IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 21 December 2023, accepted 4 January 2024, date of publication 11 January 2024, date of current version 23 January 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3352605

RESEARCH ARTICLE

An Efficient and Scalable Byzantine Fault-Tolerant Consensus Mechanism Based on Credit Scoring and Aggregated Signatures

SHIHUA TONG[1], JIBING LI[2], AND WEI FU[2]
[1]Chongqing College of Electronic Engineering, Chongqing 401331, China
[2]College of Automation, Chongqing University of Posts and Telecommunications, Chongqing 400065, China

Corresponding author: Jibing Li (s210331046@stu.cqupt.edu.cn)

This work was supported in part by the Chongqing Colleges and Universities Innovation Research Group Project under Grant CXQT21031.

ABSTRACT Practical Byzantine Fault Tolerance (PBFT), a classic consensus algorithm in blockchain technology, is extensively used in consortium blockchain networks. However, it is challenged by issues such as low consensus efficiency, poor scalability, inability to guarantee throughput with large-scale node access, and complex communication processes. To solve these problems, this paper proposes an improved PBFT consensus mechanism based on credit scoring and aggregated signatures, i.e., the CA-PBFT algorithm. First, the algorithm designs the node credit scoring mechanism, adds the coordination node in the original algorithm model, stipulates the node state and functional limitations, and realizes the dynamic joining and exiting of the nodes, to solve the low efficiency of the PBFT algorithm during the consensus process and the problem of not supporting the dynamic joining and exiting of the nodes; at the same time, the signature scheme based on the BLS aggregated signature is designed, which reduces the length of the signature and simplifies the signing process, to solve the problem of the node's signature taking up too much space during the consensus process, which affects the efficiency of the signature validation as well as the efficiency of the signature construction. Experimental results show that this consensus mechanism enables an efficient, secure, and scalable consensus process with low resource and computational costs.

INDEX TERMS Blockchain, consensus mechanisms, PBFT, credit scoring, aggregated signatures.

I. INTRODUCTION

Since the official release of the Bitcoin system [1] in 2008, blockchain, based on blockchain technology, has attracted increasing attention from researchers. Blockchain technology is developed on the technologies of economics, cryptography, and computer science [2], which can realize multi-party peer-to-peer trustworthy information transmission under the system environment without a trusted center, and its applications in the fields of smart healthcare [3], digital finance [4], Internet of Things [5], and supply chain management [6], [7], [8] have been widely researched.

The associate editor coordinating the review of this manuscript and approving it for publication was Cong Pu.

Blockchain technology, as an emerging distributed bookkeeping technology, can be mainly divided into three types: public chain, private chain, and consortium chain, based on the different environments and authority management methods of its application. A public chain is a completely open and decentralized blockchain system, in which anyone can participate freely, for example, Bitcoin is a typical representative of a public chain. In a public chain, all nodes are free to join or leave, and the reading and writing of data is done mainly through transactions or mining. The public chain is highly transparent, safe, and reliable, but its transaction speed and processing efficiency are relatively low due to its decentralized nature. A private chain, which is deployed and operated by an individual or private organization,

uses blockchain technology for its underlying bookkeeping, but bookkeeping access is limited to specific participants. A private chain is more centralized than a public chain and has the advantages of controlled access, fast transaction speed, and low cost, but its limitation is that it is less trustworthy because it is more centralized. A consortium chain, which sits between public and private chains, is a type of blockchain that applies to consortium organizations. A consortium chain is an organization with a large number of members. Compared with private and public chains, a federated chain has higher processing efficiency and lower cost while maintaining partially decentralized characteristics. Table 1 presents a comparison of the three types of blockchains.

To better solve the problem of blockchain landing application, the focus of blockchain research has gradually shifted to consensus algorithms, which, as the most critical link in the blockchain, is a protocol that ensures that the nodes on the chain are synchronized with each other and determine whether a transaction is legal and generate a block. More and more consensus algorithms appear enthusiastically, the Proof of Work (PoW) [9] consensus algorithm in the public chain can achieve consistency confirmation in the case of large-scale node access; the Proof of Stake (PoS) [10] algorithm adds the concept of tokens based on PoW algorithm, which improves the efficiency of the consensus in the blockchain; DPoS [11], as an improved scheme of PoS, achieves better performance and higher fault tolerance by decentralizing decision-making to coin holders; the Raft [12] algorithm proposes a strong leader consensus concept, which greatly improves the consensus efficiency in the non-Byzantine fault-tolerant domain; the Practical Byzantine Fault Tolerance (PBFT) [13] consensus algorithm is aimed at the Byzantine model, which realizes that even if there is a 33.3% of evil nodes in the consensus network, the consensus can be guaranteed to be completed. Table 2 presents a comparison of the classical consensus algorithms mentioned above.

PBFT is a consensus algorithm to address the presence of malicious nodes in distributed systems [14]. In this algorithm, each node sends its proposal request to other nodes, who will vote according to their own rules, and the consensus is finally reached when a certain number of votes are cast. If there are malicious nodes in the current consensus network, they may send different proposals or deceive other nodes, thus destroying the consensus result. The basic idea of PBFT is to reach a consensus by eliminating the influence of malicious nodes on the consensus network through multiple rounds of voting [15]. In each round of voting, nodes will send their proposals to other nodes and collect proposals from other nodes. Then nodes will vote according to certain rules and send the voting results to other nodes. The nodes will keep repeating this process until the voting results of all nodes are the same, the specific algorithm flow is shown in Figure 1. The BFT algorithm originally proposed by Lamport et al. [16] can only address faults in distributed

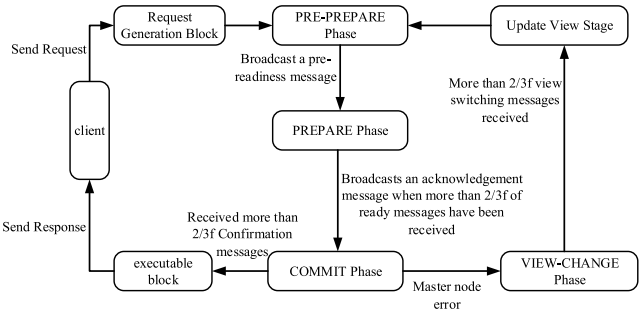

FIGURE 1. PBFT consensus process.

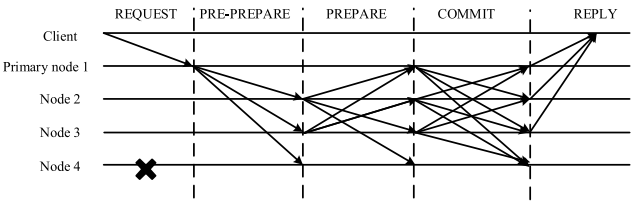

FIGURE 2. The process of the PBFT algorithm.

environments, does not consider practical algorithmic run-time performance, and requires exponential algorithmic time complexity $O(n^2)$.

As shown in Figure 2, the PBFT algorithm needs to go through a total of five phases, which are the *REQUEST* phase and the $PRE-PREPARE$ phase, the *PREPARE* phase, the *COMMIT* phase, and also the *REPLY* phase, where the peer-to-peer (P2P) communication will be initiated at the *PREPARE* node and the *COMMIT* phase. Typically, a PBFT system needs to be deployed to at least $3f + 1$ nodes, and can tolerate at most f malicious nodes with Byzantine failures, and the whole system state is determined by $2f + 1$ of these nodes. Compared to non-Byzantine systems, their main problems are the longer time taken by the consensus algorithm to determine the state, the more node resources it takes up, and the possibility of errors accumulating over a long period in the application leading to system crashes.

Compared to other consensus algorithms, the PBFT consensus algorithm is considered ideal for federation chains due to its advantages in security, efficiency, controllability, and real-time performance. However, when there are too many Byzantine nodes in the consensus network, even if it does not affect the completion of the whole consensus process, it will affect its consensus efficiency. At the same time, since the consensus network of PBFT is static, it does not support the dynamic change of nodes. If you want to join a new node or quit a node, you need to reset the consensus network, which is very inconvenient.

Aiming at the problems of low consensus efficiency of the PBFT consensus algorithm, poor scalability, node signature message structure occupying too much space, and difficulty in applying to large-scale node access, an improved PBFT consensus algorithm based on credit scoring and aggregated

TABLE 1. Comparison of three types of blockchain.

Blockchain Category	Node Authority	Decentralization	Access Mechanism	Billed Users	Transaction Efficiency	Application Scenario
Public Chain	Full Permissions	Full Decentralization	Unlimited	Arbitrarily	Low	Bitcoin Ethereum
Private Chain	Limit	Full Centralization	Authorization required	Authorized Personnel	Highest	Inside the company
Consortium Chain	Read, Trade	Semi-decentralized	Authorization required	Authorized Personnel	Higher	Hyperleger Fabric

TABLE 2. Comparison of different consensus algorithms.

Consensus Algorithm	Throughput	Fault Tolerance	Transaction Confirmation Delays	Adaptation Type
PoW	Low	$N = 2f + 1$	High	Public Chain
PoS	Low	$N = 2f + 1$	High	Public Chain/Private Chain
DPoS	Low	$N = 2f + 1$	High	Public Chain/Private Chain
Raft	High	/	Low	Consortium Chain
PBFT	Higher	$N = 3f + 1$	Low	Consortium Chain

signatures is proposed. The contributions of this paper are as follows:

1) Design node credit scoring rules and node dynamic join and node dynamic exit processes. Through the node credit scoring rules to participate in the consensus process node credit status evaluation, to avoid the evil nodes affecting the consensus process, and effectively improve the efficiency of consensus. In the original consensus model, a coordination node is added, which is responsible for realizing the function of dynamic joining and exiting of nodes, and at the same time combining with the node credit scoring mechanism to eliminate the evil nodes in the consensus network.
2) The implementation of a BLS-based aggregated signatures scheme reduces the size of the signature within the node block during the consensus process, shortens the signature time, and acts on the *PRE − PREPARE* phase and *COMMIT* phase to reduce the number of times their signatures are verified.

The rest of the paper is organized as follows. Section II presents the related work. Section III presents the detailed design scheme of the CA-PBFT algorithm. Section IV experimentally analyzes the security and performance of the algorithm. Section V summarizes this study and discusses future research directions.

II. RELATED WORK

Consensus algorithms, as a core part of the blockchain, can determine the nature of the blockchain and can enable a foundation of trust to be established between different nodes in the blockchain. To target the malicious nodes in the consensus process, the PBFT algorithm appeared, which was proposed by Castro et al. [17], the algorithm can still guarantee the normal operation of the system in the face of the existence of about 33% of the evil nodes in the blockchain system, but it can't guarantee the efficiency of the consensus when large-scale nodes access the consensus network. In response to some of the problems arising from PBFT, such as the Byzantine general problem [16], research scholars have conducted a large number of studies and improved it. For example, Xu et al. for the PBFT algorithm in which the primary node selection is unclear and the communication complexity is too high, put forward a fuzzy-set-based improved algorithm VS-PBFT, redistribute the consensus process, use a consistent hash class consensus algorithm to partition the nodes of the whole network, and in each partition to make the primary node selection to complete the global consensus, but there will be many limitations in practical applications, and the effect needs to be verified in real-world environments [18]. Wang et al. proposed an improved creditworthiness-based PBFT consensus algorithm (CPBFT), which introduces a credit rating and a credit coefficient so that the probability of each node being selected as a primary node is affected by its past behaviors, and it is more likely that a reliable primary node will be selected, but the number of nodes participating in the consensus is too many, which leads to too much complexity in communication [19]. Gan et al. combined the characteristics of the CIOT with the PBFT algorithm to propose an EIOT-PBFT multi-stage consensus algorithm, which divides its consensus process into three stages: the grouping stage, and the scoring stage, the consensus reaching the stage. At the same time, to reduce the probability of view changes, a single primary node is designed in the form of a primary node group [20]. Lao et al. proposed a scalable location-based consensus protocol, G-PBFT, which achieves high consistency efficiency, low network overhead, and high scalability through a location-based signer election and era-switching mechanism [21].

In addition, Lei et al. proposed a weighted PBFT cross-chain algorithm improvement for the problems of low fault tolerance, low throughput, and high latency of the traditional PBFT method, constructed a cluster-centered blockchain-based cross-chain exchange model, changed the role of the original consensus node, and proposed three new node types, which are consensus service node and cross-chain exchange node and application node [22]. Li et al. proposed a new protocol Gosig, which designs a new consensus protocol as well as the underlying Gosig network, using a combination of transmission and aggregate signature Gossip to achieve improved data transmission [23]. Na et al. proposed an improved algorithm based on dual-primary nodes (DPNPBFT) for the problem that PBFT is unsuitable for large-scale node access. This algorithm sets two primary

nodes to check and supervise each other to avoid the centralization problem, and also reduces the communication complexity of replica nodes, and has a higher fault tolerance rate compared to the PBFT consensus algorithm [24]. Yang et al. proposed a high fault-tolerant consensus algorithm NBFT in response to the problem of ignoring the fault-tolerance and democracy in the PBFT improvement algorithm, which follows the blockchain decentralization and democratization principles, using a consistent hash algorithm to group consensus nodes, avoiding too much communication between nodes to waste communication resources, reducing communication complexity, and improving the expandability of the network, at the same time, for the problem of fault-tolerance ability of group consensus, node decision broadcasting model and threshold counting model are proposed, and the nodes are subjected to a joint failure analysis through the two models, which makes the algorithm fault-tolerant upper limit is greater than 1/3 [25]. Jiang et al. proposed a TB-PBFT algorithm to reduce the probability of a malicious node being selected as a primary node and improve the reliability of the primary node by utilizing a comprehensive evaluation model of the coalition chain. However, the failure-resistant performance of the algorithm is affected when most of the primary nodes are malignant nodes [26]. Zhang et al. proposed a DBFT protocol based on a dual-response mechanism, which allows replica nodes to respond deterministically to the client twice, allowing for graceful performance degradation in case of failure. While the dual response mechanism still ensures good performance, at the same time it can lead to inconsistency in speculative execution [27]. He et al. proposed a PBFT based on a reputation mechanism, which reduces the risk of a malicious node being selected as a primary node through a reputation scoring mechanism and a supervisory node mechanism, thus improving the reliability of the primary node. However, there are some drawbacks to this approach, such as the risk of centralization and the possibility of adding additional system overhead [28].

III. CA-PBFT CONSENSUS ALGORITHM DESIGN

A. BLS AGGREGATED SIGNATURE SCHEME

The traditional PBFT consensus algorithm mainly consists of three phases: $PRE-PREPARE$, $PREPARE$ and $COMMIT$, in which the main process of the $PRE-PREPARE$ phase is that the primary node $PriNode$ transmits and packages the order of the transactions, as well as the message, to each replica node $RepNode$ in the consensus network and broadcasts the $PRE-PREPARE$ message to all the replica nodes, and the number of communication times of this link is $(n-1)$, where n is the total number of nodes in the current consensus network; the main process of $PREPARE$ phase is that every replica nodes are required to broadcast $PREPARE$ messages to other nodes and verify the authentication status of the packaged transactions to the remaining nodes, if the verification passes, then the $PREPARE$ message subsidiary signature, the number of communications in this phase is $(n-1)^2$; the $COMMIT$ phase needs to verify that each node broadcasts a $COMMIT$ message for the other nodes with a signature and that the number of communications is (n^2-n).

From the above analysis, it can be seen that the main communication resource consumption in the consensus process of the PBFT algorithm exists in the two stages of $PREPARE$ and $COMMIT$. To avoid the phenomenon that the number of communications in the $PREPARE$ and $COMMIT$ phases increases with the increase of nodes by a very large number of orders of magnitude, the BLS aggregated signature scheme is designed. Based on the algorithm proposed by BONEH [29], this scheme is a signature scheme based on elliptic curves, which has the characteristics of high efficiency, high security, and good scalability. By redesigning the signature process, the signature structure of each node in the consensus process of the CA-PBFT algorithm is improved, and the communication complexity $O(n^2)$ is reduced to $O(n)$, which solves the problem that the consensus network is difficult to access large-scale nodes. The use of the BLS aggregate signature scheme makes the CA-PBFT algorithm more efficient, secure, and scalable. The scheme design is divided into six phases, namely, the aggregated signature initialization environment phase, the construction key phase, the public key aggregation phase, the node group setup phase, the signature phase, and the signature verification phase, which will be described in the following sequence.

1) AGGREGATE SIGNATURE INITIALIZATION ENVIRONMENT PHASE

This scheme is built on a non-degenerate bilinear mapping $e\colon \mathrm{G}_1 \times \mathrm{G}_2 \to \mathrm{G}_t$, where G_1, G_2, G_t are all groups of prime order q, and g_1 and g_2 are the generating elements of G_1 and G_2, respectively, and there are hash algorithms H_0 such as Equation (1) that allows the computed hash to be an element of the group G_1, and hash algorithms H_1 makes the computed hash value an element in the group Z_q.

$$H_0\colon \{0,1\}^* \to \mathrm{G}_1, \quad H_1\colon \{0,1\}^* \to \mathrm{Z}_q \tag{1}$$

2) BUILDING KEYS PHASE

After passing the initialization environment, the bilinear pair system parameter par is constructed according to Equation (2), and each node $RepNode_i$ generates a unique key pair according to Equation (3): private key $PrivateKey_i$, public key $PublicKey_i$, and the public key is disclosed to the consensus network, where i is used to calibrating the node's position in the consensus network.

$$par \leftarrow (q, \mathrm{G}_1, \mathrm{G}_2, \mathrm{G}_t, e, g_1, g_2) \tag{2}$$

$$PrivateKey_i \xleftarrow{random} (Z_q),\ PublicKey_i \leftarrow {g_2}^{ski} \tag{3}$$

3) PUBLIC KEY AGGREGATION PHASE

Aggregation node $AggNode$ by the current consensus network non-primary node $PriNode$ as well as nodes with node status $BLOCK$, when the aggregation node receives the unique public key transmitted by the nodes in the consensus

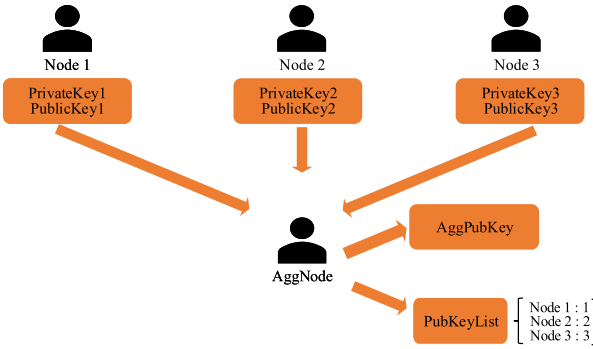

FIGURE 3. Public key aggregation process for each node.

network, it performs the aggregation operation on the public key, generates the aggregated public key *AggPubKey* by Equation (4), and sets the position of each node's serial number in the system, statistically known as *PublicKeyList*, as shown in Equation (5). The public key aggregation process for each node is shown in Figure 3.

$$AggPubKey \leftarrow \prod_{i=1}^{n} PublicKey_i^{H_1(PublicKey_i, PublicKeyList)} \tag{4}$$

$$PublicKeyList = \{PublicKey_1, PublicKey_2 \cdots, PublicKey_n\} \tag{5}$$

4) NODE CLUSTER SETUP PHASE

All nodes are provided with the *PublicKeyList*, a list of public keys published from the blockchain system, which contains information about the node's serial number position in the current system. For each node, the authentication signature parameter $u_{i,j}$ is generated for the user with serial number $j \in N$, as shown in Equation (6).

$$u_{i,j} \leftarrow H_2(AggPubKey, j)^{Prikey_i \cdot H_1(PubKey_i, PublicKeyList)} \tag{6}$$

Subsequently, each replica node $RepNode_i$ will make a certified disclosure of the serial number of each node and compute the signature parameter $u_{i,i}$ that is required for itself in a private environment as shown in Equation (7).

$$u_{i,i} \leftarrow H_2(AggPubKey, i)^{Prikey_i \cdot H_1(PubKey_i, PublicKeyList)} \tag{7}$$

Subsequently, each replica node collects the signatures of other nodes on its serial number to get the set $\widetilde{U}_i$ and adds its parameter $u_{i,j}$ to arrive at the set U_i as shown in Equation (8).

$$\begin{aligned} \widetilde{U}_i &= \{u_{i,1}, \cdots, u_{i,i-1}, u_{i,i}, u_{i,i+1}, \cdots, u_{i,n}\}, \\ U_i &= \{u_{i,1}, \cdots, u_{i,i}, \cdots, u_{i,n}\} \end{aligned} \tag{8}$$

Then a membership key *MebKey* can be calculated to represent the node's identity qualification within this group and can be signed and verified within the group, the formula is shown in Equation (9).

$$MebKey_i \leftarrow \prod_{j=1}^{n} u_{i,j} \tag{9}$$

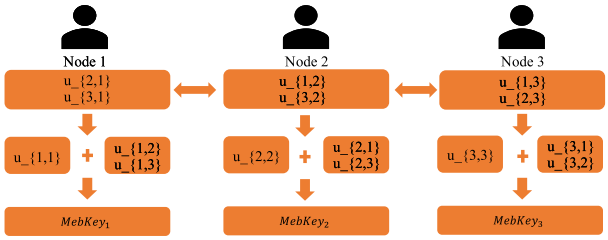

FIGURE 4. Flow of setting up groups for each node.

Up to this point, each replica node $RepNode_i$ has obtained the public-private key pair (*PrivateKey*, *PublicKey*) and the membership key $PublicKey_i$. All the nodes are verified by Equation (10), if the verification passes, all the nodes are honest, and if the verification fails, then it indicates that the node is evil. Where, the membership key $MebKey_i$ is the signature attached to H_2 (*AggPubKey*, *i*) by all nodes.

$$e(g_2, MebKey_i) = e\big(AggPubKey, H_2(AggPubKey, i)\big) \tag{10}$$

After the key-building phase, each node gets a pair of public and private keys and passes its public key $PublicKey_i$ to the aggregation node *AggNode* to generate the global aggregation public key *AggPubKey*, and at the same time sets the serial number of each node in it, and finally publishes the aggregation public key *AggPubKey* and the public key list *PublicKeyList*. At the same time, the nodes check each other's serial number in the system, sign to other nodes; and transfer back to the original node; when the node receives the reply from the other nodes, the group setup is performed, along with the self-signature to generate the membership key $MebKey_i$. At last, the nodes have generated the public and private keys as well as the group key, then it can be signed and verified freely. The specific flow of the aggregate signature group is shown in Figure 4.

5) SIGNATURE PHASE

The node $RepNode_i$ uses its private key $Privatekey_i$ and membership key $MebKey_i$ to sign the message m of the current round and forwards it to the current round of aggregation node *AggNode*, and at the same time performs a H_0 hash computation of the message *m* and the global aggregation public key *AggPubKey* as shown in Equation (11). The pseudo-code for signing message *m* is as in Algorithm 1.

$$sign_i \leftarrow H_0(AggPubKey, m)^{Privatekey_i} \cdot MebKey_i \tag{11}$$

After completing the signature, node $RepNode_i$ sends the public key $PublicKey_i$ and signature $sign_i$ to the aggregation node *AggNode* for aggregation operation. The aggregation node subgroup *S* is generated based on the signed nodes as shown in Equation (12).

$$S = S + 2^i \tag{12}$$

where the subgroup *S* statistics are performed in a bitwise manner, with the initial value set to 0, starting from the lower bit.

Algorithm 1 Signing Algorithms for Message

Input: Message to be signed m
Output: Signature $sign$

(PrivateKey,PublicKey,MebKey,AggPubKey) := Setup();
$h0$:=$Hash0(AggPubKey,m)$;
$h0$:=$pow(h0,PrivateKey)$;
$sign$:=$mul(h0, MebKey)$;

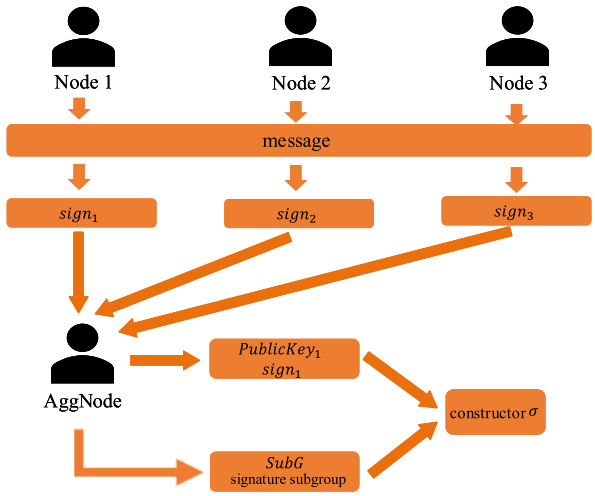

FIGURE 5. Specific process for signatures at each node.

Finally, the aggregation node can obtain the aggregated public key $AggPubKey$ and the aggregate signature $sign_1$ through the calculation of Equation (13), and at the same time, generate the structure σ:=($AggPubKey$,$sign_1$,S) to prepare for the subsequent signature validation session. The pseudo-code for generating the structure σ is as in Algorithm 2. The flow of the signature session is shown in Figure 5.

$$\begin{cases} PublicKey_1 \leftarrow \prod_{j \in \mathcal{S}} AggPublicKey_j \\ sign_1 \leftarrow \prod_{j \in \mathcal{S}} sign_j \end{cases} \tag{13}$$

6) SIGNATURE VERIFICATION PHASE

In this phase, node $RepNode_i$ will carry out the verification process, receive the structure σ from the aggregation node $AggNode$, and verify the signature $sign_a$ using the aggregate signature public key $AggPubKey$, which indicates that the verification passes and the signature is valid when and only when the Equation (14) holds. The flow of the node signature verification session is shown in Figure 6. The pseudo-code for the validation phase is as in Algorithm 3.

$$e(H_0(AggPubKey, m), PublicKey_1) \cdot e\left(\prod_{j \in \mathcal{S}} H_2(AggPubKey, j)\right) = e(sign_1, g_2) \tag{14}$$

Algorithm 2 Algorithm for Structure Generation

Input: The signature set signSet of message m, the signer public key $PublicKeySe$, the signer serial number set S
Output: Structure σ

Struct σ : {$PublicKey_a$,$sign_a$,$subgroup$};
if !chenckValid(σ) **then**
  return $NULL$;
end if
$\sigma \rightarrow subgroup$:=$S$;
$\sigma \rightarrow PublicKey_a \rightarrow Init()$;
$\sigma \rightarrow sign_a \rightarrow Init()$;
for PublicKey ∈ PubLicKeySet **do**
  $\sigma \rightarrow PublicKey_a$:=$mul(\sigma \rightarrow PublicKey_a, PublicKey)$;
end for
for $sign \in signSet$ **do**
  $\sigma \rightarrow sign_a$:=$mul(\sigma \rightarrow sign_a, sign)$;
end for
return σ;

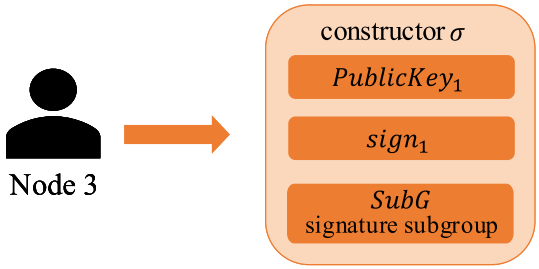

FIGURE 6. Signature verification process.

B. NODAL CREDIT SCORING MECHANISM

To manage the nodes participating in the consensus process in the current blockchain and reduce the probability of Byzantine nodes being elected as primary nodes in the consensus network, credit scoring rules are designed to divide the node state and limit their functions in the consensus process.

Credit scoring rules are defined as follows:

(1) Organization $Org_i \in \{Org_1, Org_2, \cdots, Org_n\}$, where n represents the number of organizations in the blockchain participating in the consensus process. Organizations are voluntarily applied for by individual users or other institutions and are certified and authorized by the consortium blockchain system. Each organization Org_i also contains n nodes $Node_{ij}$.

(2) The primary node $PriNode_{ij}$, where i indicates that it belongs to the organization Org_i, j represents the position of the node in the organization Org_i, and the other functions are consistent with the functions of the primary node in the PBFT algorithm.

(3) The replica node $RepNode_{ij}$, where i indicates that it belongs to the organization Org_i and j indicates the location of its node in the organization Org_i, P2P communication is used between nodes, and the nodes in the organization store

the organization's private key *PrivateKey*. For the judgment of the node's behavior, if the node maintains the normal operation of the consensus process of the blockchain network and produces blocks normally, it is an honest node; malicious tampering with block data, blocking the consensus process, and destroying the consensus protocol are bad nodes or faulty nodes.

(4) The credit value C_{ij}, where i represents the organization to which it belongs Org_i, where j indicates the location of its node in the organization Org_i. For the setting of credit value, this study sets a total of five levels, C_{max} is the maximum available credit value of the node, C_{trust} indicates that the credit value range of the node has entered the trusted stage, C_{init} indicates the initial value of the credit value of the node in the process of joining the consensus for the first time, $C_{untrust}$ indicates that the current credit value of the node is in the untrustworthy stage, and C_{min} indicates the minimum credit value of the node, and intends to enter the elimination process. The credit value range is shown in Equation (15).

$$C_{\min} < C_{untrust} < C_{init} < C_{trust} < C_{max} \tag{15}$$

Algorithm 3 Algorithms for Signature Verification

Input: Signed message m, structure σ
Output: Validation success or failure

```
if !chenckValid(σ) then
    return NULL;
end if
h0:=Hash0(AggPubKey, m);
e1:=pairing(h0,σ→PublicKey_a);
h2Prod→Init();
for j∈σ→subgroup do
    h2:=Hash02(AggPubKey,j);
    h2Prod:=mul(h2Prod,h2);
end for
e2:=pairing(h2Prod,AggPubKey);
e3:=pairing(σ → sign_a,g2);
if mul(e1,e2)==e3 then
    return TRUE;
else
    return FALSE;
end if
```

(5) Credit reward and punishment rules: To enhance the enthusiasm of the participating nodes in the consensus process to improve the consensus efficiency and avoid the evil nodes from affecting the consensus, rewards, and punishments are made according to the contributions of the nodes to the consensus process. The node reward equation is shown in Equation(16), and the node penalty equation is shown in Equation(17).

$$C_{ij} = C_{ij} + reward \tag{16}$$
$$C_{ij} = C_{ij} - punish \tag{17}$$

where C_{ij} is denoted as the credit value of node j in the organization org_i; *reward* is the reward score made by the node in the link; *punish* is the penalty score that the node received in that session.

(6) Credit recovery mechanism: To prevent the node's credit value from always being at a high level, even if it has done evil many times in a row, it still cannot reduce the possibility of it serving as the primary node, resulting in the impact of the consensus process in multiple rounds, the credit regression coefficient is set to avoid the node being punished by too high credit value and still unable to avoid it from joining the consensus after multiple rounds. The credit recovery equation is shown in Equation (18) and Equation(19).

$$C_{ij} = C_{ij} - [t/T] \times R \quad (C_{ij} > C_{init}) \tag{18}$$
$$C_{ij} = C_{ij} + [t/T] \times R \quad (C_{ij} < C_{init}) \tag{19}$$

where t represents the time from the last time the node participated in the voting process to this participation in the voting; T represents the average time constant for the node to participate in a consensus; R stands for credit recovery constant.

(7) Node credit status: $Status_{ij}$ where i is the location of the node in the organization Org_i, and j is the location of the node in the organization. According to different credit scores, four statuses are designed, and the categories are shown in Equation (20).

$$\sigma(Node_{ij}) = \{Excellent, Good, Fair, Poor\} \tag{20}$$

In the formula, *Excellent* means that the node has not participated in the generation of invalid blocks within a period T, and has not affected the consensus efficiency, and its credit value is in the range of $C_{trust} < C_{ij} < C_{max}$. *Good* means that the node has no behavior for some time has no relationship with the generation of blocks, and its credit value is in the range of $C_{init} < C_{ij} < C_{trust}$; *Fair* means that the node participates in the generation of blocks at time T and produces invalid blocks, affecting consensus, and its credit value is in the range of $C_{untrust} < C_{ij} < C_{init}$; *Poor* indicates that the node has participated in block generation within time T and has continuously produced invalid blocks, and is ready to undergo the elimination procedure, and its credit value is in the range of $C_{min} < C_{ij} < C_{trust}$. The node credit state transition is shown in Figure 7.

Among them, there are four node states, and the node with the *Excellent* state has all the functions of the node; The node with the *Good* status is the node that has joined the consensus network for the first time or has carried out credit-related mechanisms, and only restricts the priority of its primary node; Nodes in the *Fair* state are nodes that have no relevant contributions to the current consensus process, and cannot be elected as the main node in the consensus process; The node in the *Poor* state is the node that intends to enter the elimination process, and the primary node is not allowed to be elected and the possibility of serving as a replica node is

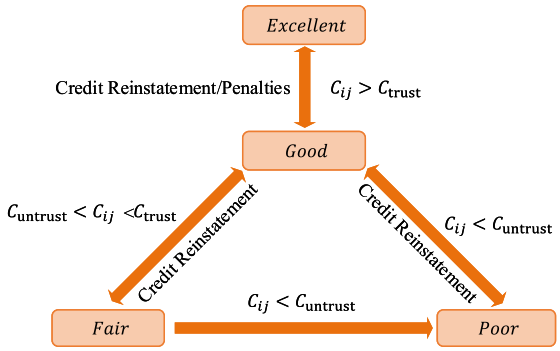

FIGURE 7. Node credit state transition.

reduced. Nodes in the *Block* state are deprived of all functions and enter the elimination procedure, as shown in Table 3.

In the process of the traditional PBFT consensus algorithm, the probability of each node becoming the primary node is the same, and they all have the right to initiate consensus and obtain consensus results. If the node to be selected as the primary node is the evil node, the communication resources of this round will be wasted, failing the consensus process of the round and the failure of the consensus result. At the same time, in the face of large-scale node access, the traditional *PBFT* consensus algorithm will carry out P2P communication in the *PREPARE* stage and the *COMMIT* stage, which requires a lot of communication resources, resulting in a sharp decline in the performance of the consensus network. To improve the participation of honest nodes in the consensus network and avoid the direct selection of evil nodes to affect the consensus process, a voting mechanism based on node credit score is designed, which can increase the probability of nodes with higher credit scores becoming primary nodes. The node credit scoring mechanism is defined as follows:

(1) Voting

According to Figure 7 and Table 3, only the nodes with the node status of *Excellent*, *Good*, *Fair*, and *Poor* have the right to vote, and the nodes participating in the consensus are selected among the above nodes. There are two options for voting, for and against. In each voting session, each node can vote for or against within the time parameter T.

(2) Voting nodes

Voting nodes can be obtained from Table 3, except for nodes with a credit status of Block, they can participate in the voting process and can increase their credit score through voting.

(3) Voting results

The calculation of voting result will be calculated by the credit status of the nodes participating in the voting and their credit scores, as shown in Equation (21).

$$Result_{ij} = State_{ij} \times C_{ij} + \sum_{k=1}^{N} \sum_{l=1}^{n} State_{kl} \times Vote_{kl} \quad (21)$$

where $State_{ij}$ is the credit state coefficient of the corresponding node; C_{ij} is the credit value of the node; N represents the number of organizations participating in voting; n represents the number of nodes in the organization that participate in voting; $Vote_{kl}$ is the vote value of the corresponding node, which is set in this article, and the value is taken as 1 for support and -1 for opposition.

(4) Rules for voting rewards and punishments

To prevent the malicious voting of the nodes participating in the voting from affecting the consensus results, and to restore the credit value of the nodes participating in the voting, the following voting reward and punishment rules are defined:

First, when a node with a node status of Poor participates in the election of a primary node or a replica node and is not successfully elected, all participating nodes that vote against it in the consensus process will receive credit recovery rewards.

Second, when the elected node successfully generates a block, all participating voting nodes that vote for it will receive credit recovery rewards.

Third, when the elected node does not produce a block and becomes a bad node or a faulty node, the participating voting nodes that oppose its support will receive credit penalties; Instead, get credit rewards.

In the voting process, nodes with voting functions will participate in this vote, and the voting result calculation Equation (21) will be used to count the voting results of this round.

In addition, improvements are made to the PBFT consensus algorithm by adding a coordination node to its original node type and establishing two tables in the coordination node (we assume that the coordination nodes are trusted and reliable secure nodes), respectively, the information table of all nodes and the information table of consensus nodes, and the newly added coordination node is responsible for coordinating the detailed information of all nodes in the entire consensus network as well as coordinating the dynamic joining and exiting of nodes in the consensus network.

The coordination node, shown in Figure 8, does not participate in the entire consensus process, but merely records the status of the nodes in the consensus process at the time of execution and receives request information from newly joined nodes and nodes to be withdrawn. When there is a proposed joining node, the node will send a request to the current network node status table and synchronize the status; when the node is proposed to exit, the coordination node will broadcast the current network node status change information to other nodes in the network.

The whole node information table and the consensus node information table are shown in Table 4 and Table 5. The members in the whole node information table include the nodes currently participating in the consensus as well as the nodes that will be eliminated in the future, and their information mainly contains the node's organizational location, public key, credit value, credit status, etc.; whereas the consensus node information table only contains the information of the nodes that are participating in the current

TABLE 3. Node state permissions.

Permissions	Excellent	Good	Fair	Poor	Block
Primary node priority	*Y*	*N*	*N*	*N*	*N*
Whether it can be used as a primary node	*Y*	*Y*	*N*	*N*	*N*
Replica node priority	*Y*	*Y*	*Y*	*N*	*N*
Whether it can be used as a replica node	*Y*	*Y*	*Y*	*Y*	*N*
Ability to vote	*Y*	*Y*	*Y*	*Y*	*N*
Consensus results can be obtained	*Y*	*Y*	*Y*	*Y*	*N*

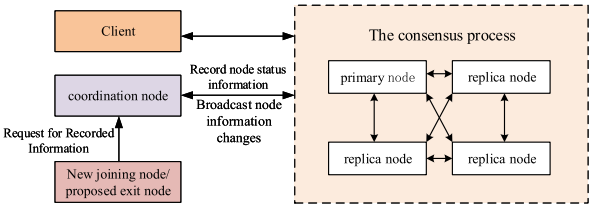

FIGURE 8. Design of coordination node.

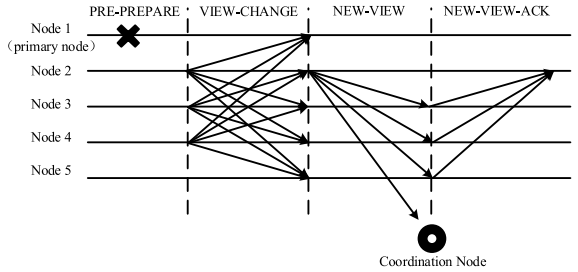

FIGURE 9. View–specific switching process.

TABLE 4. All node's information table.

ID	Public Key	Credit Status	Credit Score
1	$PublicKey_1$	*Good*	55
2	$PublicKey_2$	*Poor*	12
3	$PublicKey_3$	*Fair*	40
4	$PublicKey_4$	*Excellent*	98
5	$PublicKey_5$	*Good*	71
6	$PublicKey_6$	*Good*	70

TABLE 5. Consensus node information table.

ID	Serial Number	Public Key	Credit Status	Credit Score
1	1	$PublicKey_1$	*Good*	55
2	2	$PublicKey_2$	*Poor*	12
3	3	$PublicKey_3$	*Fair*	40
4	4	$PublicKey_4$	*Excellent*	98

consensus session, and the information is the same as that in the whole node information table, except that the node's location in the process of the consensus network has been redefined.

1) VIEW SWITCHING PROTOCOL

To ensure the consensus efficiency of the algorithm, when the replica node *RepNode* suspects that the current primary node *PriNode* is an evil node, or the credit score of the primary node is too large leading to timeout and evil. The replica node then enables the view-switching protocol to change the selection of the primary node. The view-switching protocol mainly includes the $VIEW-CHANGE$ phase, $NEW-VIEW$ phase, and $NEW-VIEW-ACK$ phase, and the view-specific switching process is shown in Figure 9.

$VIEW-CHANGE$ phase: in the current round view v, if the replica node does not receive a response to the relevant message from the primary node within the specified time T, it is determined that the primary node is faulty and a request for applying for a view-change is sent to prepare for the change of the primary node. Entering into the view with round $v+1$, it simultaneously penalizes the primary node with credit and updates the credit value score of that node, and at the same time encapsulates the $VIEW-CHANGE$ message $\ll view-change, v, h_{last}, credit_I >_{signature}, I, c >$ broadcasting to all the nodes of this consensus process, where h_{last} is the height of the block confirmed in the most recent round of this node, $credit_i$ is the node's credit score for the primary node *PriNode* after it was credit penalized, *signature* is the signature of this node's $RepNode_i$ for its encapsulated message, i is the number of the node that encapsulated this message in the consensus node's information table, and c is $PRE-PREPARE$ message and *COMMIT* message for the block generated by the height of the h_{last} node.

$NEW-VIEW$ phase: when the new primary node *PriNode* receives $2f$ identical $VIEW-CHANG$E messages from different nodes, it verifies the consistency of its view v as well as block height h. If they are consistent, the $NEW-VIEW$ message is encapsulated $\ll new-view, v, h_{next}, credit_s, node >_{signature}\gg$, and the new primary node then enters into the round $v+1$ view and broadcasts the $NEW-VIEW$ message to the other nodes, h_{next} is the height of the next generated block, and *node* is the list of all node IDs participating in this aggregated signature.

$NEW-VIEW-ACK$ phase: the primary node *PriNod*e that produces the next block will send a $NEW-VIEW$ message to each replica node and coordinator node in the consensus network, and then the replica nodes and coordinator nodes will verify the signature information in the message. If the verification passes, the primary node's credit value score in the whole node information table and the consensus node information table will be updated, and the replica node RepNode encapsulates and sends a $NEW-VIEW-ACK$ message to the primary node of the next round $\ll new-view-ack, v, h_{next}, credits > signature, i >$, where i is the number of the replica node that sent the $NEW-VIEW-ACK$ message in the consensus node

information table, and sending the $NEW - VIEW - ACK$ message indicates that the validation has passed and opens a new round of consensus.

2) NODE DYNAMIC JOINING PROCESS

Preparation phase: since the consensus method in this study is deployed in the consortium chain (permission chain) when a new node applies to join the consortium chain, it first needs to apply to join the node to be authorized to obtain its private key, public key, and other information.

Application phase: when a new node applies to join, the coordinating node will update the node's node public key, credit status, and credit score of that node in the information table of all nodes, and at the same time synchronize the information table of all nodes and the information table of consensus nodes to the new node. The coordinating node encapsulates and broadcasts a *JOIN* message to all nodes participating in this consensus network in the format of $< join, i, id, ip, PublicKey, timestamp >_{signature}$, where i is the number assigned to the new joining node in the consensus node information table, id is the ID of the new joining node in the whole node information table, ip is the ip address of the new joining node, $PublicKey$ is the BLS public key of the new joining node, $timestamp$ is the joining time of the node, and $signature$ is the signature of the coordinating node on the $JOIN$ message, which is used to verify the authenticity of the joining message, and its BLS aggregate signature method is described in the previous subsection.

Validation phase: when all the nodes in the consensus network receive the $JOIN$ message sent from the coordinating node, they need to validate the message. If the validation passes, then encapsulate and send a $JOIN - REPLY$ message in the format of $\ll join - reply, i, id >_{signature}, j >$ as well as feed the node number j to the coordinating node to indicate the agreement of the node to join the consensus process, where j is the number in this consensus network of the replica node $RepNode_j$ that sent the $JOIN - REPLY$ message.

Acknowledgement phase: when the coordinating node receives $JOIN - REPLY$ messages from nodes accumulating to $2f + 1$, it constructs a $JOIN - COMMIT$ message in the format of $< join - commit, i, id, node, v >$, where $node$ refers to the list of information about the nodes that are currently participating in the aggregation signature process, and v is the next view serial number. Finally, the coordinating node will encapsulate the $JOIN - COMMIT$ message and broadcast it to all replica nodes in this consensus process. When all the nodes participating in the consensus process receive the $JOIN - COMMIT$ message, they will verify it, and when the information is verified, the coordinating node will add the node's information to the whole node information table and the consensus node information table.

The node dynamic joining process is shown in Figure 10.

3) NODE DYNAMIC EXITING PROCESS

Application phase: the application withdrawal node encapsulates the $EXIT$ message $< exit, id, ip, timestamp >_{signature}$ and sends it to the coordinating node, where $exit$ is the node's application exit message, id is the position of the application exit node's serial number in the consensus node table, $timestamp$ is the node's application exit time, and $signature$ is the node's signature on the $EXIT$ message.

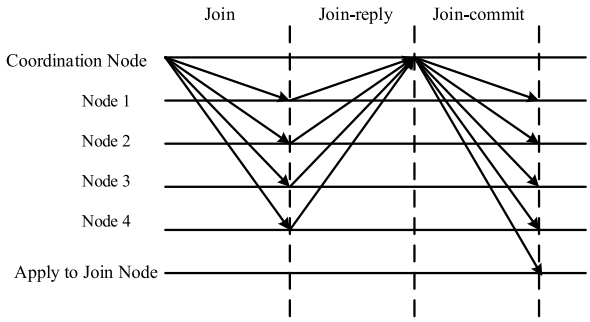

FIGURE 10. Node dynamic joining process.

Verification phase: when the coordinating node receives the $EXIT$ message, it verifies the signature $signature$ in the message encapsulated in the $EXIT$ message, and if the verification passes, it encapsulates and sends the $EXIT - REQ$ message $< exit - req, id, exitmsg >_{signature}$ to the rest of the nodes in the consensus network, where id is the serial number position of the requesting exit node in the consensus node table, and the $exitmsg$ message contains the request information of the proposed exit node. When the other node $Node$ in the consensus network receives the $EXIT - REQ$ message broadcast from the coordinating node, it verifies the $signature$ and $exitmsg$ in the message. If the validation is successful, the encapsulated $EXIT - REPLY$ message $\ll exit - reply, id >_{signature}, j >$ is sent to the coordinating node to indicate its consent for the node to exit the consensus network, where j is the number of the node that agrees to the node's exit.

Exit phase: the coordinating node starts to collect the $EXIT - REPLY$ messages broadcasted by the nodes in the consensus network, and when $2f + 1$ messages have been collected, the coordinating node starts to construct an $EXIT - COMMIT$ message in the format $< exit - commit, id, node >_{signature}$, Where $node$ is a list of ids of all nodes participating in this aggregated signature for subsequent signature verification. At the same time, it updates the whole node information table, and consensus node information table, changes the node status to $Block$, deletes the relevant information of the node, including ID, etc., and finally broadcasts the $EXIT - COMMIT$ message to the consensus network.

The node dynamic exit-specific process is shown in Figure 11.

4) ELIMINATION OF LOW CREDIT VALUE NODE PROCESS

In each consensus process, if there is a faulty node or an evil node, it will lead to the failure of this consensus, affecting the consensus efficiency and wasting the communication resources. At the beginning of each consensus, the entire

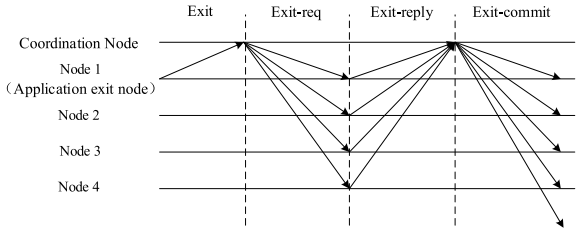

FIGURE 11. Node dynamic exiting process.

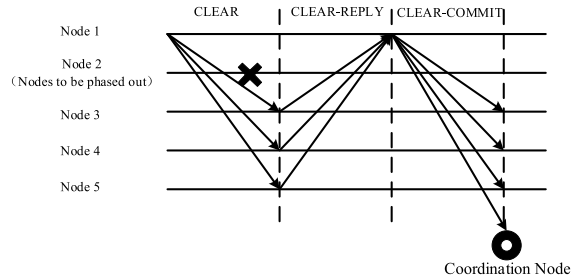

FIGURE 12. Evil node elimination process.

consensus network is checked for the existence of nodes with node credit status of *Block*, and their elimination process is performed. The specific process of eliminating evil nodes is shown in Figure 12.

Preparation phase: before starting consensus, *PriNode*, the primary node in the current consensus network, will check the credit status in the information table of all nodes in its network, and if it finds a node whose node's credit status is *Block*, it will start the process of eliminating the nodes with low credit values, encapsulate the *CLEAR* message $< clear, v, id, ip >_{signature}$ and broadcast it to the nodes whose node credit status is not *Block*, where v is the current view round number, id is the *ID* of the node whose credit status is *Block* in the whole node information table, and *signature* is the signature of the primary node for the *CLEAR* message.

Verification phase: when a node in the consensus network, except the node whose credit status is *Block*, receives a *CLEAR* message from the primary node, it verifies whether the information is correct or not, and at the same time verifies the information table of all nodes of the participating nodes whether the node's credit status is *Block* or not, and if the verification is successful, then it constructs a $CLEAR - REPLY$ message $\ll clear - reply, v, id, ip >_{signature}, j >$ and broadcasts it to the primary node, where j is the number of the signature of that node in the consensus node's information table.

Elimination phase: in the case of $CLEAR - REPLY$ messages from $2f + 1$ different nodes, verify whether the message is authentic or not. If the verification passes, the verified signatures are aggregated through *BLS* aggregation signature and the encapsulated message $CLEAR - COMMIT$ message $< clear - commit, v, id, ip, node >_{signature}$ is broadcast to the nodes in the consensus network whose node status is not *Block* and to the coordination nodes. When the node that receives this message passes on its verification, it updates the consensus node information table and completes the eviction status.

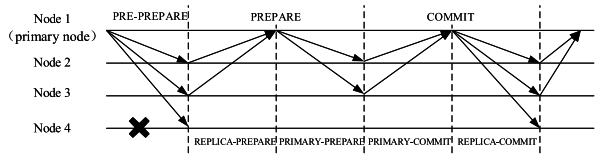

FIGURE 13. CA-PBFT algorithm consensus process.

C. ALGORITHMIC CONSENSUS PROCESS

The CA-PBFT algorithm maintains the consensus process of the traditional PBFT algorithm and requires an initialization environment configuration before proceeding to the consensus session, first of all, it is necessary to initialize the cryptographic environment of the aggregated signature scheme par, the parameters of each node's public-private key and aggregated public key $(PrivateKey_i, PublicKey_i, MebKey_i)$. The CA-PBFT algorithm consensus process is shown in Figure 13.

$PRE - PREPARE$ phase: the primary node *PriNode* will pack the set of transactions $T = \{T_1, T_2, \cdots T_{max}\}$ that exists in the transaction pool into the block *Block*, and at the same time broadcast the $PRE - PREPARE$ message $\ll pre - prepare, v, n, H, d, credit_{PN} >_{S_{PN}}, B >$, where $credit_{PN}$ is the node authentication information, which contains the block *Block*'s $PRE - PREPARE$ message and marks the serial number as n, the height of the block *Block* is H, and d is the block Block digest, namely the block hash value.

PREPARE phase: the backup node $RepNode_i$ enters the *PREPARE* state and sends a *PREPARE* message to the primary node *PriNode*. At the same time, $RepNode_i$ executes the sign function and attaches the signature s_i as well as the node information state $credit_i$, which encapsulates and sends the *PREPARE* message $< prepare, v, n, H, credit_i, i >_{S_i}$ to a primary node *PriNode*. The primary node *PriNode* performs signature verification, node information state table verification of *PREPARE* messages from different replica nodes, and counts the set S of that replica node. When the number of verified *PREPARE* messages reaches $2f + 1$, the collected signatures and corresponding public keys are then aggregated to generate the structure $\sigma_{prepare}$ in that round, and at the same time construct the *PREPARE* message $< prepare, v, n, \sigma_{prepare}, credit_{PN} >_{PN}$. And broadcast the *PREPARE* message to other backup nodes based on the node information in $credit_{PN}$.

COMMIT phase: backup node i enters the *COMMIT* phase and replies to the primary node *PN* to confirm the *COMMIT* message, which is in the format $< commit, v, n, H, credit_i, i >_{S_i}$. When the primary node receives a *COMMIT* message with the same $2f + 1$ *Hash* values, view number v, and node information $credit_{PN}$, it aggregates the collected signatures

and the corresponding public keys and generates a signature structure σ_{commit} for the round, which is encapsulated and stored in the *COMMIT* message, with the message format $< \quad commit, v, n, H, \sigma_{commit}, credit_{PN}, PN \quad >_{S_{PN}}$. Finally, an acknowledgment message is broadcast to the backup node based on the local node information table.

Reply phase: after the backup node receives the *COMMIT* message from the primary node, the backup node adopts the same verification rules as in the *PREPARE* phase to test whether the *COMMIT* message is legitimate or not. Passing the test means that the block *Block* is recognized by the consensus network, and the backup node starts to execute the transaction operation in the block *Block*. When the transaction is finished, the result is fed back to the client, when the client receives $f + 1$ the same results from different nodes, it means the request is completed, otherwise, it means failure.

IV. SECURITY AND PERFORMANCE ANALYSIS

A. SECURITY ANALYSIS OF AGGREGATE SIGNATURE SCHEMES

1) SECURITY MODEL

The digital signature designed in this scheme defines security as the aggregate signature scheme in a game that is said to be secure if the saboteur holds the maximum attack resources but is unable to reach the sabotage. That is, the saboteur cannot make an existential forged message attack. The verification process of the aggregate signature scheme is as follows:

Set up saboteur A and defender C. There exist n signature members $\{U_1, U_2, U_3, \cdots, U_n\}$, of which there exist l signature members that are manipulated by the saboteur A and that can generate l signature member signatures, where $1 \leq l \leq n$.

Step 1: defender C runs the initialization algorithm par, generates the relevant parameters $params$, and sends the published parameters to saboteur A.

Step 2: saboteur A initiates a *hash* value query, and defender C may manipulate the hash random predictor to respond to saboteur A request, or saboteur A creates a message digest m_i as input, and defender C manipulates the signature random predictor to respond to a portion of the signature content in the digest m_i.

Step 3: when the saboteur has initiated multiple queries, the aggregate signature structure σ^* is output for the message m^*, where m^* is not part of any of the inputs in the non-partial signatures in item 2. If σ^* passes the verification of the message m^* sent by the signature members $\{U_1, U_2, U_3, \cdots, U_n\}$, it is determined that the saboteur A attack is successful.

As a result, under a stochastic prediction model, there exist n signature members $U_1, U_2, U_3, \cdots, U_n$, and saboteur A gains control of l signers and can select messages for imitation under unrestricted conditions, and in polynomial time t, launch up to q_H *Hash* queries and up to q_s partial signature queries to the defender. Assuming that the saboteur A achieves a probability of success of ε under $((n, l), q_H, q_s, \varepsilon)$, the scheme is not sufficiently secure, otherwise the scheme complies with the security setting.

2) SECURITY VERIFICATION

Assume that the signers $\{U_1, U_2, U_3, \cdots, U_n\}$ have and have only one honest signer, U_1, and that the disruptor has the maximum attack capability, and that the disruptor can gain control of $n - 1$ signers. Let the generating element of group G_1 be P. The computational $Diffe - Hellman$ problem on group G_1 is defined as $(aP, bP) \in G_1$, and defender C uses the saboteur A as a subroutine to output the value of abP.

Step 1: defender C runs the initialization algorithm to generate the additive cyclic group G_1 and the multiplicative cyclic group G_2, as well as the bilinear mapping $e : G_1 \times G_1 \rightarrow G_2$, which generates the system parameters $par = \{G_1, G_2, e, q, P, H_1, H_2, H_3\}$.

Step 2: defender C uses list L_1 to record H_1 values. Saboteur A inputs signature member L and user public key $PublicKey_i (1 \leq i \leq n)$, and defender C queries list L_1. If there exists a record in L_1 that is the same as $(L, PublicKey_i, a_i)$, it responds to a_i to saboteur A. Alternatively, challenger C arbitrarily chooses $a_i \in Z_q^*$ and saves $(L, PublicKey_i, a_i)$ to L_1 and responds a_i to saboteur A.

Step 3: saboteur A sends message m_i, defender C checks if (m_i, L, P, W_i) information exists in list L_2, if it exists, then sends W_i to saboteur A. Otherwise, defender C randomly chooses $c_i \in Z_q^*$ and calculates $W_i = c_i(bP) \in G_1$, and records (m_i, L, P, W_i) in list L_2, and returns W_i to defender C.

Step 4: in the case of determining the set of signature members L, saboteur A uses the public key $PublicKey_i$, the signature order D_i, and the defender C inquires whether there exists the corresponding record $(D_i, PublicKey_i, V_i)$ in the list L_3, and if it exists returns to the saboteur A. If it doesn't exist, then the defender C randomly chooses the $V_i \in Z_q^*$ and records $(D_i, PublicKey_i, V_i)$ in the list L_3, and returns the V_i to the saboteur A.

Step 5: saboteur A inputs user U_i, and if $U_i \neq U_1$, then defender C randomly selects $PrivateKey_i \leftarrow Z_q^*$, calculates the public key $PublicKey_i = PrivateKey_iP$, obtains $(U_i, PrivateKey_i, PublicKey_i)$, stores it in the list S, and returns $PrivateKey_i$ to saboteur A. If $U_i = U_1$, then defender C orders $PublicKey_1 = aP$, resulting in $(U_1, PrivateKey_1)$, which is stored in list S.

Step 6: saboteur A records user U_i's partial signature result in list P. Saboteur A inputs user U_i and his public key $PublicKey_i$, and verifies whether $\left(U_i, PublicKey_i\right)$ is a valid signature for message m_j. When $U_i \neq U_1$, defender C queries table L_1 for $(L, PublicKey_i, a_i)$, L_2 for (m_i, L, P, W_i), L_3 for $(D_i, PublicKey_i, V_i)$ and also S for the private key $PrivateKey_i$, and computes U_i's signature on the part of m_j, $Sign_i = (V_i + a_iPrivateKey_i)W_i$, and returns $Sign_i$ to the saboteur A. When $U_i = U_1$, exit the partial signature query and return failure.

Step 7: for the signature set $\{L = U_1, U_2, U_3, \cdots, U_n\}$ and the message m^*, the disruptor A outputs the forgery result σ^*.

TABLE 6. Experimental hardware environment configuration.

Hardware Environment	Configuration Model/Version
CPU	$i7-11800H$
RAM	$32G$
Network Bandwidth	$1000MB$
Hard Disk	$1T$

If the above process is not terminated, the private key of user U_1 is not successfully submitted to the private key query list, and σ^* has not been partially signed query, then σ^* forged successfully. Saboteur A succeeds, then Equation (22) holds.

$$(\sigma^*, P) = (\mathfrak{c}^*(bP, V_1P + a_1(aP) + \sum_{i=2}^{n}(V_iP + a_iPublicKey_i)) \tag{22}$$

The defender C can calculate the partial signature result σ_1 of U_1 by looking up the list $\{L_1, L_2, L_3, S, P\}$ and the valid signature result σ^* of the saboteur A. The result σ_1 can be obtained from the Equation (23).

$$\sigma_1 = \sigma^* - \sum_{i=2}^{n}(V_i + a_i\text{PrivateKey}_i)[c^*(b\text{P})] \tag{23}$$

It also follows from the partial signature Equation (24).

$$\sigma_1 = (V_1 + a_1(a\text{P}))c^*(b\text{P}) \tag{24}$$

So defender C can calculate abP from Equation (25).

$$ab\text{P} = (\sigma_1 - c^*(b\text{P}))(a_1c^*)^{-1} \tag{25}$$

Therefore, if saboteur A can defeat the design with a great advantage ε in polynomial time, then defender C can also solve the CDH problem with the resources of saboteur A in the same amount of time [30].

B. PERFORMANCE ANALYTICS

The system building and testing environment is mainly realized through *Linux* virtual machine services. The specific hardware and software version parameters and configuration information are shown in Table 6 and Table 7. The experiments implement PBFT, CPBFT [19], G-PBFT [21], VS-PBFT [18], RB-BFT [28] and CA-PBFT algorithms, and the experiments test the performance of the CA-PBFT algorithm through the comparative analysis of the communication complexity, the signature construction efficiency, and the signature verification efficiency, as well as the comparison of the consensus delay, node joining consensus delay, node exiting consensus delay, and the reliability. The comparison of the above algorithms is shown in Table 8.

1) COMMUNICATION COMPLEXITY ANALYSIS

The traditional PBFT consensus algorithm process of three phases always needs $(2n^2 - n - 1)$ times of communication, from Figure 2, the communication complexity is $O(n^2)$, where n is the number of nodes, in the combination of PBFT improved algorithm after the aggregation of signatures, the algorithm process is shown in Figure 13, the number of communication in the three phases is only $(5n - 7)$, and the communication complexity is only $O(n)$.

TABLE 7. Experimental software environment configuration.

Software Environment	Configuration Model/Version
Node.js	$V6.5.11$
Bootstrap	$V4.0$
Hyperleger Fabric	$V1.0$
Language Environment	$Go1.11.4$

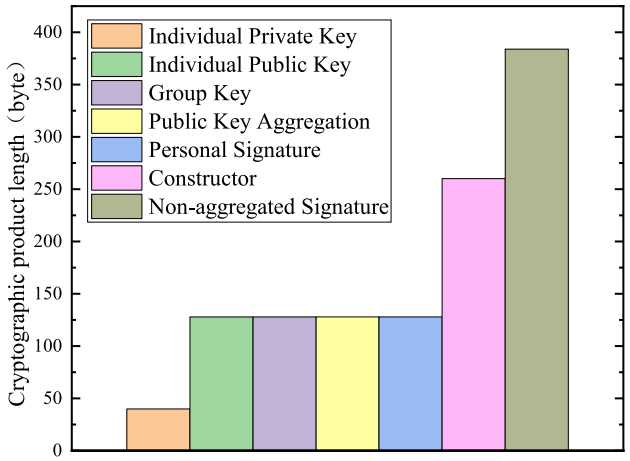

FIGURE 14. Cryptographic product length.

2) ANALYSIS OF THE ACTUAL LENGTH OF CRYPTOGRAPHIC ARTIFACTS AT VARIOUS STAGES

A comparative analysis of the relevant data generated at each stage of the designed aggregate signature scheme, such as the personal public and private key (*Privatekey*, *PublicKey*), the membership key *MebKey*, the aggregated public key *AggPubKey*, the personal signature *sign*, the structure σ and the length of the signature generated without the use of this scheme.

The experiment sets the member users to be 3, and makes 3 of them sign a message with aggregation, and S is set to 32 bits. The result is shown in Figure 14, in which each element conforms to the curve definition, and the structure body contains the aggregated public key, aggregate signature, and subgroups. The length of the structure is significantly smaller than the length of the non-aggregated signature, which is only 67.77% of the length of the non-aggregated signature.

3) SIGNATURE CONSTRUCTION AND VERIFICATION EFFICIENCY ANALYSIS

A signature construction comparison test is conducted for the *Schnorr* signature algorithm [31] and the signatures used in this paper to obtain the signature construction time by setting the time point to obtain the actual construction time of the signature. This experiment tests the signature verification time by setting different numbers of nodes to analyze the

TABLE 8. Comparison with other algorithms.

Algorithm	Improvement Point	Fault-Tolerant Limit	Communications Complexity	Dynamic
PBFT	/	1/3	$O(n^2)$	NO
CPBFT[19]	Creditworthiness-Based PBFT Consensus Algorithm	1/3	$O(n^2)$	NO
G-PBFT[21]	Scalable Location-Based Consensus Protocol	1/3	$O(n^2)$	NO
VS-PBFT[18]	Improved Consensus Algorithm Based on Vague Sets	1/2	$O(n^2)$	YES
RB-BFT[28]	Reputation-Based Byzantine Fault-Tolerant Algorithm	1/2	$O(n)$	NO
CA-PBFT[proposed]	PBFT Consensus Mechanism based on Credit Scoring and Aggregated Signatures	1/2	$O(n)$	YES

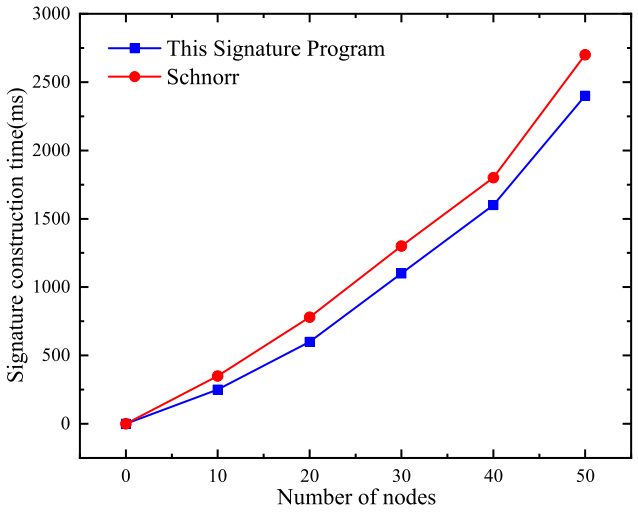

FIGURE 15. Signature construction test.

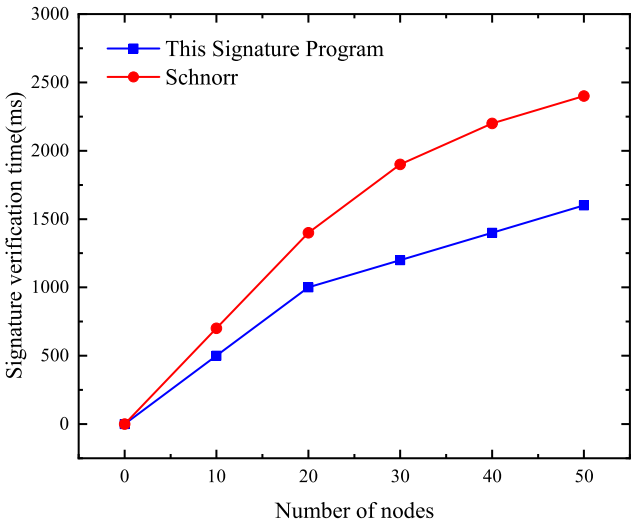

FIGURE 16. Verify signature efficiency test.

impact of the number of nodes on the efficiency of the signature construction of this scheme.

Figure 15 shows the construction time comparison between *Schnorr* and the signature scheme used in this paper for different numbers of nodes. From the figure, it can be seen that the signature construction time increase of this scheme is not as large as the increase of the *Schnorr* scheme in the face of the increase in the number of nodes, and the overall improvement in efficiency is about 16.6%.

For signatures, the importance of signature verification efficiency is at the highest level, because in the consensus process, the construction of the signature needs only once, while the signature verification needs many times. For example, if there is a Byzantine fault node in the consensus process, the node needs to be put forward for the consensus process and the signature verification phase needs to be carried out again, so there is a signature construction and signature aggregation and two signature verifications in the consensus process, so the signature verification efficiency is a key factor that affects the performance of the consensus algorithm. In the *Schnorr* signature, the signer needs to calculate all the hash values of all the signature messages before generating the final signature value, while the signature scheme used in this paper only needs to aggregate the public keys of all the signature messages, and the aggregated public key and the hash value of the message can be calculated once to generate the final signature value, which greatly reduces the amount of computation compared to the *Schnorr* signature approach. When the number of signers is more than 50 or so, the design scheme in this paper can achieve 1.49 times faster signature construction speed than the *Schnorr* signature scheme through the aggregation of signatures, which has the best performance compared to the Schnorr signature, as shown in Figure 16.

4) NODE DYNAMIC JOIN AND EXIT TEST ANALYSIS

In this experiment, by initially setting up a consensus network of 4 nodes, the time trend of adding a new node into the consensus network to participate in communication is tested by adding nodes one by one. Simultaneous simulations of the VS-PBFT algorithm and CA-PBFT algorithm are performed and analyzed in comparison.

As can be seen from Figure 17, the difference between the time taken to join nodes in the CA-PBFT and the VS-PBFT is small in the initial stage, but with the increase in the number of nodes in the consensus network, the time taken by the newly joined nodes gradually increases and the nodes gradually widen the gap during the process of node increase, and the time taken decreases by an average of 24.47%, which can be obtained from the fact that this algorithm makes

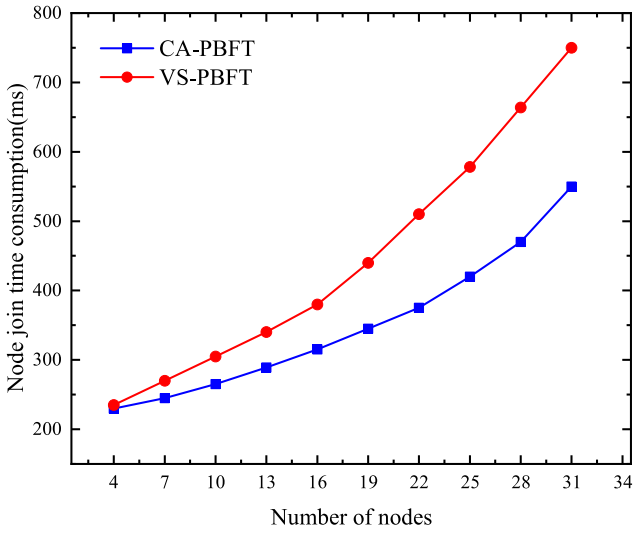

FIGURE 17. Node dynamic join consensus test analysis.

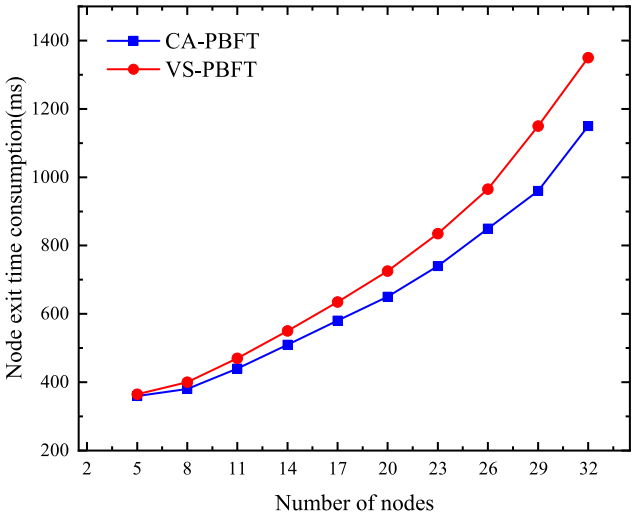

FIGURE 18. Node dynamic exit test analysis.

a better improvement for the expandability, and it requires fewer communication resources.

The number of nodes in the node dynamic exit test starts with $3f + 2$, i.e., the time taken by nodes to exit the current consensus network is tested starting from 5 to evaluate whether the node dynamic exit function meets the performance expectations.

As can be seen from Figure 18, the difference in the time taken to exit the nodes in the two algorithms is small, but as the number of nodes in the consensus network increases, the time taken to exit the nodes is proposed to increase gradually, and the time taken by the present improved algorithm decreases by an average of 10.57% in comparison to VS-PBFT, which can be obtained from the fact that the present improved algorithm has made a better improvement for extensibility, and it requires fewer communication resources.

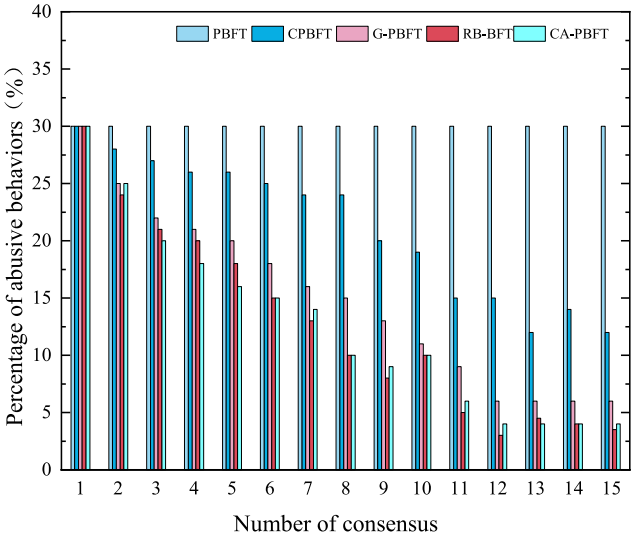

FIGURE 19. Reliability analysis.

5) RELIABILITY ANALYSIS

In the PBFT consensus algorithm, it is not possible to make any processing action for the evil nodes in the consensus network. In this paper, the improved algorithm can eliminate the evil nodes in the consensus network by combining the node credit scoring mechanism and adding the coordination node to the model.

As can be seen from Figure 19, after 15 rounds of consensus, the percentage of evil nodes in this improved algorithm is maintained at around 3.56%. Among them, the G-PBFT is always maintained at around 6% although it also tends to decrease; the RB-BFT, although it is sometimes lower, has an unstable trend; while the PBFT is always inactive for the evil nodes within the consensus network, and can not be eliminated and withdrawn from the process. This proves that the node credit scoring mechanism of this algorithm can largely curb the evil nodes from influencing the consensus network.

6) CONSENSUS LATENCY ANALYSIS

Consensus delay is the time delay required to reach consensus among a group of nodes and is a measure of network performance and running time of the consensus algorithm. The delay calculation is shown in Equation (26).

$$T_{delay} = T_{send} + T_{wait} + T_{consensus} + T_{verify} \quad (26)$$

where T_{send} denotes the time from the initiation of the transaction request by the block to the submission to the consensus network to start the transaction; T_{wait} denotes the time consumed from the reception of the initiation request to the entry of the consistency endorsement; and T_{verify} denotes the time consumed by the node to verify the initiated transaction

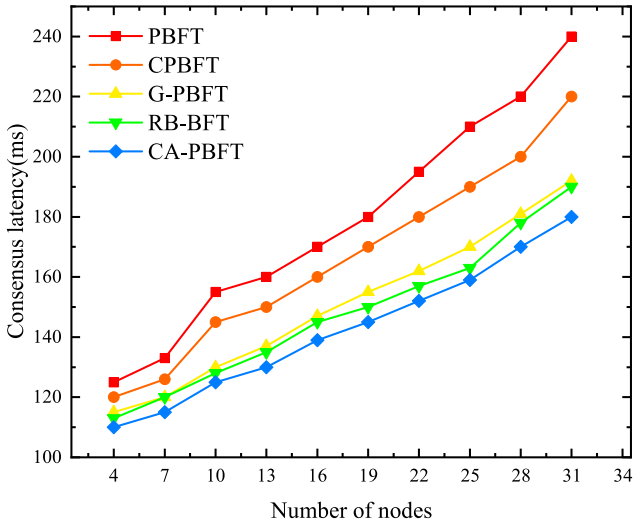

FIGURE 20. Consensus latency test.

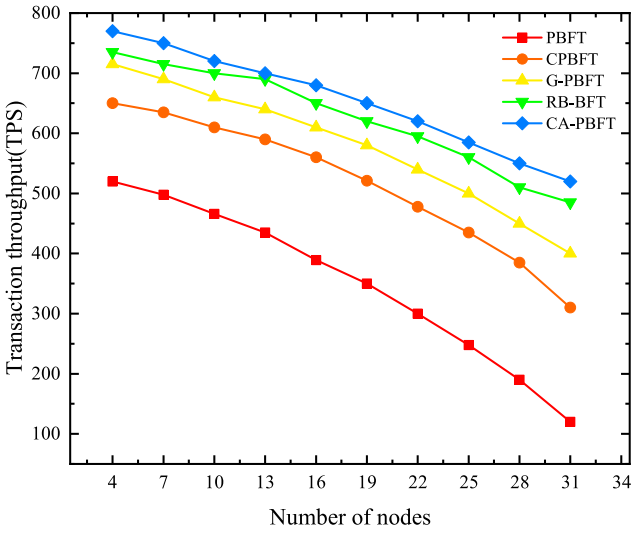

FIGURE 21. Transaction throughput test.

and that the verification passes the write to the blockchain to generate the block.

The delay size is generally closely related to the size of the block size, when the block size is larger, the process of triggering consensus will be more difficult, resulting in a longer T_{wait}; if the block contains more transaction information, it will also make the node verification in the consensus process as well as the execution of the transaction request time longer, resulting in a larger T_{verify}. For this reason, we set fixed conditions for the experiment; and conducted a comparative analysis test in the case of a block size of $100tx$.

As analyzed in Figure 20, when the nodes exceed a certain size, the performance gap between CA-PBFT and PBFT, CPBFT, G-PBFT, and RB-BFT in terms of delay is getting bigger and bigger and occupies more and more advantages, which proves that the algorithm effectively reduces the delay of consensus by making improvements to the signature part of the consensus process and making the signature structure of the consensus process smaller, which compares with the time required for G-PBFT and RB-BFT by about 5.8% and 4.3% in the time required for consensus, respectively.

7) TRANSACTION THROUGHPUT ANALYSIS

The transaction throughput (TPS) performance metric is the number of transactions that can be completed in a given time. In the PBFT consensus algorithm, the transaction throughput performance metric can be defined by the following two factors:

Network Latency: The lower the network latency, the faster messages are exchanged between each node, and the shorter the transaction confirmation time is, thus allowing more transactions to be completed.

Transaction size: the smaller the transaction size, the fewer messages are exchanged between each node, and the shorter the confirmation time for the transaction, thus allowing more transactions to be completed.

This test experiment is only for the process from the time the client sends a transaction request to the time the transaction execution is completed and written to the blockchain in Hyperleger Fabric. The throughput calculation is shown in Equation (27).

$$\mathrm{TPS} = \frac{count_{tx}}{\Delta t} \tag{27}$$

where Δt is the Hyperledger Fabric definition time interval; $count_{tx}$ is the number of valid transactions completed in the Hyperleger Fabric within Δt.

In this experiment for transaction throughput, it is proposed to set different numbers of nodes to compare and analyze, according to the number of $3f + 1$, the number of nodes since the beginning of 4, the transaction throughput test results are shown in Figure 21.

As can be seen in Figure 21, the PBFT, CPBFT, G-PBFT, RB-BFT, and CA-PBFT algorithms all show a trend of decreasing transaction throughput as the number of nodes in the consensus network increases. This is because when the number of nodes increases, the communication complexity in the consensus network becomes higher and higher due to the P2P communication mode used in the blockchain. Also, the throughput is related to the size of nodes in the current consensus network. In the traditional PBFT consensus algorithm, the communication complexity is $O(n^2)$, while the communication complexity of the CA-PBFT consensus algorithm is only $O(n)$, which greatly alleviates the problem of decreasing transaction throughput due to the large node size of the consensus network, and ensures high transaction throughput in the face of large-scale node access.

With the combined effect of the aggregated signature scheme and the node credit scoring mechanism, the

transaction throughput of CA-PBFT is improved by about 14.47% and 9.8% over G-PBFT and RB-BFT, respectively.

V. CONCLUSION AND FUTURE WORK

In this paper, we propose an improved solution to the problems of low consensus efficiency and poor scalability of the PBFT consensus algorithm. By introducing a node credit scoring mechanism, coordinating nodes, and a BLS-based aggregated signature scheme, we have improved consensus efficiency, solved the problem of poor scalability, and reduced communication complexity to some extent. However, our program still has some limitations and challenges. First, the current node credit scoring mechanism is not yet perfect in terms of credit rewards and penalties. For nodes with too high credit value, they may be allowed to perform multiple consensuses in case of jeopardizing the consensus process behavior, resulting in a waste of communication resources. To solve this problem, we need to design more reasonable credit reward and punishment mechanisms in future research to prevent abuse by malicious nodes. Second, the introduction of coordination nodes may exacerbate the centralization problem in the consensus network. To solve this problem, we can consider designing a decentralized coordination node election mechanism to ensure the decentralized nature of the consensus network. Again, although we have partially verified the security of the BLS-based aggregated signature scheme, more in-depth security research is needed before applying it to real enterprise scenarios. Regarding the application scope of the improved algorithm, we believe that it may be suitable for distributed systems that require efficient consensus and scalability, such as finance, IoT [32], [33], healthcare, and supply chains. However, in practical applications, the algorithms need to be further adapted and optimized according to specific scenarios and needs. In addition, advanced security technologies based on AI/ML have been widely used in future networks, and these technologies provide us with new ideas to further optimize our consensus mechanism by introducing AI/ML technologies [34]. In addition to the improved methods mentioned above, future research can also explore the use of convolutional neural network (CNN) [35] to improve consensus algorithms. Convolutional neural networks have achieved remarkable results in the fields of image recognition and pattern recognition, and have the advantages of parallel computing and automatic feature extraction. Therefore, the introduction of convolutional neural networks into consensus algorithms is expected to improve the accuracy and efficiency of consensus. How to effectively integrate these techniques into the consensus mechanism still requires further exploration and experimentation in our future research.

In conclusion, our improvement scheme provides a useful attempt to address the limitations of the PBFT consensus algorithm, but it still needs to be further improved and refined in future research. We hope to provide more efficient, secure, and scalable solutions for consensus algorithms for distributed systems through our continuous efforts.

REFERENCES

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," Decentralized Bus. Rev., Bitcoin, White Paper, Nov. 2008.

[2] T. Huynh-The et al., "Blockchain for the metaverse: A review," *Future Gener. Comput. Syst.*, Jun. 2023.

[3] J.-S. Lee, C.-J. Chew, J.-Y. Liu, Y.-C. Chen, and K.-Y. Tsai, "Medical blockchain: Data sharing and privacy preserving of EHR based on smart contract," *J. Inf. Secur. Appl.*, vol. 65, Mar. 2022, Art. no. 103117.

[4] S. Kumar, B. Kumar, Y. Nagesh, and F. Christian, "Application of blockchain technology as a support tool in economic & financial development," *Manager-Brit. J. Administ. Manag.*, Apr. 2022, pp. 1278–1746.

[5] F. Azzedin and M. Ghaleb, "Internet-of-Things and information fusion: Trust perspective survey," *Sensors*, vol. 19, no. 8, p. 1929, Apr. 2019.

[6] Q. Xia, E. Sifah, A. Smahi, S. Amofa, and X. Zhang, "BBDS: Blockchain-based data sharing for electronic medical records in cloud environments," *Information*, vol. 8, no. 2, p. 44, Apr. 2017. [Online]. Available: https://www.mdpi.com/2078-2489/8/2/44

[7] R. Kumar, N. Marchang, and R. Tripathi, "SMDSB: Efficient off-chain storage model for data sharing in blockchain environment," in *Machine Learning and Information Processing*. Singapore: Springer, Apr. 2021, pp. 225–240.

[8] P. K. Wan, L. Huang, and H. Holtskog, "Blockchain-enabled information sharing within a supply chain: A systematic literature review," *IEEE Access*, vol. 8, pp. 49645–49656, Mar. 2020.

[9] M. Jakobsson and A. Juels, "Proofs of work and bread pudding protocols," in *Proc. Secure Inf. Netw. Commun. Multimedia Secur. IFIP TC6/TC11 Joint Workshop Conf. Commun. Multimedia Secur. (CMS)*. Boston, MA, USA: Springer, Sep. 1999, pp. 258–272.

[10] F. Saleh, "Blockchain without waste: Proof-of-stake," *Rev. Financial Stud.*, vol. 34, no. 3, pp. 1156–1190, Feb. 2021.

[11] D. Larimer, "Delegated proof-of-stake (DPoS)," BitShare, White Paper, 2014, vol. 81, p. 85.

[12] D. Ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," in *Proc. USENIX Annu. Techn. Conf. (USENIX ATC)*, Jun. 2014, pp. 305–319.

[13] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proc. OsDI*, vol. 99, 1999, pp. 173–186.

[14] W. Zhong, W. Feng, M. Huang, and S. Feng, "ST-PBFT: An optimized PBFT consensus algorithm for intellectual property transaction scenarios," *Electronics*, vol. 12, no. 2, p. 325, Jan. 2023.

[15] M. Vukoli, "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication," in *Proc. Int. Workshop Open Problems Netw. Secur.* Cham, Switzerland: Springer, May 2016, pp. 112–125.

[16] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, pp. 382–401, Jul. 1982.

[17] M. Castro and B. Liskov, "Practical Byzantine fault tolerance and proactive recovery," *ACM Trans. Comput. Syst.*, vol. 20, no. 4, pp. 398–461, Nov. 2002.

[18] G. Xu and Y. Wang, "Improved PBFT algorithm based on vague sets," *Secur. Commun. Netw.*, vol. 2022, pp. 1–7, Mar. 2022.

[19] Y. Wang, Z. Song, and T. Cheng, "Improvement research of PBFT consensus algorithm based on credit," in *Proc. Int. Conf. Blockchain Trustworthy Syst.* Singapore: Springer, 2020, pp. 47–59.

[20] B. Gan, Y. Wang, Q. Wu, Y. Zhou, and L. Jiang, "EIoT-PBFT: A multi-stage consensus algorithm for IoT edge computing based on PBFT," *Microprocess. Microsyst.*, vol. 95, Nov. 2022, Art. no. 104713.

[21] L. Lao, X. Dai, B. Xiao, and S. Guo, "G-PBFT: A location-based and scalable consensus protocol for IoT-blockchain applications," in *Proc. IEEE Int. Parallel Distrib. Process. Symp. (IPDPS)*, May 2020, pp. 664–673.

[22] L. Lei, L. Song, and J. Wan, "Improved method of blockchain cross-chain consensus algorithm based on weighted PBFT," *Comput. Intell. Neurosci.*, vol. 2022, pp. 1–9, Aug. 2022.

[23] P. Li, G. Wang, X. Chen, F. Long, and W. Xu, "Gosig: A scalable and high-performance Byzantine consensus for consortium blockchains," in *Proc. 11th ACM Symp. Cloud Comput.*, Oct. 2020, pp. 223–237.

[24] Y. Na, Z. Wen, J. Fang, Y. Tang, and Y. Li, "A derivative PBFT blockchain consensus algorithm with dual primary nodes based on separation of powers-DPNPBFT," *IEEE Access*, vol. 10, pp. 76114–76124, 2022.

[25] J. Yang, Z. Jia, R. Su, X. Wu, and J. Qin, "Improved fault-tolerant consensus based on the PBFT algorithm," *IEEE Access*, vol. 10, pp. 30274–30283, 2022.

[26] W. Jiang, X. Wu, M. Song, J. Qin, and Z. Jia, "Improved PBFT algorithm based on comprehensive evaluation model," *Appl. Sci.*, vol. 13, no. 2, p. 1117, Jan. 2023.

[27] J. Zhang, Y. Rong, J. Cao, C. Rong, J. Bian, and W. Wu, "DBFT: A Byzantine fault tolerance protocol with graceful performance degradation," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 5, pp. 3387–3400, Sep. 2022.
[28] F. He, W. Feng, Y. Zhang, and J. Liu, "An improved Byzantine fault-tolerant algorithm based on reputation model," *Electronics*, vol. 12, no. 9, p. 2049, Apr. 2023.
[29] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *Proc. EUROCRYPT*, 2003, pp. 416–432.
[30] I. Shparlinski, "Computational Diffie–Hellman problem," in *Encyclopedia of Cryptography and Security*, H. C. A. van Tilborg and S. Jajodia, Eds., Boston, MA, USA: Springer, 2011, pp. 240–244.
[31] C. P. Schnorr, "Efficient signature generation by smart cards," *J. Cryptol.*, vol. 4, no. 3, pp. 161–174, Jan. 1991.
[32] J. Du, C. Jiang, E. Gelenbe, L. Xu, J. Li, and Y. Ren, "Distributed data privacy preservation in IoT applications," *IEEE Wireless Commun.*, vol. 25, no. 6, pp. 68–76, Dec. 2018.
[33] A. Khalil, N. Mbarek, and O. Togni, "Fuzzy logic based security trust evaluation for IoT environments," in *Proc. IEEE/ACS 16th Int. Conf. Comput. Syst. Appl. (AICCSA)*, Nov. 2019, pp. 1–8.
[34] J. Du, C. Jiang, J. Wang, Y. Ren, and M. Debbah, "Machine learning for 6G wireless networks: Carrying forward enhanced bandwidth, massive access, and ultrareliable/low-latency service," *IEEE Veh. Technol. Mag.*, vol. 15, no. 4, pp. 122–134, Dec. 2020.
[35] G. Muhammad and M. Alhussein, "Security, trust, and privacy for the Internet of Vehicles: A deep learning approach," *IEEE Consum. Electron. Mag.*, vol. 11, no. 6, pp. 49–55, Nov. 2022.

SHIHUA TONG is currently pursuing the Ph.D. degree in education with Southwest University. He is a Professor with the Chongqing College of Electronic Engineering and a Master's Tutor with the Chongqing University of Posts and Telecommunications and a model of Teaching and Educating People in Chongqing. He presided over/researched more than 40 provincial and ministerial scientific research projects. He has published more than 40 academic papers, more than 70 authorized patents/soft works, five monographs, and two popular science books. His research interests include education and teaching reform, embedded and artificial intelligence, network security, and blockchain. He was the Winner of the Outstanding Teacher Award from the National Huang Yanpei Vocational Education Award. In recent years, he has won one (first prize) and one (second prize) for national teaching achievements; one special prize, two (first prize), and two (second prize) for provincial and ministerial teaching achievements; one (second prize) for the Chongqing Science and Technology Progress Award; and one (first prize) for the Chongqing Excellent Education and Scientific Research Achievements. He was selected for the Chongqing High-Level Talent Special Support Program (Famous Teacher) and the Chongqing University Excellent Talent Support Program. He was the Chongqing University Innovation Research Group Leader, the Chongqing Famous Teacher Training Program Famous Teacher Studio Leader, the Vice Chairperson of the Chongqing Youth Scientific Quality Research Association, the Chongqing Vocational Education Evaluation and Certification Expert, the Chongqing Smart Community Construction Consulting Expert, and Chongqing Technical Education Expert Think Tank Expert.

JIBING LI is currently pursuing the degree in electronic information with the School of Automation, Chongqing University of Posts and Telecommunications. He has participated in one national key research and development program project and one major/key project in Chongqing Municipality. He has disclosed two invention patents and applied for one soft work. His research interests include blockchain technology, information security, and cryptography technology.

WEI FU is currently a Professor with the Chongqing University of Posts and Telecommunications, the Director of the Chinese Society of Automation-Chongqing University of Posts and Telecommunications, and an Observer of the National Standardization Technical Committee for Intelligent Buildings and Residential Digitization (SAC/TC426). She presided more than one national scientific and technological major special project, two sub-projects of the national 863 project, and observer. She also presided more than five major/key projects in Chongqing and more than 20 general projects in Chongqing. She has published more than ten papers, 15 SCI/EI retrieved papers, 20 authorized invention patents, and nine utility model patents. She has co-edited three textbooks and two monographs. Her main research interests include the Internet of Things technology, smart cities, smart healthcare, blockchain technology, and wireless sensor networks and their applications.