symmetry

MDPI

*Article*

# Raft-PLUS: Improving Raft by Multi-Policy Based Leader Election with Unprejudiced Sorting

**Jinjie Xu [1,2], Wei Wang [1], Yu Zeng [3], Zhiwei Yan [3] and Hongtao Li [3,*]**

[1] Computer Network Information Center, Chinese Academy of Sciences, Beijing 100190, China; xujinjie@cnic.cn (J.X.); wangwei@cnic.cn (W.W.)
[2] University of Chinese Academy of Sciences, Beijing 100049, China
[3] China Internet Network Information Center, Beijing 100190, China; zengyu@cnnic.cn (Y.Z.); yanzhiwei@cnnic.cn (Z.Y.)
* Correspondence: lihongtao@cnnic.cn

**Abstract:** Raft is a fast, scalable, understandable consensus algorithm widely used in distributed systems. The Leader handles client requests and interacts with other servers to reach a consensus, so a stable, reliable, and powerful Leader is crucial for the cluster. We designed a policy-based voting mechanism to make the elected Leader as reliable as possible. In order to improve the asymmetric relationship between the Followers and Leader, we designed a mechanism to trigger a new round of the election actively so that the Leader node can actively transform into a Follower under certain conditions and enhance the symmetry between servers. Our proposed Raft-PLUS algorithm makes the elected Leader as reliable as possible through four election policies and designed three opposition policies to trigger a new round of the election. To verify the effectiveness of the Raft-PLUS algorithm, we configure different election and opposition policies on 12 servers to simulate the election and opposition process of the Leader and explain the process. To demonstrate the advantages of the Raft-PLUS algorithm, we built key-value stores based on Raft and Raft-PLUS, and we tested the performance of Raft-PLUS and the Raft algorithm in normal and abnormal states. Experimental results show that the Raft-PLUS algorithm has similar write throughput to the Raft algorithm under normal conditions. Regarding the quality of the Leader network changes, the average write throughput of the Raft-PLUS algorithm is 40% higher than that of the Raft algorithm. The Leader's CPU usage fluctuated; the average write throughput of Raft-PLUS was 38% higher than Raft.

**Keywords:** blockchain; consensus algorithm; Raft; leader election



**Citation:** Xu, J.; Wang, W.; Zeng, Y.; Yan, Z.; Li, H. Raft-PLUS: Improving Raft by Multi-Policy Based Leader Election with Unprejudiced Sorting. *Symmetry* **2022**, *14*, 1122. https://doi.org/10.3390/sym14061122

Academic Editors: Sergei D. Odintsov, László T. Kóczy and Kuo-Hui Yeh

Received: 11 April 2022
Accepted: 27 May 2022
Published: 29 May 2022



## 1. Introduction

Blockchain is a combination of peer-to-peer networks, distributed systems, and cryptography. In recent years, many applications of blockchain technology have emerged—Bitcoin, Ethereum, etc. A typical blockchain application usually transmits data through a P2P network, adopts a consensus algorithm to achieve data consistency, and uses cryptography principles to ensure security features [1]. Because of a series of technologies, blockchain has the characteristics of decentralization, immutability, traceability, etc.

According to different access methods, there are three types of blockchain: public blockchain, consortium blockchain, and private blockchain. Public blockchain allows anyone on the Internet to participate. The private blockchain is typically created and used only by individuals or individual organizations, providing organizations with a secure, traceable system [2]. The openness of consortium blockchain is between public blockchain and private blockchain. Consortium blockchain allows designated persons or institutions to read or write data on the blockchain. The consensus algorithm originated in the field of distributed systems to solve the problem of agent crashes or communication interruption in multi-agent systems [3]. Different types of blockchains usually adopt different consensus algorithms [4,5]. Due to the existence of malicious nodes, the public blockchain needs to

use a consensus algorithm with Byzantine Fault Tolerance, such as PoW (Proof of Work) [6], PoS (Proof of Stake) [7], PBFT (Practical Byzantine Fault Tolerance) [8], etc. Because the number of nodes in the consortium blockchain is smaller than that in the public chain and the evil cost is more than in the public chain, crash fault tolerance consensus algorithms can be adopted, such as Paxos [9] and Raft [10].

## 2. Related Work

### *2.1. Raft's Problem*

Paxos is the most classical algorithm of the crash fault tolerance consensus algorithms. However, Paxos is difficult to understand and implement, which can be reflected in several articles trying to explain its principle [11,12]. Raft is a consensus algorithm with intelligibility as its design goal and is no different from Paxos in efficiency. It decomposed the consensus problem into three subproblems (leader election, log replication, and safety) and solved them separately. This approach makes Raft easier to understand and implement. At present, Raft is one of the most active CFT consensus algorithms widely used in blockchain systems and distributed storage [13–15].

In the Raft algorithm, the Leader acts as the central node for data distribution in the distributed system, and the performance of the entire system will be significantly affected when the server performance decreases due to some reasons, such as network fluctuations or limitations on server capacity. Moreover, only when the Leader node is down will the cluster start a new round of voting due to the heartbeat packet timeout and elect a new leader. This method strengthens the power of the Leader node, resulting in asymmetric status between the Leader node and other nodes. In order to improve the asymmetry of the status between nodes in the Raft algorithm, we propose the Raft-PLUS algorithm, which actively triggers a new round of leader elections by adding an opposition mechanism and conducting unbiased sorting according to various strategies in the election. The optimal node acts as the Leader of the next term, ensuring that the cluster has more robust and stable performance.

### *2.2. Raft's Improvement*

As a classic and efficient consistency algorithm, the Raft algorithm has many improvements in different scenarios. Pâris et al. proposed a dynamic linear voting protocol applied to the Raft algorithm, which improves the system availability and enables clusters to tolerate fewer participants [16]. Fu et al. proposed the AdRaft algorithm by improving log replication and leader election phases to improve throughput and reduce the latency of the blockchain system [17]. HhRaft can resist certain Byzantine attacks by adding a Monitor node to monitor the process of Leader election and log replication [18]. Weighted RAFT improves the Leader election process by considering the communication quality of wireless networks and the computing state of IoT devices, making the node with better performance become the Leader and reducing the delay of system forwarding [19]. Zizhong et al. proposed the CRAFT algorithm by using erasure coding to save storage and network cost while keeping the same liveness as Raft [20]. Dezhi et al. introduced digital signatures and nested hash mechanisms to make Raft byzantine fault tolerant [21]. Rihon et al. used the Kademlia algorithm to network Raft's underlying network and to reduce the communication complexity [22].

In summary, the improvement of the Raft algorithm mainly focuses on three aspects. (1) Make the Raft algorithm resist malicious attacks and improve algorithm security. (2) Improve the log replication process of the algorithm to improve the algorithm's efficiency. (3) Improve the Leader election process, reduce the election time and elect a suitable node as the Leader. However, no research has proposed an objection mechanism based on dynamic monitoring of the leader node status. When the status of the leader node declines, this mechanism allows the cluster to actively enter the following term and elect a new leader, thereby improving the performance and stability of the cluster. This paper dynamically monitors the status of the Leader node by adding an objection mechanism. When the leader

node is not suitable, the objection vote triggers a new round of the election, and a more powerful Leader node is elected through the election strategy.

This paper is organized as follows. Section 2 provides the process of server state conversion and Leader election in the Raft algorithm. In Section 3, the basic principle of Raft-PLUS, the policy of election and opposition are presented. Then, experiments and results are discussed in Section 4. Finally, Section 5 concludes this paper.

## 3. Raft Algorithm

The Raft algorithm builds on the ideas of the Paxos algorithm by splitting the algorithm into three main parts—leader election, log replication, and security—and by reducing the number of states in the state machine to increase its understandability. This section describes the server state and leader election process of the Raft algorithm.

### *3.1. State Conversion*

In the Raft algorithm, the server has three states: Leader, Candidate, and Follower (Figure 1). Their state transition is controlled by random timeout, and each server is in these three states at any time. Each state has different operations. The Leader processes client requests forwarded by other servers and sends log entries required by each Follower. If the log entries are empty, the Leader sends heartbeat packets. The Candidate sends a vote request RPC to other servers until more than half of the responses are received or the timer times out again to enter the election of the next term. When the Candidate receives a legitimate heartbeat packet from the Leader, the Candidate will return to the Follower state. The Follower responds to a vote request sent by a Candidate and sends a vote request response to a Candidate. It also responds to requests from the Leader, and if the request contains log entries, the log will be safely added to the local state machine, and the result of processing will be sent to the Leader.

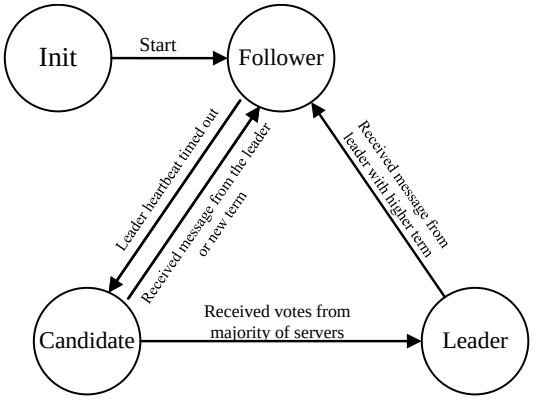


**Figure 1.** State conversion.

### *3.2. Leader Election*

In the Raft algorithm, the initial state of all servers is Follower. When the timers on some servers expire, the status of these servers will become Candidate and send RequestVote RPCs to other servers. Followers will verify that the received RequestVote RPC conforms to the election rules (including verifying the term number, last log index, and last log term). If the result is passed, the Follower will vote for the corresponding Candidate on a first-come-first-served basis. When a Candidate receives votes from a majority of servers in the whole cluster, it wins the election of the current term, becomes the Leader, and sends a heartbeat packet to the other servers in the cluster to announce the identity of the Leader. After receiving the heartbeat packet from the Leader, other Candidates verify the validity of the heartbeat packet. If the verification succeeds, the Candidate will return to the Follower state. If the verification fails, it will remain in the Candidate state.

If all the votes are divided among multiple Candidates, and no one gets a majority of the votes, then the current term of elections will be invalidated. Over time, the Candidate

will increase the term and start the next round of elections. Since each timeout period is a random number in an interval, it is highly unlikely that the votes will be divided in each round. This scheme can quickly complete the Leader election.

## 4. Raft-PLUS Algorithm

In the Raft algorithm, the Leader needs to periodically send heartbeat packets to other servers to refresh the timeout on the Follower. If the Leader's performance deteriorates due to some reasons (network fluctuations, limitations on server capacity, or disk IO overloaded), the entire system's performance will deteriorate. In this case, the best practice is actively switching the leader server, turning the current Leader into a Follower, and starting a new round of leader elections. However, in the Raft algorithm, as long as the current Leader can still send heartbeat packets, it will not initiate a new round of elections, making the Leader more powerful than other nodes. In the Raft-PLUS algorithm, we propose a negative voting strategy to strengthen the symmetry between nodes. Followers can send a negative vote (each Follower sends a negative vote once in each term) to the Leader of the current term. When the Leader receives more than half of the negative votes of the cluster, it will voluntarily give up the leader status. The system then starts a new election.

At the same time, to avoid selecting an inappropriate server as the Leader in the election, we propose a voting mechanism with strategies. Unlike the first-come-first-served voting strategy in the Raft algorithm, in the Raft-PLUS algorithm, each Follower can choose a variety of election policies to vote for a Candidate. Specifically, each Follower will collect Candidate RequestVote RPC for a period of time and add these requests to the vote request pool. This stage, called the vote collection stage, starts from the first RequestVote RPC received, and the end is controlled by the timer and the number of requests in the vote request pool. The timer sets a fixed value, which should be greater than 0 and less than the difference between the maximum and minimum values of the follower election timeout. The voting stage is entered when the time expires or the number of requests in the vote request pool exceeds 1/3 of the cluster number. During the voting stage, followers select the best Candidate from the vote request pool based on the configured voting policy and send the vote to the Candidate. Figure 2 compares the Raft and Raft-PLUS algorithm of Follower operations after receiving RequestVote RPC.

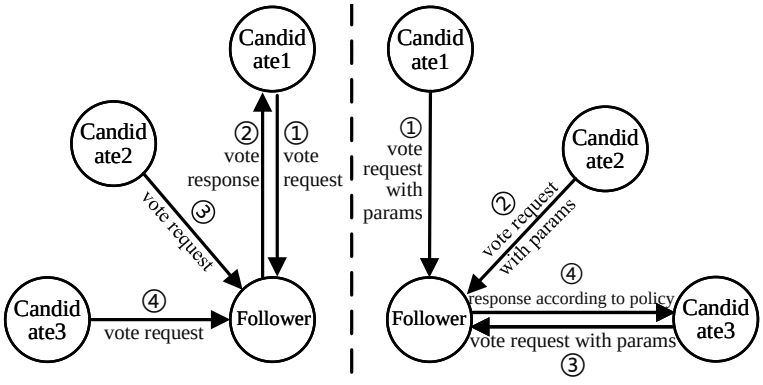


**Figure 2.** The process of Follower after receiving a request in Raft (**left**) and the Raft-PLUS (**right**) algorithm.

In the Raft algorithm, the Follower votes on a first-come, first-served basis. Once the Follower receives the RequestVote RPC of the Candidate, it votes for the Candidate immediately after verification. In the Raft-PLUS algorithm, we design a voting pool, add it to the voting pool after receiving the vote, wait for a particular condition, and vote according to the election strategy. In the experiment, we found that when the cluster is small, it is usually because of the condition of receiving more than one-third of the number of requests triggered to enter the voting stage. When the cluster is large, it is often because

the timer expires to enter the voting stage. This method can effectively shorten the election time and increase the system's availability.

### *4.1. Opposition Policy*

We have designed three opposition policies. When the Leader reaches the set opposition conditions, the Followers will send the negative vote to the Leader, and the Followers can choose appropriate opposition policies according to different concerns.

- Opposition policy based on network measurement:
  The network performance of delay and jitter from the Follower to the Leader is measured on a voting basis. When the network quality is lower than the threshold (e.g., the delay is more than 100 ms or the jitter is more than 50 ms), the Follower sends the negative vote to the Leader.
- Opposition policy based on processing capacity:
  This policy votes on the Leader's processing capability, and the server's processing capability is measured by the hardware performance (including CPU usage, memory usage, and disk I/O usage) and the time required to commit logs. If the processing capability of the Leader is lower than the threshold set by the Follower, the Follower sends the negative vote to the Leader.
- Opposition policy based on the number of logs committed:
  This policy votes according to the number of logs committed by the Leader. When the number of logs committed by the Leader reaches a certain threshold, the Follower sends the negative vote to the Leader.

### *4.2. Election Policy*

There are four election policies. Candidates send voting requests to other servers with all parameters required by election policies (Figure 2 right part). Like opposition policies, followers can also choose appropriate election policies based on different concerns.

- Election policy based on network measurement:
  This policy is similar to the opposition policy based on network measurement. The network delay and jitter from followers to candidates are measured as the basis for voting. To minimize the time cost of network measurement, the Follower immediately begins to measure the network performance when it receives a vote request from the Candidate—the Follower votes for the Candidate with the best network performance.
- Election policy based on processing capacity:
  This policy is similar to the opposition policy based on processing capacity, which sorts candidates according to their processing capacity and votes for the Candidate with the best processing capacity.
- Election policy is based on the number of times the leader has been elected:
  Based on the number of times the leader has been elected, this policy votes for the Candidate who has been elected more times to ensure the leader's stability. If multiple candidates have been elected the same number of times, the vote is given to the Candidate who sends the request first.
- Election policy based on the number of client requests received:
  A server in any state may receive client requests. If a Follower receives a client request, it will forward the request to the Leader. The Leader handles the request and sends the result to the client. This policy is based on the number of client requests received by the server as the basis for voting. Followers will vote for the Candidate that has received the most client requests during the previous term.

Figure 3 shows the Leader election and opposition process of the Raft-PLUS algorithm. In the Leader election phase, the Candidate sends RequestVote RPC containing election parameters (including processing capacity, the number of times the leader, and the number of client requests received) to other servers. After receiving the RequestVote RPC message, the Follower will verify the request, add the voting request to the request pool and start the

timer. When the timer times out (usually set to one-third to one-half of the election timeout) or the number of voting requests in the request pool exceeds one-third of the number of clusters, the Follower will sort the requests according to the preset election policy and send the vote to the highest-ranked Candidate. If the Candidate receives a majority of votes, it will successfully transform into a Leader and start processing requests from clients.

When the election is completed, the cluster enters the leader opposition phase, and each Follower verifies whether it satisfies the opposition policy according to the heartbeat packet with parameters sent by the Leader. If the Leader does not satisfy the Follower's requirements, the Follower sends a negative vote to the Leader. After receiving the majority of negative votes, the Leader's heartbeat packet will no longer refresh the Follower's election timer. After the timer expires, the cluster will enter a new term and start the election.

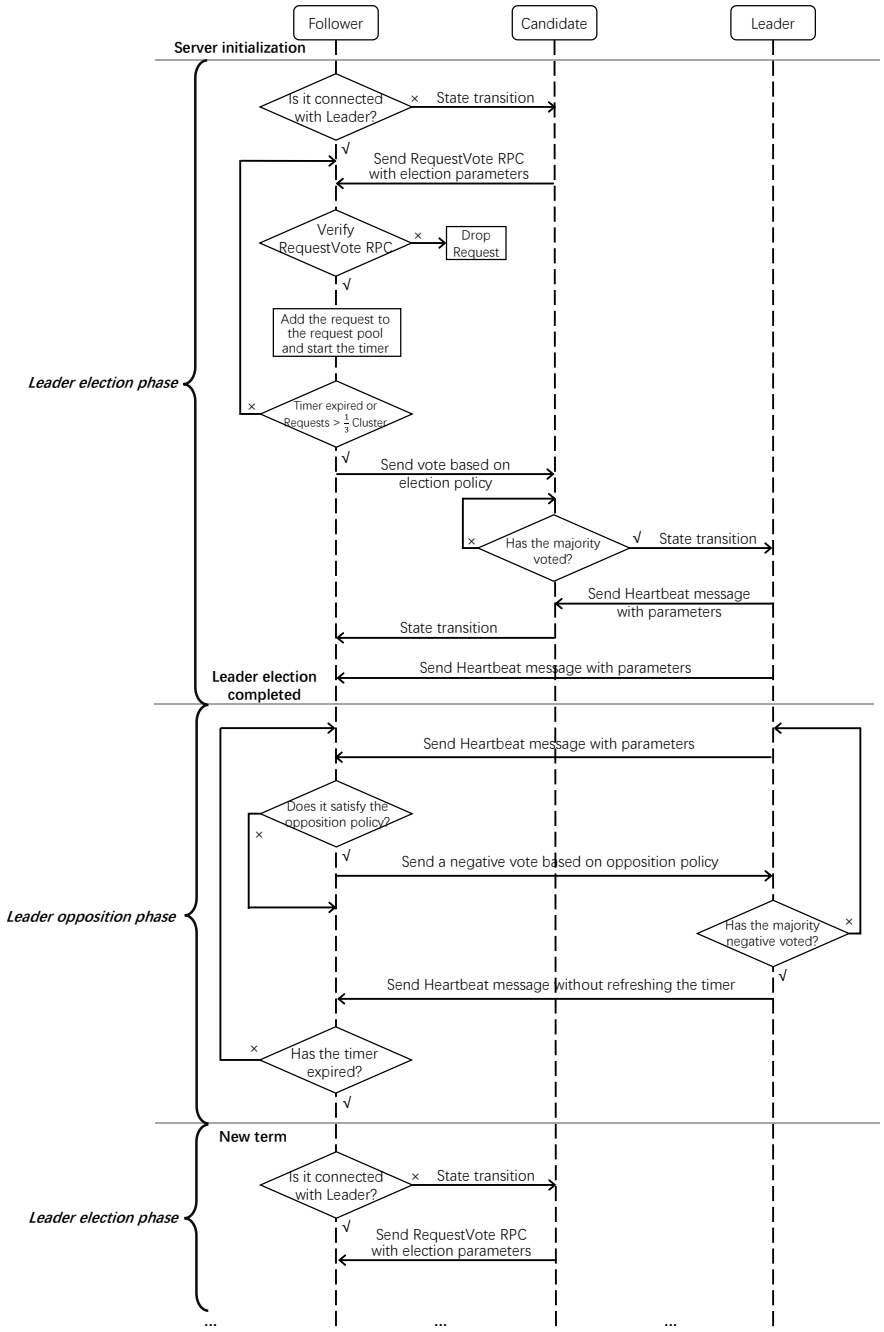


**Figure 3.** Leader election and opposition process in the Raft-PLUS flow chart.

## 5. Experiment

We performed functional verification and performance testing of the Raft-PLUS algorithm. In the functional verification, we designed a set of experiments to verify the effectiveness of the Raft-PLUS algorithm. The cluster has 12 servers; each server is called node1, node2 … node12. The opposition policy value of each server equals the remainder of server ID divided by 3, which ranges from 0 to 2, corresponding to the three opposition policies of network measurement, processing capability, and the number of logs committed. The thresholds of opposition policies based on the number of logs committed are set to 5.

Similarly, the election policy value equals the remainder of the ID divided by four and ranges from 0 to 3, corresponding to four elections policies: network measurement, processing capacity, the number of times the leader is elected, and the number of client requests received. Based on the experimental configuration described above, node1 is configured with processing capability policy in election and opposition policy, and node2 is configured with election policy based on the number of times the leader is elected and opposition policy based on the number of logs committed.

Figure 4 shows the experimental results, and Figure 4a describes the election process. In the beginning, all servers change from the initial state to the Follower state, and then due to timeout, some servers change to the Candidate state. Followers receive RPC from the candidates and send their votes according to the election policy. Finally, node11 receives seven votes (including its vote), gets the majority of the votes of the cluster, and becomes the leader for the current term. Figure 4b describes the process that after receiving more than half of the negative votes, node11 voluntarily gives up the leader status, becomes a follower, and enters a new term. Node1, 4, 10, and 7 send the negative votes due to the leader's lack of processing capacity. After the client has sent five requests, Node2, 8, and 6 also send negative votes to the leader. In the end, Node11 receives seven negative votes from the followers and actively changes the follower status. The system enters the next term.

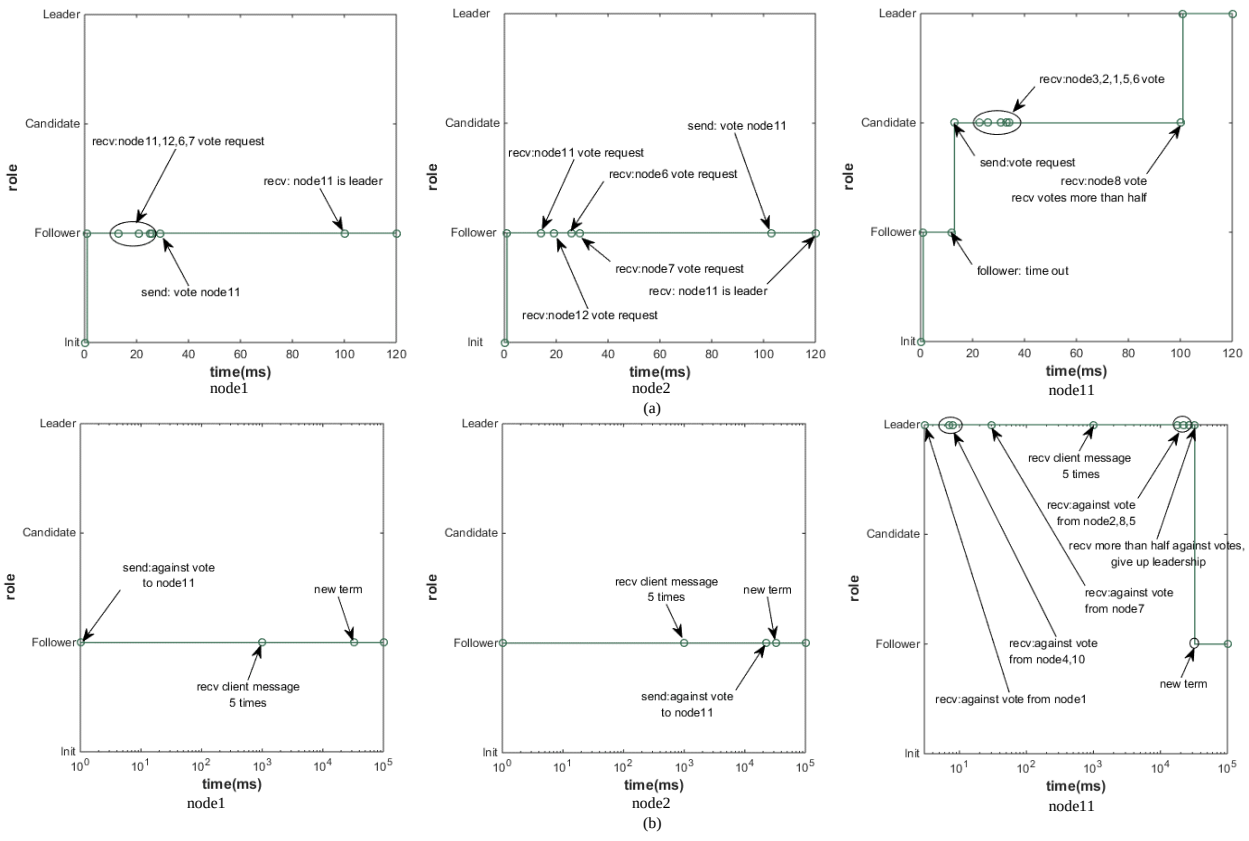


**Figure 4.** Result of experiment. (**a**) Election process. (**b**) Opposition process.

In the performance test, we compared the performance of the Raft and Raft-PLUS algorithms in the normal state and the abnormal state. In the normal state, the performance of the two algorithms under different numbers of nodes is shown in Figure 5. The experimental results show that in the TPS (Transaction Per Second) performance test at the scale of 10 to 50 nodes, the performance of the Raft-PLUS algorithm is generally the same as that of the original Raft algorithm and there is no additional overhead or performance degradation.

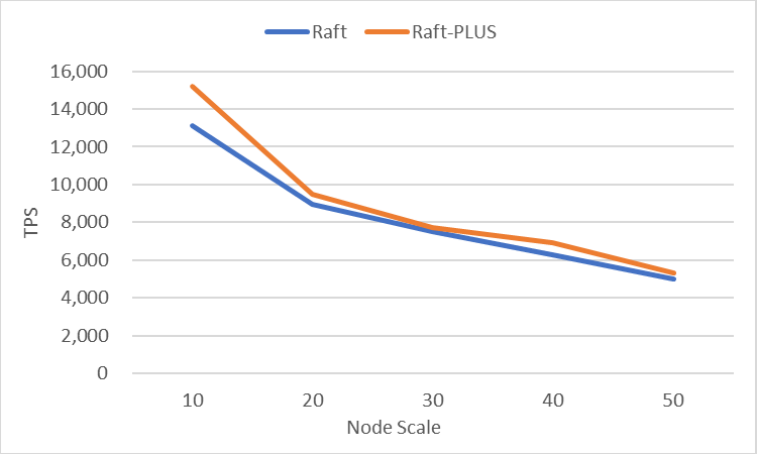


**Figure 5.** Performance comparison between Raft and Raft-PLUS algorithms in normal state.

In the abnormal state, we simulate the performance degradation of the Leader node at the scale of 10 nodes and compare the performance of Raft and Raft-PLUS algorithms. The experimental results are shown in Figure 6. As the network quality decreases (the test range of network quality is: the node network bandwidth range is 40–400 kbps, the delay range is 0–50 ms, and the packet loss rate range is 0–10%), the TPS of Raft is also declining, significantly when the packet loss rate exceeds 5% (the network quality is 50%), the system performance drops rapidly, and the Raft-PLUS algorithm maintains a stable TPS because it can actively switch the leader in time. As a result, Raft-PLUS has an average write throughput 40% higher than Raft.

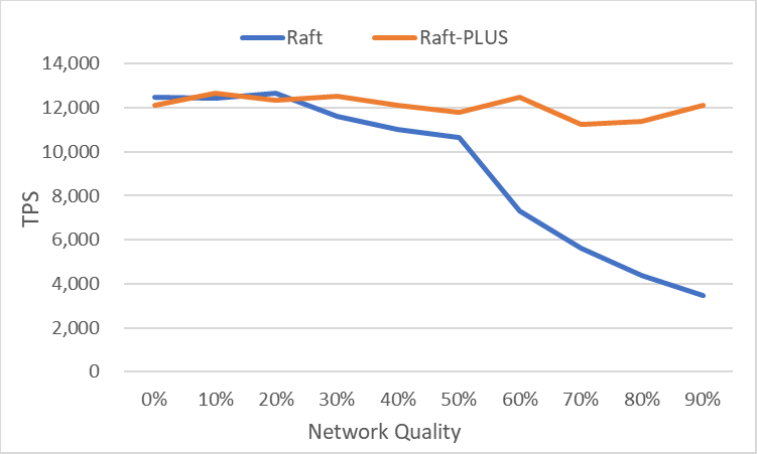


**Figure 6.** Raft and Raft-PLUS algorithm on network quality and TPS comparison.

When the CPU usage increases, as shown in Figure 7, the TPS of the Raft algorithm keeps decreasing, especially when the CPU usage reaches more than 80%, the system performance will drop rapidly. At this time, the best solution is to trigger the Leader node replacement actively. Turn the current Leader node into a Follower node, and elect a new Leader. When another node becomes the Leader, the state of the previous Leader node will

no longer affect the performance of the cluster. If the node continues to be the cluster leader, it will significantly reduce the performance of the entire system. Therefore, the Raft-PLUS algorithm switches the Leader node when the CPU usage reaches more than 60%, which ensures the system's performance. As a result, Raft-PLUS has an average write throughput 38% higher than Raft.

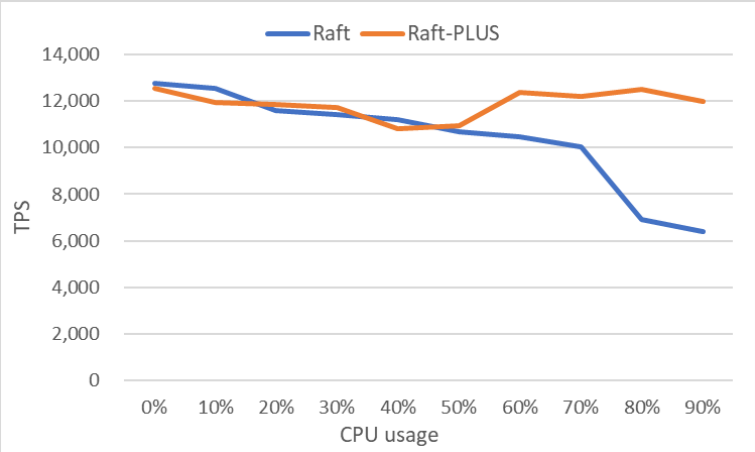


**Figure 7.** Raft and Raft-PLUS algorithms on CPU Usage and TPS comparison.

## 6. Discussion

In this section, we compared the Raft-PLUS algorithm with other similar studies in raft improvements. Table 1 compares the research on Leader election of the Raft-PLUS algorithm and other improved Raft algorithms. Most algorithms adopt the first-come-first-served method in the follower voting stage. KRaft combines the characteristics of the Kademlia protocol to calculate the voting results according to the hash table, and the Raft-PLUS and Weighted Raft algorithms vote according to the weight of the Candidate. However, the cluster state changes dynamically. Most of the improved algorithms do not consider the dynamic changes of the cluster state, while Pirogue and Raft only consider the node state in the voting stage and do not track the cluster state. Raft-PLUS keeps monitoring the node states during the cluster service period. It monitors the status of nodes and activates a new round of the election. We also noticed that some algorithms had improved security so that the improved algorithm could resist a certain degree of Byzantine attack. However, the participants are all authenticated and licensed in the consortium chain. When a Byzantine attack occurs, network communication logs can analyze attackers and remove them from the network to ensure the security of the blockchain network.

**Table 1.** Comparison with other research.

| Features \ Research | Raft | Raft-PLUS | Pirogue [16] | hhRaft [18] | Weighted RAFT [19] | AdRaft [17] | KRaft [22] |
|---|---|---|---|---|---|---|---|
| Improvements | \ | Leader election | Leader election | Leader election and safety | Leader election | Leader election and log replication | Leader election and log replication |
| Voting method | FCFS [1] | VBW [2] | FCFS | FCFS | VBW | FCFS | CBT [3] |
| Dynamically monitor cluster | no | yes | yes | yes | no | no | no |
| Active election | no | yes | no | no | no | no | no |
| Resist Byzantine attacks | no | no | no | yes | no | no | no |

[1] First Come First Serve [2] Voting By Weight [3] Calculate By Table.

## 7. Conclusions

This paper presents the Raft-PLUS algorithm, Improving Raft by multi-Policy based Leader-election with Unprejudiced Sorting. In the Raft-PLUS algorithm, if the Leader

triggers the preset opposition policy, it will actively give up the Leader state. The cluster will enter the following term and start a new round of elections. In the election process, we propose voting with policies instead of the first-come-first-served method in the Raft algorithm. Followers can choose appropriate strategies so that the best candidates can be elected for complicated applications [23]. Experiments show that the Raft-PLUS algorithm is effective in Leader node switching. Furthermore, compared with the original algorithm in the performance test, the experimental results show that the Raft-PLUS algorithm has the same performance as the original algorithm under normal conditions, with low extra overhead, and higher and more stable performance under certain conditions.

**Author Contributions:** Methodology, J.X.; software, J.X.;formal analysis, Z.Y.; Funding acquisition, Y.Z. and Z.Y.; Project administration, H.L.; Writing —original draft, J.X.; Writing—review & editing, W.W. and Z.Y. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was funded by the National Key Research and Development Program of China grant number: 2019YFB1804500. This work of Z.W. Yan was funded by the Beijing Nova Program of Science and Technology grant number: Z191100001119113.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** Not applicable.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Nakamoto, S. Bitcoin: A Peer-to-Peer Electronic Cash System. 2008. Available online: https://bitcoin.org/bitcoin.pdf (accessed on 10 April 2022)
2. Dylan, Y.; Peter, M.; Nik, R.; Karen, S. Blockchain Technology Overview. *NIST Interag./Intern. Rep.* **2018**, *8202*, 1–59.
3. Randell, B.; Lee, P.; Treleaven, P.C. Reliability issues in computing system design. *ACM Comput. Surv.* **1978**, *10*, 123–165. [CrossRef]
4. Zhang, S.; Lee, J.H. Analysis of the main consensus protocols of blockchain. *ICT Express* **2020**, *6*, 93–97. [CrossRef]
5. Du, M.; Ma, X.; Zhang, Z.; Wang, X.; Chen, Q. A review on consensus algorithm of blockchain. In Proceedings of the 2017 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Banff, AB, Canada, 5–8 October 2017; pp. 2567–2572.
6. Jakobsson M.; Juels A. Proofs of Work and Bread Pudding Protocols. In *Secure Information Networks*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 258–272.
7. Saleh, F. Blockchain without waste: Proof-of-stake. *Rev. Financ. Stud.* **2021**, *34*, 1156–1190. [CrossRef]
8. Castro, M.; Liskov, B. Practical byzantine fault tolerance. In *Proc OSDI*; USENIX Association: Louisiana/New Orleans, USA, 1999; Volume 99, pp. 173–186.
9. Lamport, L. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM* **1978**, *21*, 558–565. [CrossRef]
10. Ongaro, D.; Ousterhout, J. In search of an understandable consensus algorithm. In Proceedings of the 2014 USENIX Annual Technical Conference (Usenix ATC 14), Philadelphia, PA, USA, 19–20 June 2014; pp. 305–319.
11. Lamport, L. Paxos made simple. *ACM SIGACT News* **2001**, *32*, 51–58.
12. Meling, H.; Jehl, L. Tutorial summary: Paxos explained from scratch. In *International Conference on Principles of Distributed Systems*; Springer: Cham,, Switzerland, 2013; pp. 1–10.
13. Etcd-io. etcd. Available online: https://etcd.io/ (accessed on 10 April 2022).
14. TiKV-Authors. TiKV. Available online: https://tikv.org/ (accessed on 10 April 2022).
15. Hyperledger Fabric Authors. Hyperledger Faric. Available online: https://www.hyperledger.org/use/fabric (accessed on 10 April 2022).
16. Pâris, J.F.; Long, D.D.E. Pirogue, a lighter dynamic version of the Raft distributed consensus algorithm. In Proceedings of the 2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC), Nanjing, China, 14–16 December 2015; pp. 1–8.
17. Fu, W.; Wei, X.; Tong, S. An improved blockchain consensus algorithm based on raft. *Arab. J. Sci. Eng.* **2021**, *46*, 8137–8149. [CrossRef]
18. Wang, Y.; Li, S.; Xu, L.; Xu, L. Improved Raft Consensus Algorithm in High Real-Time and Highly Adversarial Environment. In *International Conference on Web Information Systems and Applications*; Springer: Cham, Switzerland, 2021; pp. 718–726.
19. Xu, X.; Hou, L.; Li, Y.; Geng, Y. Weighted RAFT: An Improved Blockchain Consensus Mechanism for Internet of Things Application. In Proceedings of the 2021 7th International Conference on Computer and Communications (ICCC), Chengdu, China, 10–13 December 2021; pp. 1520–1525.

20. Wang, Z.; Li, T.; Wang, H.; Shao, A.; Bai, Y.; Cai, S.; Xu, Z.; Wang, D. An Erasure-coding-supported Version of Raft for Reducing Storage Cost and Network Cost. In Proceedings of the 18th USENIX Conference on File and Storage Technologies (FAST 20), Santa Clara, CA, USA, 25–27 February 2020; pp. 297–308.
21. Tan, D.; Hu, J.; Wang, J. VBBFT-Raft: An understandable blockchain consensus protocol with high performance. In Proceedings of the 2019 IEEE 7th International Conference on Computer Science and Network Technology (ICCSNT), Dalian, China, 19–20 October 2019; pp. 111–115.
22. Wang, R.; Zhang, L.; Xu, Q.; Zhou, H. K-Bucket based Raft-like consensus algorithm for permissioned blockchain. In Proceedings of the 2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS), Tianjin, China, 4–6 December 2019; pp. 996–999.
23. Zhang, S.; Lee, J.H. A group signature and authentication scheme for blockchain-based mobile-edge computing. *IEEE Internet Things J.* **2019**, *7*, 4557–4565. [CrossRef]