → H	Leader ‘1’ sends request to the Hub ‘H’
3	H → 2,3,4,5	Hub broadcasts to all Followers	H → 2	Sends to the deputy ‘2’
4	2 → H → 1	‘2’ sends a reply back through H	2 → H → 3,4,5	‘2’ broadcasts to followers
5	3 → H → 1	‘3’ responds	C1 → 1; 5 → H → 2	New client request and ‘5’ sending back to ‘1’
6	4 → H → 1	‘4’ replies to ‘1’	1 →H → 2; 4→H →2	‘1’ sends new request to ‘s’ and parallelly 4 replies to ‘2’
7	1 →C1	Leader ‘1’ replies to Client (first request)	2 →H →1; 5 →H →2	‘2’ sends the summary of votes to ‘1’; ‘5’ replies to ‘2’ on second request
8	1 →H →2,3,4,5	New broadcast for second Client request	1 →C1; 4 →H →2	While ‘1’ is replying to C1 on 1st request, 4 responds to ‘2’ for the 2nd request

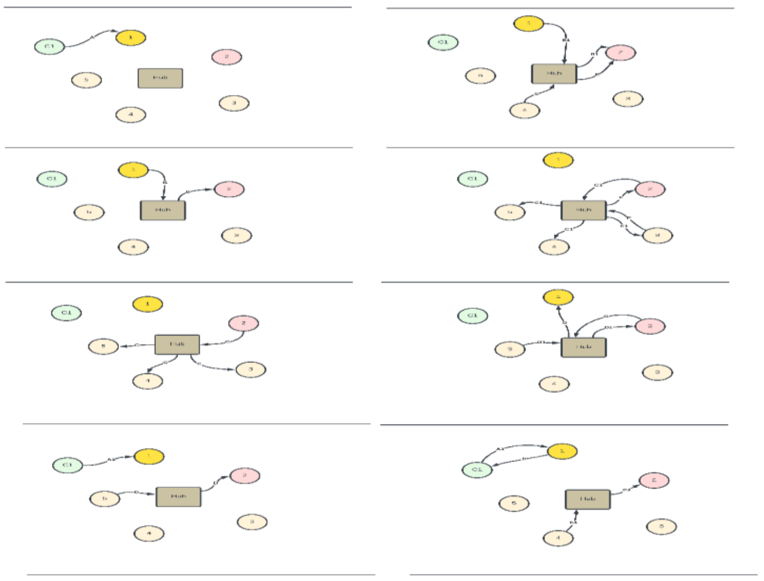


Figure 4: Client command propagation through repliaction in raft cluster

In table 1,the tabular construction of events in the order shows how the the modified raft takes in more client requests and delegates them to a proxy leader, effectively parallelizing the communication process while not overloading the central hub. By the time of second client request was received by the leader ‘1’, the modified raft has already taking replies from the followers.

6 Conclusion

In this paper, we presented a solution to improve the performance of the Raft consensus algorithm in throughput and scalability to improve the overall performance of a permissioned blockchain network which uses Raft. Since consensus is central to any blockchain network, improving the consensus protocol will definitely have orders of magnitude improvement in performance depending on how well the consensus protocol is designed. We showed that even with the increase in the number of nodes, the 2-level log replication broadcasting, along with delayed local reads, improves the overall performance.

References

[1] D. Ongaro and J. Ousterhout, “In Search of an Understandable Consensus Algorithm,” p. 18.

[2] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou, “A survey of distributed consensus protocols for blockchain networks,” *IEEE Communications Surveys & Tutorials*, vol. 22, p. 1432—1465.

[3] Z. Ye, X. Tong, W. Fan, Z. Zhang, and C. Jin, “A Raft Variant for Permissioned Blockchain,” *International Conference on Database Systems for Advanced Applications*, p. 685—689.

[4] H. Evrard, “Modeling the raft distributed consensus protocol in LNT,” *arXiv preprint arXiv:2004.13284*.

[5] W. Tang, “Raft-Forensics: High Performance CFT Consensus with Accountability for Byzantine Faults”.

[6] D. Huang, X. Ma, and S. Zhang, “Performance Analysis of the Raft Consensus Algorithm for Private Blockchains,” *IEEE Trans. Syst. Man Cybern, Syst.*, vol. 50, no. 1, pp. 172–181, Jan. 2020, doi: 10.1109/TSMC.2019.2895471.

[7] Digital Equipment Corporation and L. Lamport, “The part-time parliament,” in *Concurrency: the Works of Leslie Lamport,* D. Malkhi, Ed., Association for Computing Machinery, 2019. doi: 10.1145/3335772.3335939.

[8] L. Lamport, “Fast Paxos,” *Distrib. Comput.*, vol. 19, no. 2, pp. 79–103, Oct. 2006, doi: 10.1007/s00446-006-0005-x.

[9] L. Lamport, “Generalized consensus and Paxos”.

[10] P. Sutra and M. Shapiro, “Fast genuine generalized consensus,” *2011 IEEE 30th International Symposium on Reliable Distributed Systems*, p. 255—264.

[11] P. J. Marandi, M. Primi, N. Schiper, and F. Pedone, “Ring Paxos: A high-throughput atomic broadcast protocol,” *2010 IEEE/IFIP International Conference on Dependable Systems & Networks (DSN)*, p. 527—536.

[12] P. J. Marandi, M. Primi, and F. Pedone, “Multi-ring paxos,” *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2012)*, p. 1—12.

[13] C.-S. Barcelona, “Mencius: building efficient replicated state machines for WANs,” *8th USENIX Symposium on Operating Systems Design and Implementation (OSDI 08)*.

[14] M. Biely, Z. Milosevic, N. Santos, and A. Schiper, “S-paxos: Offloading the leader for high throughput state machine replication,” *2012 IEEE 31st Symposium on Reliable Distributed Systems*, p. 111—120.

[15] “MAKING RAFT CONSENSUS A LITTLE MORE FAULT-TOLERANT,” vol. 12, no. Special Issue 5, pp. 3763–3769, doi: 10.48047/ecb/2023.12.si5a.0273.

[16] H. Howard, “ARC: analysis of Raft consensus”.

[17] S. Rizvi, B. Wong, and S. Keshav, “Canopus: A scalable and massively parallel consensus protocol,” p*roceedings of the 13th International Conference on emerging Networking EXperiments and Technologies*, p. 426—438.

[18] A. Nagawiecki and S. Patterson, “Layered Paxos: A Hierarchical Approach to Consensus”.

[19] C. Oliveira, L. C. Lung, H. Netto, and L. Rech, “Evaluating raft in docker on kubernetes,” *Advances in Systems Science: Proceedings of the International Conference on Systems Science 2016 (ICSS 2016) 19*, p. 123—130.