electronics

MDPI

Article

P-Raft: An Efficient and Robust Consensus Mechanism for Consortium Blockchains

Shaofei Lu [1,2,*], Xuyang Zhang [1], Renke Zhao [1,2], Lizhi Chen [1], Junyi Li [1,2] and Guanzhong Yang [1,2]

[1] College of Computer Science and Electronic Engineering, Hunan University, Changsha 410082, China; zhangxuyang@hnu.edu.cn (X.Z.); zzzk@hnu.edu.cn (R.Z.); chenlzs@hnu.edu.cn (L.C.); junyilee@hnu.edu.cn (J.L.); gzyang@hnu.edu.cn (G.Y.)
[2] Hunan Provincial Key Laboratory of Blockchain Infrastructure and Application, Changsha 410082, China
* Correspondence: sflu@hnu.edu.cn

Abstract: With the rise in blockchain technology, consortium blockchains have garnered increasing attention in practical applications due to their decentralization and immutability. However, the performance of current consortium blockchains remains a significant obstacle to large-scale commercial adoption. The consensus algorithm, as a fundamental component of blockchain technology, plays a critical role in ensuring both security and efficiency. Unfortunately, most existing consensus algorithms for consortium blockchains are vote-based consensus algorithms, and the performance of vote-based consensus algorithms is largely limited by the performance of the leader node. Therefore, we present P-Raft: a high-performance consensus algorithm that builds upon the Raft algorithm and leverages node server performance evaluations. The primary objectives of this article included enhancing the efficiency of Leader processing, promoting the utilization of the consortium blockchain, and ensuring the robustness of Leader election. Specifically designed to meet the service requirements of consortium blockchain's consensus mechanism, the P-Raft incorporated the Yasa model, which evaluated the instant machine performance of each node. The performance of each node is then associated with the election timeout, ensuring that nodes with superior performance are more likely to be chosen as Leaders. Additionally, we implemented a leader verification mechanism based on the Bohen-Lynn-Shacham (BLS) signature, which prevented malicious Byzantine nodes from becoming Leaders without receiving enough votes. Empirical findings show that the P-Raft can swiftly designate high-performing nodes as Leaders, thereby greatly improving service efficiency in the consensus process and the overall performance of the consensus mechanism. Ultimately, P-Raft is better equipped to meet the demands of consortium blockchain applications for large-scale transactions.

Keywords: blockchain; consensus mechanism; performance evaluation; Raft

Citation: Lu, S.; Zhang, X.; Zhao, R.; Chen, L.; Li, J.; Yang, G. P-Raft: An Efficient and Robust Consensus Mechanism for Consortium Blockchains. *Electronics* **2023**, *12*, 2271. https://doi.org/10.3390/electronics12102271

Academic Editor: Mehdi Sookhak

Received: 24 April 2023
Revised: 13 May 2023
Accepted: 16 May 2023
Published: 17 May 2023

1. Introduction

As the underlying technical support of many digital currencies, blockchain has attracted wide attention in recent years. The essence of blockchain is a chain data structure in which each node in the network operates data in a distributed environment with the help of cryptography, a consensus algorithm, and a smart contract. As a new distributed computing paradigm, blockchain provides a decentralized, immutable, transparent, and traceable distributed database solution [1].

However, in the blockchain system, the number of nodes engaged in accounting was substantial, and there was distrust between the nodes. In this circumstance, the consensus algorithm adopted by the blockchain system needed to have a Byzantine fault tolerance for errors [2]. According to the different scales and power of nodes in the blockchain, Blockchain can be broadly classified into three distinct types: public blockchains, consortium blockchains, and private blockchains [3,4]. Consortium blockchains only allow

trusted and authorized entities to join the consortium blockchains, which helps to improve the overall stability and safety of the consortium chain.

The consensus algorithm assumed a paramount role within the blockchain system because it could solve the problem of how a single node in a blockchain network achieved message consistency [5]. In the blockchain system, the consensus algorithm assumed a paramount role. The Practical Byzantine Fault Tolerance (PBFT) was frequently employed in consortium blockchains, but non-Byzantine Fault Tolerance mechanisms such as Raft and PoET were also viable options. The Raft algorithm allows multiple nodes to work together as a cluster, and when some nodes exit or fail, the cluster can still reach a consensus quickly. Most existing consensus algorithms for consortium blockchains are vote-based consensus algorithms, and the performance of vote-based consensus algorithms is largely limited by the performance of the leader node. This is because these consensus algorithms mainly rely on the leader node to initiate the consensus process. Once the performance of the leader node is poor or it becomes a Byzantine malicious node, it results in the low performance of the consortium blockchain or an unsafe consensus.

The Raft algorithm sets three roles: Leader, Follower, and Candidate. The Raft achieves consistency by selecting a Leader and then letting it manage the replication log with full responsibility. The Leader is responsible for handling written requests, managing log replication, and constantly sending heartbeat messages, meaning that a Leader with better machine performance and stability can make the consensus more efficient. In the Raft algorithm, each node had the same possibility of being the Leader, meaning that the Raft algorithm election could not guarantee that the performance of the Leader was the best in the blockchain network.

The Raft algorithm was easy to implement while being highly understandable. However, Raft is not suitable for Byzantine fault-tolerant environments: Raft assumes that nodes participating in the consensus only have a Fail-stop Failure. However, the election of Leader nodes should take into account the impact of network attacks and malicious nodes. In scenarios where consortium blockchains are deployed on public networks or in sensitive applications such as finance, it is necessary to consider Byzantine fault tolerance. Although PBFT can achieve Byzantine fault tolerance, its communication complexity is higher, and its throughput is not as good as that of the Raft algorithm.

The primary objectives of this article include enhancing the efficiency of Leader processing, promoting the utilization of consortium blockchain, and ensuring the robustness of Leader election. To address these aforementioned issues, the present article introduced a novel consensus algorithm, P-Raft, which offered the following key contributions:

- We introduced an enhanced election algorithm that assessed the performance of individual nodes. The election algorithm evaluated the performance of nodes. It could correctly select the node with the best machine performance as the Leader and satisfy the data consistency and availability.
- We proposed a leader verification mechanism based on the BLS signature to prevent the malicious Byzantine node from becoming the Leader during the election by pretending that it had received enough votes.

The subsequent sections of this article are structured as follows. Section 2 delves into the related work. Section 3 presents the proposed consensus algorithm, P-Raft. Section 4 showcases the experimental results. Section 5 provides a summary of this article.

2. Related Work

Unlike the public blockchain, the consortium blockchain only allows trusted and authorized entities to proceed through the activities within the consortium blockchain [6]. This means that a new member needs to join a consortium blockchain with the permission of the consortium, which is usually given by certification authorities. In this way, members of the consortium blockchain are actual participants to a certain extent, and the overall stability and safety of the consortium chain are improved.

Raft is a voting-based consensus algorithm [7]. It sets a Leader in the cluster, and the Leader needs to completely accomplish the consensus. However, when multiple Candidates send vote requests, each Candidate can easily fail to obtain majority votes, leading to numerous rounds of Leader elections. While Raft applies a randomized election timeout mechanism to limit the multiple Candidates broadcasting voting requests simultaneously in an election, election conflicts inevitably occur when Candidate nodes are shut down or the network is delayed. In addition, the Leader of Raft is responsible for synchronizing the state of the entire cluster and handling external events. A Leader with poor performance negatively affects the consensus efficiency of the whole network.

Recently, some other researchers have concentrated on enhancing the capability of Raft. The Kraft consensus algorithm, as presented in Reference [8], is rooted in the Kademlia protocol [9]. Kraft utilizes K-Buckets to optimize the process of leader election and successfully mitigates the challenge of vote splitting in the Raft algorithm. It has higher stability and reduces the negative effects on Leader election efficiency. However, the process of electing a Leader in Kraft still relies on the generation of a random number. This means that the selected Leader may exhibit a subpar performance, ultimately hindering the consensus efficiency of the entire cluster. The CRaft consensus algorithm [7], which integrates Raft and credit models, has been proposed to assess the nodes according to their transactions. However, the handling ability of CRaft for network partition was relatively weak: in the event of a network partition, the algorithm may result in multiple partitions where nodes believe themselves to be the primary node, ultimately leading to the failure of reaching a consensus. Wu, YS. et al. [10] proposed a consensus algorithm that incorporated node activity with Raft. By utilizing the weighted PageRank algorithm, it assigned distinct PR values, thereby mitigating the influence of malevolent nodes on the overall network consensus to a significant extent. However, honest nodes need to spend a considerable amount of time to acquire a high PR value, which is not friendly to the honest nodes that join later.

The Raft algorithm should be implemented in an ideal environment that does not tolerate Byzantine nodes, which means that it is not suitable for consortium blockchains. To solve this problem, some researchers have conducted research on Raft-like algorithms that tolerant Byzantine faults.

Sihan T [11] proposed a consensus algorithm that was based on the Schnorr signature. It requires the client to sign the message before sending it to the leader in order to thwart any potential interference from the Byzantine leader; measures were taken to safeguard the logs. However, the algorithm's reliance on frequent communication between the nodes made it vulnerable to network latency and instability, rendering it less effective in unstable network environments. THCM [12], proposed by Jiang X based on trust evaluation, divided the nodes in the network into multiple layers. The nodes within each layer utilized the enhanced Raft algorithm, while the nodes situated between layers employed the PBFT algorithm, thus effectively circumventing the Byzantine issue. However, its structure and trust evaluation model made it quite complex, which could make it difficult to understand and implement.

The above researchers enhanced the efficiency of Raft by evaluating the behavior of the nodes. However, few researchers have made good use of the performance of the nodes to improve the efficiency of Raft.

3. Materials and Methods

3.1. Raft Algorithm

The Raft consensus algorithm employs a heartbeat mechanism to initiate the process of leader election. In the Raft consensus algorithm, if the Follower fails to receive Leader's heartbeat message within a designated time frame, it transitions into a Candidate state and initiates the Leader election process. The Leader election process of Raft is illustrated in Figure 1.

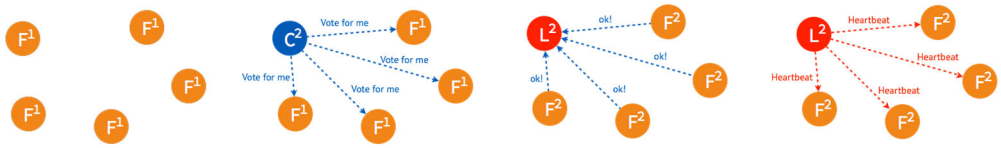

Figure 1. The election process of Raft.

The Follower (F^1 in Figure 1) votes for the Candidate (C^2 in Figure 1) who first sent the voting request. If none of the Candidates obtained majority votes, a new Leader election would be started. The Candidate that received the majority of votes became the Leader (L^2 in Figure 1) and demonstrate its leadership by sending a heartbeat message to the other nodes in the cluster. The other nodes will turn into Follower (F^2 in Figure 1) after received the heartbeat message.

3.2. P-Raft Consensus Mechanism

In Raft's Leader elections, Follower nodes could only cast their votes for only one Candidate node in each election. Raft uses a randomized election timeout mechanism to prevent multiple Candidates from starting the election at the same time. However, it cannot guarantee that the performance of the node is elected as the Leader.

We propose a consensus algorithm called P-Raft based on performance evaluation. It added a performance score to each node participating in the consensus. The higher the performance score was, the easier the node would find being elected as the Leader.

The intricate process of Leader election in P-Raft is expounded below and illustrated in Figure 2.

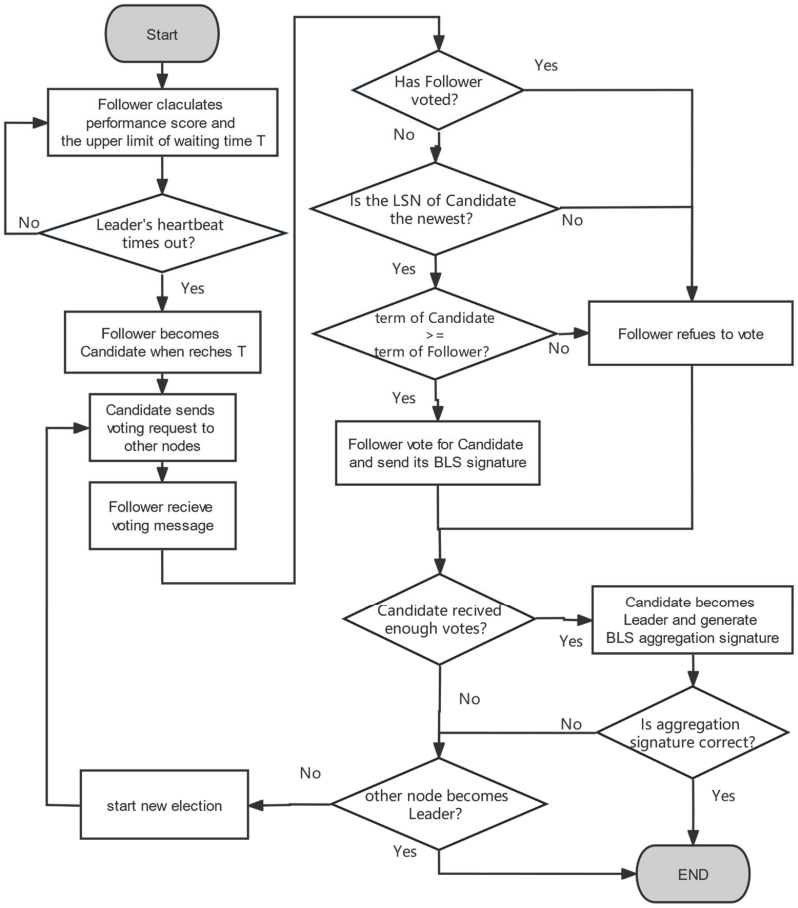

Figure 2. Process of Leader election of P-Raft.

(1) If a Follower node fails to receive any heartbeat message from the Leader in the set period, it reads its real-time performance and calculates the current performance score according to the Yasa model. Based on the performance score, it calculates the upper limit of the waiting time for Leader election T, which indicates how long each node has to think about whether to become the Leader or not. As the performance score increases, the Follower is prompted to swiftly transition into a Candidate and commence the election process.

The formula for calculating the upper limit of waiting time T based on the node performance score is as follows:

$$T = T_{max} \times \gamma(1 - C), \quad (1)$$

T_{max} is the maximum value of the upper limit time T, and γ is the gain parameter. In this paper, we set $\gamma = 1$.

(2) After waiting for T, the Follower node initiated an election, increased the current term, and changed the node's identity as a Candidate.

(3) The Candidate sent voting requests to the other nodes. The node that received the voting request compared its term with that of the request sender. If the sender's term was higher than its own, the node withdrew from the election and assumed the role of a Follower. When the received term of all other nodes was lower than the term itself, the node voted for itself. Finally, the Leader was determined by the number of received votes.

(4) Once the Leader was confirmed, the confirmation message was sent to other nodes. The Follower verifies the Leader when it receives messages from the Leader. After the main selection process is over, other nodes switch to the follower state. The process of Leader verification is introduced in Section 3.4.

3.3. Evaluation Model of Node's Performance: Yasa Model

To evaluate the comprehensive performance of each node, we proposed a node performance evaluation model: Yasa Model. This evaluated the current performance of the node's server and provided a score. The performance score of each node consisted of the machine performance score and stability score. The calculation formula for the node performance score was as follows:

$$C = w_m \times C_m + w_s \times C_s, \quad (2)$$

where C_m is the node's machine performance score, C_s is the node's stability score, w_m is the weight of C_m, and w_s is the weight of C_s. The value of w_m and w_s can be decided by the actual application requirements. As consortium chain environments typically use relatively stable servers, the performance differences resulting from the assigned values of w_m and w_s had little impact on the election priority sequence. Therefore, for experimental convenience, both values were set to 0.5 in this paper. However, if unstable servers were used for nodes in the consortium chain, stability scores should be given greater consideration. Users can adjust the values of w_m and w_s according to their actual needs by modifying the configuration file.

In most of the previous studies, the utilization of system hardware resources has been selected as the load index, and the most common indicators were CPU utilization and memory utilization. These two indicators could directly reflect the system's performance, so they must be considered. However, considering that there are certain differences in the processing capabilities of cluster servers for different requests in the scenario of high concurrency, it is limited to measure the load performance of servers by only one or several load indicators. In many cases, users need to evaluate node server performance from a holistic perspective. The proposed node performance evaluation model is needed to balance different demands.

Therefore, we proposed the following calculation model for machine performance scoring:

Each node reads the server's CPU idle rate, memory idle rate, GPU idle rate, and network bandwidth idle rate into the following matrix X:

$$X = \{c_i,\ m_i,\ g_i,\ n_i\}, \tag{3}$$

where c_i represents the CPU idle rate, m_i represents memory idle rate, g_i represents GPU idle rate, and n_i represents network bandwidth idle rate.

The formula of C_m is as follows:

$$C_m = \sum_{j=1}^{4} (w_j \times x_{1j}), \tag{4}$$

In previous research, weight coefficients were usually determined by empirical assignments and a lack of necessary mathematical analysis and quantitative basis. In order to enhance the accuracy and reliability of the performance score calculation, this paper selected the Analytic Hierarchy Process (AHP) [13] to calculate the weight coefficients scientifically.

(1) Theoretical basis [13]

Definition 1. *Assume a matrix* $Q = (q_{ij})_{n\times n} (i, j \in (1, 2, \ldots, n))$*, if Q satisfies: (1)* $q_{ij} > 0$*; (2)* $q_{ii} = 1$*; (3)* $q_{ij} = 1/q_{ji}$*; then, the matrix can be called a positive reciprocal matrix.*

Definition 2. *Assume a positive reciprocal matrix* $Q = (q_{ij})_{n\times n}$*, if Q satisfies* $q_{ij} = q_{ik}/q_{jk} (i, j \in (1, 2, \ldots, n))$*, then the matrix Q can be called a consistent matrix.*

Theorem 1. *An n-order positive reciprocal matrix Q is a congruent matrix if and only if its largest characteristic root* $\lambda_{max} = n$*. It can be assumed that there are n load indicators in the system, and the weight coefficients of each indicator are, respectively expressed as* $M_1, M_2, \ldots, M_n$*, then the weight coefficient vector is* $M = (M_1, M_2, \ldots, M_n)^T$*. Comparing the weight coefficients in pairs then obtains the ratio matrix Q*:

$$Q = \begin{bmatrix} M_1/M_1 & M_1/M_2 & \cdots & M_1/M_n \\ M_2/M_1 & M_2/M_2 & \cdots & M_2/M_n \\ \vdots & \vdots & & \vdots \\ M_n/M_1 & M_n/M_2 & \cdots & M_n/M_n \end{bmatrix} = \left(q_{ij}\right)_{n\times n}, \tag{5}$$

According to the matrix Q, $q_{ii} = 1$, $q_{ij} = 1/q_{ji}$, $q_{ij} = q_{ik}/q_{jk}$, we can then continue to obtain:

$$QM = \begin{bmatrix} M_1/M_1 & M_1/M_2 & \cdots & M_1/M_n \\ M_2/M_1 & M_2/M_2 & \cdots & M_2/M_n \\ \vdots & \vdots & & \vdots \\ M_n/M_1 & M_n/M_2 & \cdots & M_n/M_n \end{bmatrix} \begin{bmatrix} M_1 \\ M_2 \\ \vdots \\ M_n \end{bmatrix} = \begin{bmatrix} nM_1 \\ nM_2 \\ \vdots \\ nM_n \end{bmatrix} = nM, \tag{6}$$

where n is the characteristic root of matrix Q, and is the largest characteristic root, denoted as λ_{max}. M is the eigenvector of the matrix and the weight coefficient vector of each load index that needs to be calculated.

(2) Calculation of weight coefficient vector

First, we need to build the analytic hierarchy model. Take machine performance evaluation as the goal level, the CPU idle rate, Memory idle rate, network bandwidth idle rate, and GPU idle rate can be taken as the criteria level.

We employed the "1–9" rating scale to assess the significance of each load metric to construct the judgment matrix Q when combined with the server test situation. During the actual testing, it was observed that the rates of CPU, Memory, and GPU idling were relatively high, while the network bandwidth was relatively low. The comparison results obtained based on the test of the server and the "1–9" scaling method are shown in Table 1, which represents the load evaluation index.

Table 1. The impact of server resources on distributed system performance.

Q	CPU	GPU	Memory	Net
CPU	1	1	1	3
GPU	1	1	1	3
Memory	1	1	1	3
Net	$\frac{1}{3}$	$\frac{1}{3}$	$\frac{1}{3}$	1

Therefore, the judgment matrix can be written as:

$$Q = \begin{bmatrix} 1 & 1 & 1 & 3 \\ 1 & 1 & 1 & 3 \\ 1 & 1 & 1 & 3 \\ \frac{1}{3} & \frac{1}{3} & \frac{1}{3} & 1 \end{bmatrix}, \tag{7}$$

After the judgment matrix Q was constructed, it was added by columns and then normalized. Finally, rows were added and normalized by the last column elements from left to right to obtain its approximate feature vector M'.

$$M' = (0.3,\ 0.3,\ 0.3,\ 0.1)^T, \tag{8}$$

The maximum characteristic root of the judgment matrix Q could be obtained by calculating the approximate feature vector. The formula is as follows:

$$\lambda_{max} = \sum_{i=1}^{n} \frac{(QM)_i}{nM_i}, \tag{9}$$

$(QM)_i$ represents the i-th element of QM. According to the formula, the maximum characteristic root $\lambda_{max} = 4$, which is the order of the positive reciprocal matrix. According to Theorem 1, the matrix Q is the consistent matrix, so the approximate eigenvector M' is the eigenvector M. Therefore, the formula for calculating C_m is as follows:

$$C_m = \sum_{j=1}^{4} (w_j \times x_{1j}),\ (w_1 = 0.3, w_2 = 0.3, w_3 = 0.3, w_4 = 0.1), \tag{10}$$

In accordance with the formula, w_1 denotes the weight of the CPU, w_2 represents the weight of the GPU, w_3 signifies the weight of memory, and w_4 denotes the weight of the network bandwidth.

During the process of Raft consensus, all logs are committed to the Leader first. The Leader then sends the AppendEntries RPC synchronization log to the follower. If the Leader is unstable and fails frequently, log synchronization and client interaction may occur, resulting in consensus failure. Therefore, a scoring method to measure the stability of the node is needed. The calculation formula of C_s is as follows:

$$C_s = \frac{2}{1 + e^{\mu \times t_k}}, \tag{11}$$

t_k is the number of downtimes in the server within time k; μ is a gain parameter. A higher value of μ results in reduced system tolerance towards node downtime behavior. Figure 3 shows the process of updating the performance score.

3.4. Leader Verification Mechanism

When there is a Byzantine node, the election encounters the situation of forged voting messages. The Byzantine node may falsify that it has received voting messages from most other nodes during the election process. To prevent malicious nodes from forging their term and preempting the cluster Leader, we propose a leader verification mechanism that combines the threshold signature scheme with the BLS signature.

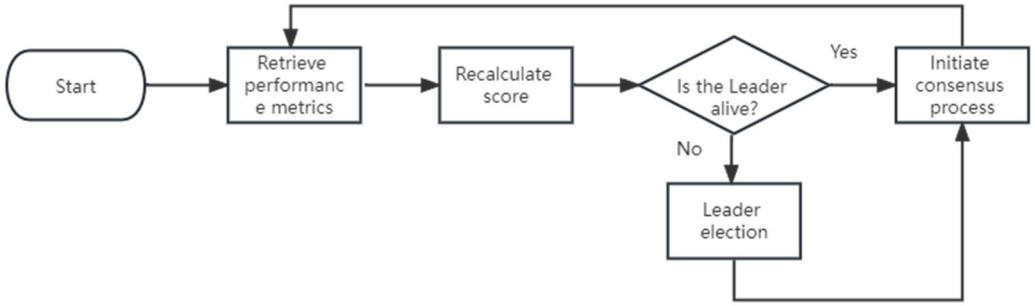

Figure 3. Process of performance score update.

The Boneh-lynn-shacham (BLS) signature algorithm was proposed by Boneh et al. [14]. The BLS signature could aggregate multiple signatures and m-n multiple signatures without generating a random number. BLS signature reduced the redundant communication overhead between nodes.

The process of leader verification is as follows:

Phase 1: A Candidate node receives partial BLS signatures from more than 2/3 of the Followers and generates BLS -aggregated signatures.

Phase 2: The Candidate node dispatches a message appended with the BLS aggregated signature and awaits the validation of the BLS aggregation signature by other nodes. If the verification process is successful, the Follower can provide positive feedback. However, if the verification process fails, negative feedback is given.

Phase 3: The Candidate node that receives 2/3 positive feedback turns into the Leader and sends a heartbeat message with positive feedback to other nodes to complete the election process.

When a Candidate or Follower who has lost contact with the Leader detects a new Leader, it first requests the Leader for the BLS aggregated signature generated during the election. The Leader is recognized when the aggregated signature is correct.

4. Results

This section compares the P-Raft algorithm with the Raft algorithm in terms of time consumption and the result of the Leader's election.

4.1. Experimental Environment

We incorporated P-Raft into Hyperledger Fabric, supplanting the former Raft module. Subsequently, we deployed and evaluated the enhanced Fabric platform within a service environment comprising Alibaba Cloud servers. These servers were categorized into five distinct types based on their respective configurations, and the total number of servers amounted to 30. Detailed server configurations are given in Table 2.

Table 2. Server configurations.

Server Type	A	B	C	D	E
vCPU	1	2	4	2	2
CPU Clock Speed/Turbo	2.5 GHz/ 3.2 GHz	-/3.5 GHz	-/3.5 GHz	2.5 GHz/2.7 GHz	2.5 GHz/-
Memory	2G	4G	8G	8G	8G
GPU	-	-	-	-	NVIDIAP4
Region	Hangzhou	Guangzhou	Guangzhou	Shanghai	Shenzhen
Amount	7	8	2	12	1
Network Bandwidth	50 Mbps				
OS	Ubuntu 16.04				
HyperLedger Fabric	v2.0.0				
Docker	1.8.2				

The construction of the node performance evaluation model and the consensus phase of both the P-Raft and Raft algorithms were implemented using Go1.17.1, and the P-Raft algorithm was deployed to HyperLedger Fabric to verify its effectiveness.

4.2. Changes in Nodes' Stability Score

C_s represents the stability of a node. The node with a higher score has higher reliability. Therefore, nodes with low C_s are not suitable for the Leader. This section tests unstable nodes.

If a node with an unstable status assumes the role of Leader in the cluster, it may result in a system failure, requiring the other nodes to initiate a new election for Leadership. Should a node experience frequent outages, the cluster will perceives it as unstable, and its score decreases, thereby reducing the likelihood of it being elected as the Leader and maintaining the cluster's stability.

In this experiment, an unstable node P was set in the blockchain network of 10 nodes. Within a certain period, after it was added to the cluster network, P went down for times, and other nodes could not receive the message from P. The impact of P's behavior on its stability score was controlled by the number of its downtime and the parameter μ. In this experiment, 4 different values of μ were set according to Formula (9). The initial reputation values of the nodes were all one, and the parameters μ ranged from 0.5 to 3.

As can be seen from Figure 4, when a node went down, its stability score gradually decreased, but the speed of reduction depended on the value of μ. It can be seen that the downtime behavior of the node has a great impact on the stability score of the node, which means that the node is not suitable to serve as the Leader.

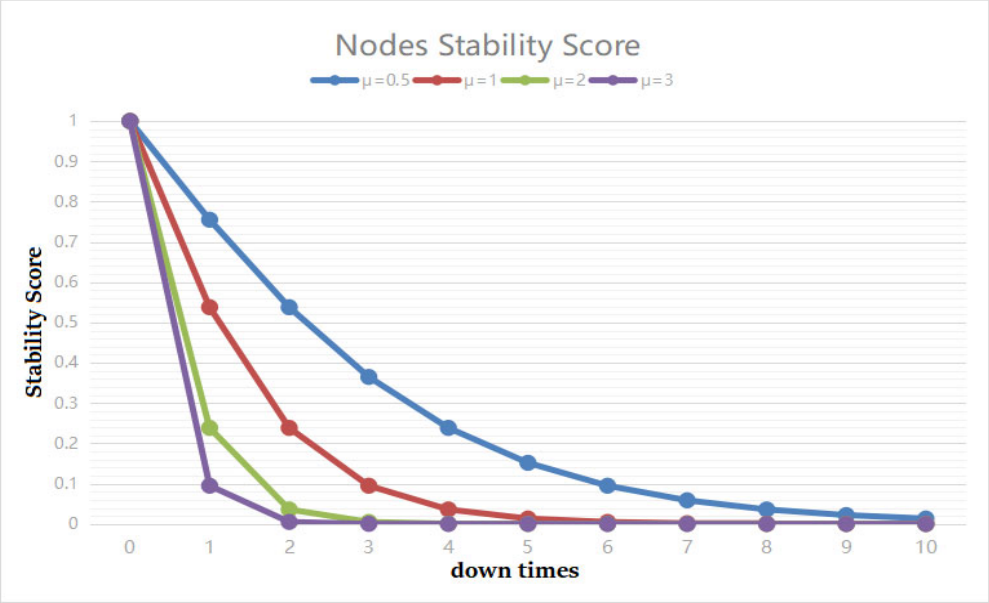

Figure 4. The changes in a node's stability score when μ is different.

4.3. Performance of P-Raft

4.3.1. Election Result

This section tested the election result of the Raft and the P-Raft algorithm when the size of cluster N in the blockchain network was 20 and 30. We ran the Leader election process 1000 times and collected the election results.

Table 3 depicts the election outcomes of both Raft and P-Raft algorithms when the number of nodes (N) was 30, which was made up of 7 A nodes, 8 B nodes, 2 C nodes, 12 D nodes, and 1 E node; this was the result when the number of nodes N was 20, which were made up of 7 A nodes, 7 B nodes, 2 C nodes, 2 D nodes, and 1 E node as shown in Table 4.

Table 3. Leader election result when N = 30.

Algorithms	A1	A2	A3	A4	A5	A6	A7	B1	B2	B3	B4	B5	B6	B7	B8	C1	C2	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12	E1
Raft	41	32	30	1	36	42	33	37	36	36	37	31	31	45	36	47	47	28	32	32	1	27	32	33	32	43	31	31	39	41
P-Raft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	591	0	0	0	0	0	0	0	0	0	0	0	0	409

Table 4. Leader election result when N = 20.

Algorithms	A1	A2	A3	A4	A5	A6	A7	B1	B2	B3	B4	B5	B6	B7	C1	C2	D1	D2	D3	E1
Raft	55	58	62	47	53	64	56	60	12	51	44	50	43	33	12	61	60	61	54	64
P-Raft	0	0	0	0	0	0	0	0	0	0	0	0	0	0	8	699	0	0	0	293

Table 3 shows that in the 1000 Leader election, node C2 was elected as Leader 591 times, and E1 was elected as Leader 409 times in the P-Raft algorithm; Table 4 shows that node C2 was elected as the Leader 699 times, and E1 was elected as the Leader 292 times in the P-Raft algorithm. This is because the election timeout of the P-Raft could be related to the server performance, and nodes with better performance had a shorter timeout. The election timeout of Raft is random, meaning that all the nodes had a chance to become the Leader.

4.3.2. Efficiency of Leader Election

In this section, we evaluated the latency of Leader election in both the Raft and P-Raft algorithms for N = 10, 20, and 30 nodes. We ran the Leader election process 1000 times and collected the average election latency. The result is shown in Figure 5:

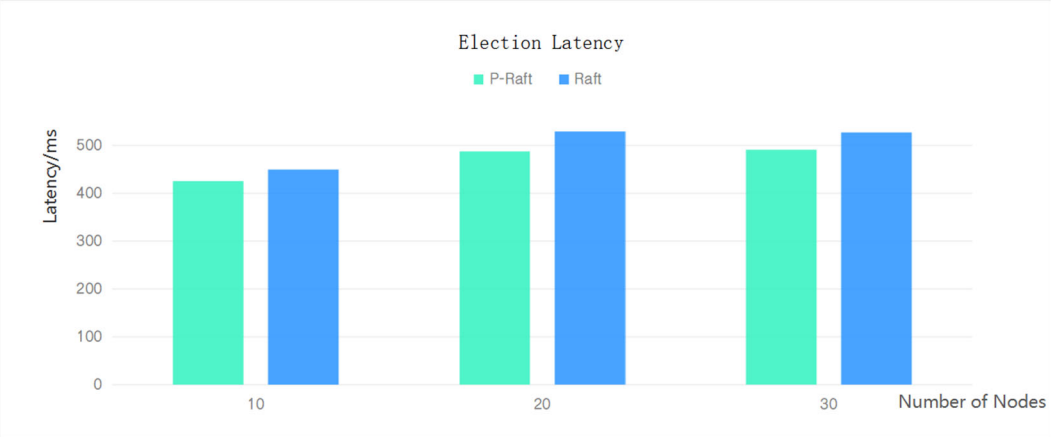

Figure 5. Latency of Leader election.

As depicted in Figure 5, P-Raft exhibited an election latency reduction of approximately 6% compared to Raft. In the P-Raft algorithm, the waiting time T is related to node performance, and the node with the best performance could always initiate the election first. However, in the Raft algorithm, the election delay was random because the value of waiting time T was random, relatively high values could be generated, and the average election delay of Raft was higher than that of P-Raft.

4.3.3. Byzantine Fault-Tolerance

This section aims to evaluate the resilience of the P-Raft algorithm against the interference and damage caused by malicious Byzantine nodes in the consensus cluster.

To achieve this goal, we conducted a comparison between the election results of the P-Raft algorithm and the Raft algorithm in a cluster of five nodes with one Byzantine node present. Specifically, the Byzantine node was initially transformed into a Leader and sent heartbeat messages to other nodes. We repeated the Leader election process 1000 times and collected the corresponding election results for analysis.

As depicted in Table 5, we considered a cluster composed of one A node, one B node, two C nodes, and one D node. Among them, C2 was the Byzantine node, and A1, B1, C1, and D1 were non-Byzantine nodes. We presented the election results of both algorithms under these conditions.

Table 5. Election result when Byzantine nodes were present.

Algorithms	A1	B1	C1	C2	D1
Raft	0	0	0	1000	0
P-Raft	0	0	1000	0	0

The results of the election in the presence of Byzantine nodes are depicted in Table 5. It can be observed that in 1000 leader elections, node C1 in the P-Raft algorithm was elected as the Leader 1000 times. By contrast, in the Raft algorithm, the Byzantine node C2 was elected as the Leader 1000 times. This discrepancy could be attributed to the fact that the P-Raft election process involved the collection of signatures from the nodes that had already cast their votes. Once the new Leader was elected, the collected signatures were presented to other nodes as proof. Raft, on the other hand, did not verify the Leader's signatures. This left Raft vulnerable to Byzantine nodes that manipulated the vote count to become the Leader. In such cases, P-Raft could detect forged votes by the number of signatures, whereas Raft failed to identify malicious behavior.

5. Conclusions

In this study, we proposed a consensus algorithm named P-Raft and a performance evaluation model called the Yasa model. P-Raft adds the attribute of the performance score to each node participating in the consensus. The higher the performance score is, the easier it will be for the node to be elected as the Leader. P-Raft also adds a leader verification mechanism based on the BLS signature to prevent the malicious Byzantine node from becoming Leader without receiving enough votes. Experimental results showed that it could elect the node with the best performance score among the available nodes as the Leader, and its latency of Leader in the election was less than the normal Raft algorithm, which reduced the probability of an invalid election caused by vote partitioning in Raft, reducing the average election cycle and time of the algorithm and enhancing the resilience of the blockchain network.

P-Raft can correctly select the node with good machine performance as the Leader and satisfy the data consistency and availability. It can also prevent the Byzantine node from being Leader, which is suitable for a consortium blockchain.

The accuracy and computational speed of the proposed Yasa model required further improvement. Additionally, the efficiency of the Leader verification mechanism in P-Raft required enhancement, and its Byzantine fault tolerance capability required further strengthening. Potential research directions included implementing advanced algorithms to enhance the precision and computational efficiency of performance evaluation, as well as adopting updated signature algorithms to improve the speed of both user signature and verification.

Author Contributions: Conceptualization, S.L., X.Z. and R.Z.; methodology, R.Z. and X.Z.; software, X.Z. and L.C.; validation, X.Z. and R.Z.; formal analysis, X.Z. and R.Z.; investigation, R.Z.; resources, S.L., R.Z. and L.C.; data curation, L.C.; writing—original draft preparation, X.Z. and R.Z.; writing—review and editing, S.L., X.Z. and R.Z.; visualization, X.Z.; supervision, S.L., G.Y. and J.L.; project administration, S.L. and R.Z. All authors have read and agreed to the published version of the manuscript.

Funding: This research was supported by the Special Funds for Construction of Innovative Provinces in Hunan Province of China, grant numbers 2020GK2016, 2020GK2006, and 2020GK2007.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

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

References

1. Cui, S.; Lu, Y.; Chang, X. Research on model of blockchain-enabled power carbon emission trade considering credit scoring mechanism. *Electr. Power Constr.* **2019**, *40*, 104–111.
2. Lamport, L.; Shostak, R.; Pease, M. The Byzantine generals problem. *ACM Trans. Program. Lang. Syst.* **1982**, *4*, 382–401. [CrossRef]
3. Zheng, Z.; Xie, S.; Dai, H.; Chen, X.; Wang, H. An overview of blockchain technology: Architecture, consensus, and future trends. In Proceedings of the 2017 IEEE International Congress on Big Data (BigData Congress), Honolulu, HI, USA, 25–30 June 2017; pp. 557–564.

4. Lu, S.; Pei, J.; Zhao, R.; Yu, X.; Zhang, X.; Li, J.; Yang, G. CCIO: A Cross-Chain Interoperability Approach for Consortium Blockchains Based on Oracle. *Sensors* **2023**, *23*, 1864. [CrossRef] [PubMed]
5. Guerrero-Sanchez, A.E.; Rivas-Araiza, E.A.; Gonzalez-Cordoba, J.L.; Toledano-Ayala, M.; Takacs, A. Blockchain Mechanism and Symmetric Encryption in A Wireless Sensor Network. *Sensors* **2020**, *20*, 2798. [CrossRef] [PubMed]
6. Singh, A.; Saha, R.; Conti, M.; Kumar, G. PoSC: Combined Score for Consensus in Internet-of-Things Applications. In Proceedings of the 2022 Fourth International Conference on Blockchain Computing and Applications (BCCA), San Antonio, TX, USA, 5–7 September 2022; pp. 173–180.
7. Chen, Y.; Liu, P.; Zhang, W. Raft consensus algorithm based on credit model in consortium blockchain. *Wuhan Univ. J. Nat. Sci.* **2020**, *2*.
8. 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.
9. Xu, H.; Zhang, L.; Liu, Y.; Cao, B. Raft based wireless blockchain networks in the presence of malicious jamming. *IEEE Wirel. Commun. Lett.* **2020**, *9*, 817–821. [CrossRef]
10. Wu, Y.S.; Wu, Y.S.; Liu, Y.R.; Shi, T.J. The research of the optimized solutions to Raft consensus algorithm based on a weighted PageRank algorithm. In Proceedings of the 2022 Asia Conference on Algorithms, Computing and Machine Learning, Hangzhou, China, 25–27 March 2022; pp. 784–789.
11. Tian, S.; Liu, Y.; Zhang, Y.; Zhao, Y. A byzantine fault-tolerant Raft algorithm combined with Schnorr signature. In Proceedings of the 2021 15th International Conference on Ubiquitous Information Management and Communication (IMCOM), Seoul, Republic of Korea, 4–6 January 2021; IEEE: New York, NY, USA, 2021; pp. 1–5.
12. Jiang, X.; Sun, A.; Sun, Y.; Luo, H.; Guizani, M. A Trust-Based Hierarchical Consensus Mechanism for Consortium Blockchain in Smart Grid. *Tsinghua Sci. Technol.* **2022**, *28*, 69–81. [CrossRef]
13. Saaty, T.L. Decision making with the analytic hierarchy process. *Int. J. Serv. Sci.* **2008**, *1*, 83–98. [CrossRef]
14. Boneh, D.; Lynn, B.; Shacham, H. Short signatures from the Weil pairing. In Proceedings of the 2001 International Conference on the Theory and Application of Cryptology and Information Security, LNCS 2248, Gold Coast, Australia, 9–13 December 2001; Springer: Berlin/Heidelberg, Germany; pp. 514–532.