An Optimization Strategy for PBFT Consensus Mechanism Based On Consortium Blockchain

Feilong Wang[1*], Yipeng Ji[2*], Mingsheng Liu[3*], Yangyang Li[4†], Xiong Li[5,6], Xu Zhang[7], Xiaojun Shi[8]
[1]SISE, Yanshan Univerity, Qinhuangdao, China; [2]SCSE, Beihang University, Beijing, China;
[3]Shijiazhuang Institute of Railway Technology, Shijiazhuang, China; [4]National Engineering Laboratory for Public Safety Risk Perception and Control by Big Data, CAEIT, Beijing, China;
[5]Beijing Biwei Network Technology Co., Ltd., Beijing, China; [6]Beijing Xindatek Technology Co., Ltd., Beijing, China;
[7]CNCERT, Beijing, China; [8]Department of Science and Technology, China Electronics Technology Group Corporation
wangfeilong@stumail.ysu.edu.cn;jiyipeng@buaa.edu.cn;liums601001@sina.com;
liyangyang,shixj@cetc.com.cn;zhangxu@cert.org.cn;li.xiong@foxmail.com

ABSTRACT

At present, the transaction delay of the consortium block chain applying the Practical Byzantine Fault Tolerance (PBFT) consensus protocol can only reach 2 to 5 seconds, and the throughput cannot reach tens of thousands. In addition as the number of nodes increases, the performance of the consortium block chain declines very quickly. The main challenge of previous research are to realize communication network topology of PBFT algorithm and high information exchange in the case of Byzantine failure, thus, this paper proposes an optimized Byzantine fault-tolerant algorithm to solve the performance bottleneck of the consortium chain. First of all, for the communication network structure of the whole network broadcast, we have reached an agreement on the transaction according to the pre-prepare and prepare phases of PBFT, and generally enter the commit phase, there is a high probability that the leader is honest, so we will communicate with the commit phase The network is optimized as a star communication structure. Second, combined with Tendermint, merge the view-change process of Byzantine failures of the normal consensus process, and switch the leader according to the longest chain principle. The algorithm is based on a partially synchronized network model to ensure the security and liveness of the protocol, and improve the performance and effective robustness.

CCS CONCEPTS

• **Computer systems organization** → **Availability**; • **Networks** → *Network protocol design*; • **Security and privacy** → Distributed systems security.

[*] Feilong Wang, Yipeng Ji and Mingsheng Liu are contributed equally to this work.
[†] Yangyang Li is the corresponding author.

BSCI '21, June 7, 2021, Virtual Event, Hong Kong.

ACM ISBN 978-1-4503-8400-1/21/06…$15.00
https://doi.org/10.1145/3457337.3457843

KEYWORDS

Consortium Blockchain, Performance, Byzantine Fault Tolerant, Robustness, PBFT

ACM Reference Format:
Feilong Wang[1*], Yipeng Ji[2*], Mingsheng Liu[3*], Yangyang Li[4†], Xiong Li[5,6], Xu Zhang[7], Xiaojun Shi[8]. 2021. An Optimization Strategy for PBFT Consensus Mechanism Based On Consortium Blockchain. In *Proceedings of the 3rd ACM International Symposium on Blockchain and Secure Critical Infrastructure (BSCI '21), June 7, 2021, Virtual Event, Hong Kong*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3457337.3457843

1 INTRODUCTION

In recent years, the performance[7] bottleneck of the consortium block chain consensus protocol has been restricting the development of the consortium block chain. And the fault tolerance of the known consensus protocol has always been controversial. When a carefully designed malicious behavior occurs to the consortium chain system, the protocol cannot completely tolerate errors and operate normally. This situation will cause the system performance to be very inefficient or even destroyed. Consistency of the agreement[19]. Although there have been many relatively practical consensus protocols (for example PBFT, Tendermint) in academia, the protocol must increase the cost of node communication time while ensuring consistency and termination. And when a Byzantine error occurs, the performance of the protocol drops very quickly, resulting in a huge difference between the Byzantine situation and the performance under normal operating conditions. If the proposer is malicious while the protocol is running, it will have worse results (forking). Therefore, it is very meaningful to design a high-performance consensus protocol that satisfies the asynchronous[16] environment and Byzantine fault tolerance.

Currently, many consensus algorithms based on distributed consensus have emerged from academia. Based on the consortium chain scenario, the same can also be applied to this. As the first-generation Byzantine fault-tolerant consensus protocol, the DLS[5] consensus protocol proposes a synchronization hypothesis. After the global stable time GST, the system tends to be synchronized. This situation can be completely Byzantine fault-tolerant. The protocol uses the synchronization assumption to overcome the FLP[6] impossibility theorem, and is based on the single-round[23] voting model. In order to ensure security, the protocol adopts the method of mutual communication and notification between nodes (full

network broadcast), which makes the complexity of information communication reaches $O(N^2)$. In addition, it uses an independent synchronized clock to synchronize time. In actual situations, the clock generally deviates from time to time "clock drift", and it is easy to become an attack target of a Byzantine environment, leading to security issues for the protocol. Therefore, in 1999, on the basis of the DLS consensus protocol, the PBFT consensus protocol proposed by Miguel Castro and Barbara Liskov, security can tolerate (n-1)/3 Byzantine node[9]. However, this protocol also has corresponding problems after overcoming some of the defects of DLS. For example, PBFT is based on a two-round voting model, which also increases the amount of information exchanged and the complexity of communication (complexity issues) and when a failure occurs in order to ensure the activity of the system, a view-change will be generated Event, in this case, the performance will be exponentially lower than normal.

In this paper, based on the performance and security[13] problems of the PBFT (Practical Byzantine Fault Tolerance) consensus protocol, we have introduced a longest chain election rule. At the beginning of each round of consensus, according to the latest local node, block height is used to elect the leader node, and the leader node with the largest height is elected. The election of the honest node as the leader avoids the proposal of the Byzantine node, reduces the possibility of the master node doing evil, and combines a dynamic node list to replace the consensus nodes that participated in the previous round[28]. We also achieve unlimited parallel expansion of the number of nodes, so the performance is not affected by the increase in the number of nodes. We divide node roles as two roles in consensus, consensus roles and candidate roles. When a new round of consensus is started, the candidate nodes will replace the nodes that participated in the previous round of consensus, and the role will be changed from candidate nodes to consensus nodes to achieve dynamic authorization of nodes to participate in the consensus. In terms of performance, optimize the commit phase of PBFT and improve the communication topology of the entire network of a star network structure. The replica nodes only need to exchange information about the leader node, and the threshold signature[17] mechanism is introduced. In the commit phase, the replica node only needs to communicate with the leader node, and the master node determines the final decision value based on the collected information. As a result, the communication complexity of the commit phase is reduced from $O(N^2)$ to $O(N)$, and the liveness and safety of the protocol are guaranteed through the lock mechanism [5, 8] and the empty block mechanism. And combined with the Tendermint consensus protocol, the view-change phase is integrated into the normal phase of the consensus, so that the performance is not much different from the normal situation when a Byzantine error occurs, and the consistency and termination of the algorithm can be guaranteed.

2 RELATED WORK

The first major problem solved by the consortium chain consensus protocol is the Byzantine generals problem. Regarding the consistency problem with Byzantium, Lamport et al. propose two solutions. The OM verbal agreement can prevent the traitor generals from less than one-third of the total number of generals One situation reached an agreement; SM has a written agreement that can reach an agreement in the case of any number of generals and possible traitors. But the shortcoming of these two solutions is that a large amount of information exchanges is required to ensure security. On the basis of the above two solutions, Miguel Castro and Barbara Liskov proposed a new Byzantine fault-tolerant state machine[14] replication algorithm in [2], PBFT, which reduces the amount of information exchange with communication and improves performance. More than double the order for magnitude, but still can be further optimized in terms of algorithm resource consumption. For the performance problems of the BFT protocol, many improved BFT consensus protocols have emerged based on the PBFT protocol, such as Ripple, Tendermint, Zyzzyva, SBFT, BA, HoneyBadgerBFT, Gosig, Tangaroa, and so on. The above protocol optimizes the protocol process of different levels and aims to improve the performance of the consensus algorithm.

This part is the improvement strategy of different BFT protocols. Ripple consensus algorithm addresses the problem of high latency of existing Byzantine fault-tolerant algorithms in asynchronous networks, and proposes to use trusted sub-networks in collective networks to circumvent this problem. On the basis of PBFT, Tendermint simplifies the process of the BFT[11] protocol based on the gossip protocol[3], merges the leader switching process of the normal process, and uses a novel mechanism to ensure termination. For the time being, the status of the replicas is inconsistent, and then the request sequence is corrected to be positive. Such a process has a significant performance improvement compared with the existing BFT protocol. SBFT[24] adopts four optimization strategies, collector, threshold signature[17], optimistic fast path, redundant servers. Compared with the PBFT consensus algorithm in performance, it can provide twice the throughput and 1.5 times the transaction delay. This allows it to provide significantly better performance in large-scale distributed systems (such as consortium chains).

Algorand[25] adopts a new Byzantine protocol BA, and is based on verifiable random function to increase the flexibility of participating in consensus. Nodes can privately check whether to participate in the consensus process of the next set of transactions. Fault tolerance avoids malicious targeted attacks on selected consensus nodes, and can achieve transaction delays of less than one minute. The Tangaroa[27] protocol is a Byzantine fault-tolerant consensus protocol based on the Raft protocol, which increases the complexity of the protocol to ensure security, fault tolerance and liveness.

Different from the above BFT protocol optimization strategy, the protocol proposed in this paper still guarantees consistency and activity in the asynchronous network environment, and can provide good performance and good robustness. The protocol uses a longest chain election rule and a dynamic node replacement list to increase the security and scalability of the protocol, and optimize the communication mode in the core phase Commit phase based on the PBFT consensus protocol, reducing the secondary communication process to linear process. And through the threshold signature, Gossip mechanism to reduce the workload of the agreement, aiming to achieve the purpose of reducing transaction delay and improving performance.

3 SYSTEM MODEL

In this Byzantine fault-tolerant model, we assume that the system is composed of a set of node lists $N = \{n_0, n_1...\}$ and any number of clients. The clients and nodes must pass Identity verification can participate in the system and communicate with each other through the p2p[22] network. And suppose there is a list of faulty nodes $C = \{c_0, c_1...\}$ in the node list, and a list of Byzantine nodes $F = \{f_0, f_1, ...\}$, the list of loyal nodes $G = \{g_0, g_1, ...\}$, and the number of loyal nodes satisfies $G > max(C, F)$. And any number of malicious clients.

Possible failures of the failed node:

(1) Node is down

(2) node cannot send/receive information

Possible Byzantine error:

(1) Node maliciously rejects/sends messages

(2) Node maliciously sends inconsistent information

(3) The node cooperates with other malicious copies to commit evil

(4) Malicious client behavior

We assume that the network is based on a partially synchronized network model. During the operation of the system, the communication between nodes and the communication between the client and the node can ensure that there is a certain upper bound Δ for the delay in sending and receiving information, but not The specific size of the delay is unclear. That is to say, although the information will be delayed, the message will be received after a period of time. And set a timeout clock at each node to calculate the message transmission timeout. Based on the above assumptions and restrictions, our agreement can ensure the activity of the BFT algorithm.

This model is based on Byzantine nodes and loyalty nodes with some restrictions. 1. Byzantine nodes cannot forge the identity of loyal nodes and tamper with information, but can only forge the identities of other Byzantine nodes. 2. The loyal node can accurately send information on the correct node. 3. The node can clearly know the identity of the sender of the information. 4. Missing information can be detected. Based on these restrictions, loyal nodes can make the same correct decision, and Byzantine nodes cannot interfere with the execution process of loyal nodes, thereby ensuring the consistency of the BFT algorithm.

4 PROTOCOL

4.1 Normal Path

In the FLP Impossibility Theorem, it has been proved that in the distributed consensus algorithm, even if there is an unreliable communication process, it will cause the consensus to fail to reach a consensus result. So our algorithm is based on the PBFT consensus protocol and built on a partially synchronous communication model. And the algorithm is a consensus process based on round voting. Each round corresponds to a height. Each round contains three main workflows, pre-prepare, prepare, and commit. Its algorithm is similar to the Tendermint consensus algorithm for master node switching in the Byzantine situation. It also uses a lock mechanism and an empty block mechanism to integrate the view change process in a Byzantine error situation into a normal round. Different from the PBFT consensus protocol, this algorithm improves the voting process of the entire network broadcast in the commit phase to a star network voting. During the commit voting process, each replica node sends the voting results to the master node, and the master node collects the votes. The node is equivalent to a collector, and finally combines the voting results to send a message to the replica node whether to commit the proposal. Such an improvement will make the communication complexity of the commit process linearly complex.

When each round is opened, only nodes in the same height can participate in the consensus voting for that round. The remaining nodes state in idleness. Each round of consensus includes a pre-elected master node and some replica nodes, and the replica nodes are in the same state. And the next round of consensus will re-elect the master node and replica node. This algorithm uses a longest chain election rule and a dynamic node list to select consensus nodes. At the beginning of each round, the election of the master node is based on the latest height of the local block chain. The leader node with the largest block height is elected to ensure that the block chain is up-to-date and reliable, and will be selected before the start of the next round of consensus. Replacing the consensus nodes that participated in the previous round from the dynamic node list ensures the security of the entire system and prevents some malicious nodes from attacking the system.

pre-preapre phase: The new round corresponds to a height. When the new round starts, the node also enters the pre-prepare phase. At the beginning of this phase, if the master node does not lock the block of the previous round, it will construct a block based on the transaction information submitted by the client, and encapsulate the status information about the master node into a message packet and broadcast it to other replica nodes. Otherwise, the locked block will be regarded as the proposed block, and the proof of the locked block will be attached (2f+1 pieces of prepare voting information were received in the previous round). After the master node broadcasts the proposal message, it will start a timer. If the proposal message sent by the master node is not received within the timeout period, the node will vote against the proposal (prevote for nil), which is equivalent to voted for an empty proposal. If the node receives the proposal, it will verify the validity of the proposed block, including verifying the status message and signature in the proposal message. At the end of the pre-prepare stage, the verification result (approve or oppose the proposal) will be encapsulated into a prepare message and broadcast to other replica nodes.

prepare phase: After the node broadcasts the prepare voting information, the node enters the prepare phase. At this stage, nodes will collect voting information (prepare messages) of each node on the proposal. Collecting prepare votes at this stage mainly includes four situations. The first case is that the 2f+1 prepare message about the proposed block is received normally, and the legality of the received prepare message is checked. If the legality check passes, the proposed block of this round will be locked(lockedBlock and lockedView). The second case is that 2f+1 prepare messages are received, but it is a prepare message for the empty block nil. The third case is that the received legal prepare message about the proposed block proposal or the empty block nil is less than 2f+1, the fourth case is No 2f+1 prepare messages were received within the timeout period of this phase. When the replica node belongs to the

first case, that is, under normal circumstances, it will vote for the proposed block proposal and send a pre-commit proposal message to the leader; the remaining three cases are all sent to the leader pre-commit nil message to the leader. At the end of the prepare phase, the master node will collect the pre-commit message sent by the replica node.

commit phase: The improvement strategy of this protocol is mainly for this stage. Figure1 and Algorithm1 provides an overview of the entire commit after optimization.When the replica node sends the pre-commit message, it enters the commit phase. The purpose of this phase is to execute the proposed block and place the block in the local database. The leader node will collect the pre-commit message of the replica node and verify the legitimacy of the message, and will check whether the collected legal pre-commit message satisfies f+1. If there are f+1 pre-commit messages for the proposed block proposal, a proof of the f+1 pre-commit messages will be generated and encapsulated in the commit message and signed and broadcast to the replica node, otherwise an empty area will be broadcast Block nil commit message; if f+1 empty block pre-commit messages are collected or f+1 any proposed pre-commit messages are not collected, then a commit nil message will also be signed and broadcast . The replica node receives the commit proposal message and has the opportunity to write the proposed block into the log. The block height is increased by one to enter the consensus of a new height. Otherwise, the height remains unchanged. The next round of consensus on the proposal will continue.

Synchronization phase: At the end of a round of consensus, our protocol adds a new synchronization process, which aims to align the state of the failed node with the replica node. Because during the commit phase, it may be down or offline due to network reasons or the node's own reasons, which will cause the submitted block to time out and not be written to the ledger. This is not a malicious error, and the uncommitted block itself is a consensus. And is recognized, so the faulty node can request the missing blocks from other normal nodes. When the failed node recovers, it will broadcast a status request message to other nodes, and other nodes will feed back the latest height of their local ledger and the hash of the block according to the type of the message. If the faulty node locked a block in the previous round, and the corresponding hash of the block is equal to the feedback block hash, it can be directly written to the block, and the missing block can be directly synchronized from other nodes.

Figure2 describes the entire workflow of the improved protocol. The consensus of each round includes three stages: pre-prepare, prepare, and commit.

4.2 View Change

This protocol does not have an explicit view change phase like most BFT consensus protocols. For example, PBFT is specifically the view-change process of the master node switching in the Byzantine situation. When enough nodes suspect that the leader node is a Byzantine node, they will actively broadcast a view-change message and collect enough 2f+1 valid messages according to whether The view-change message then becomes the new leader. The entire view-change phase is very time-consuming, therefore, the performance of PBFT in the Byzantine situation is very different from the performance under normal conditions. In response to the above situation, we learned from Tendermint's processing method and integrated view-change into the normal consensus process instead of dealing with this special situation separately.

In the Byzantine situation, the block may not be placed in the current round. The purpose of view-change is to switch the Byzantine leader to a loyal node, and put the blocks that have not been placed in this round to the next round and continue to complete the placement when the leader node is loyal. Therefore, our agreement draws on the Tendermint consensus protocol, and only needs to complete the work that the view-change needs to complete in the normal consensus process. First, at the beginning of a new round, actively switch the leader node. Secondly, at the end of the round, if the block has not been placed due to a Byzantine error, you can lock the block for the round first, and then place an empty block, which will not affect the security of the protocol and cause a fork. Finally, at the beginning of the next round (now the new leader), unlock the locked round and block, and use the unlocked block as the proposed block to continue the normal consensus process. Unlike the view-change process of most BFT protocols, our protocol does not require network bandwidth requirements and a large amount of information exchange, so the performance under the Byzantine situation is not much different from that under normal conditions.

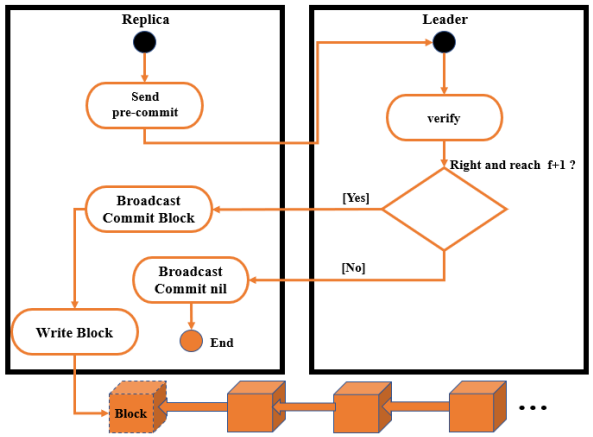

Figure 1: Overview of commit phase

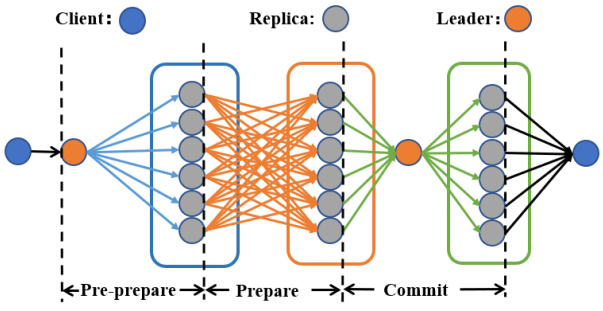

Figure 2: Overview of a protocol round

Algorithm 1: Consensus Algorithm

Data: replica i, current block $b(i)$, current round $r(i)$, current height $h(i)$
Result: block to be submitted *decision*

commit phase:
for i *in* $N\{0, 1, 2..i, i+1..\}$ **do**
  if i *lock* $b(i)$ **then**
    send $< Pre-commit, h(i), r(i), hash(b(i)) >$ to leader
  else
    send $< Pre-commit, h(i), r(i), NULL >$ to leader
  end
end
if *leader receive* $f+1$ $< Pre-commit, h(i), r(i), hash(b(i)) >$ **then**
  broadcast $< Commit, h(i), r(i), hash(b(i)) >$
  $decision \leftarrow b(i)$
  $h(i) \leftarrow h(i)+1$
  $newRound(r(i) \leftarrow 0)$
else
  broadcast $< Commit, h(i), r(i), NULL) >$
  $r(i) \leftarrow r(i)+1$
end

5 SAFETY AND LIVENESS

5.1 Safety

5.1.1 Loyalty nodes will not commit two different blocks B and B′ at the same height. In our hypothetical system model(section III) , the number of loyal replicas is assumed to be greater than the maximum number of failed replicas and Byzantine replicas, that is, greater than one-third of the total number of replicas. When a loyal replica submits a block B at height h, it means that the leader node has received at least f+1 pre-commit messages about block B, which proves that more than f+1 replicas have received 2f+1 corresponding areas The prepare message of block B. If another block B' is submitted, it is proved that more than f+1 replicas have received the prepare message corresponding to block B' of 2f+1, which means that f+1 Byzantine nodes voted for B and B' at the same time,but the system assumes that Byzantine nodes do not exceed one-third of the total number of nodes. Obviously, this situation does not exist.

There are two different blocks at the same height. The reason for this situation is probably that the leader node is a Byzantine node. In the pre-prepare phase, the leader node sends block B and B' proposals to different replicas. The communication mechanism and round voting mechanism of the protocol can avoid the occurrence of bifurcation, which guarantees the security of the protocol.

5.1.2 Block B proposed by the Byzantine leader node will not be written to the database. When the client submits a request to the system, the request will be broadcast to each replica node in the system, including the leader node. The leader node sorts and packs into blocks according to the request, and enters the pre-prepare phase to broadcast to the replica node Proposal block. If the leader node is a Byzantine node, the order of the request may be disrupted and then broadcast to other nodes, and other nodes will have the requested backup locally. When the proposal is received, the replica node will verify the validity of the request. Once invalid, it will Vote against, so the leader Byzantine node cannot allow the loyal replica node to submit invalid blocks, thereby destroying the security of the protocol.

This situation exists. When the system enters the Commit phase, if the leader node is a Byzantine node, it may send inconsistent or even not send commit messages to other nodes. There are several scenarios here: 1. If the leader node receives the pre-commit message of a certain block of f+1, and sends a message of <commit,h,r,nil> to other replica nodes, and the mechanism of the protocol The f+1 pre-commit message QC certificate[**?**] will be attached when sending the commit message, and other replica nodes will verify this QC to understand whether the message sent by the leader node is true, and the loyal node will make the correct decision based on the verification result. 2. The leader node will not broadcast the commit message to other nodes regardless of whether it receives f+1 pre-commit messages. Once the protocol's timeout mechanism detects that the commit message has not been received over time, it will place an empty block and proceed to download Re-elect the master node once and continue to formulate the block in the previous round to ensure the consistency of the agreement.

5.2 Liveness

Since our leader node is generated through a random verifiable function, no Byzantine node can interfere with the process. And there is no way to increase the probability of a Byzantine node being elected leader. Although the Byzantine node may be elected as the leader, under the assumptions of the system model and the rules of the agreement, the Byzantine node cannot destroy the termination of the agreement, that is, its liveness. The protocol can run forever under the system model, unless the system is actively shut down.

This protocol is based on a partially synchronized network model, that is, information transmission has a definite but unknown upper bound. When the protocol is running, the nodes may wait for messages to be transmitted to each other. For example, the nodes are in the prepare phase, and the nodes wait for each other's prepare message. The node will collect 2f+1 prepare messages. If the collection is full, it will enter the commit phase, but The node will not wait for the collection of prepare messages all the time, because this will destroy the activity of the protocol, so we set a timeout timer and wait for a limited time. Once it times out, the protocol will continue to run.

5.2.1 Any honest replica locks a block B in round r, and block B will be committed in round r′ >= r. When the node locks the block B in the round r, it means that the node has entered the commit phase and has collected 2f+1 prepare messages from different replica nodes. At this time, the nodes in the agreement have reached an agreement on block B, and only block B needs to be placed in the commit phase. However, in round r, there may be a faulty node or the block cannot be placed on the market due to network reasons. The solution of our agreement is to reach agreement on the block height of each node through a synchronization process.

To make matters worse, the leader node is a Byzantine node, which will deliberately destroy the order for block B in round r, that is, make the block height between nodes inconsistent. For example, there is a situation where the leader broadcasts different blocks and B'to the nodes in the pre-prepare phase, so that in the end, because the two blocks cannot reach a consensus (security) at the same height, an empty block will be placed. Block B will continue the consensus process in the r+1 round until the node status is consistent.

5.2.2 In each round of consensus, there are enough consensus nodes. If more than one-third of honest nodes are locked on different blocks in different rounds, the leader will eventually broadcast the proof of the corresponding locked block on the later round, which will make it more reliable The locked node on the previous round is unlocked. The unlocked nodes can continue to vote on subsequent proposals, which can ensure that enough nodes participate in the consensus, thereby ensuring the Liveness of the system.

In addition, the timeout period for each node to wait to receive the complete proposal block sent by the leader and possible proof information can be increased as the round progresses, so that when the size of the proposal is relatively fixed, the entire network can guarantee the proposed block Received by each node.

6 CONCLUSION

The PBFT (Practical Byzantine Fault Tolerance) consensus algorithm is based on a two-round voting mechanism, which greatly improves the performance of the alliance chain, but its complex communication structure and inefficient robustness make the performance of the alliance chain not good. Therefore, we have improved and optimized based on the PBFT consensus protocol. Compared with PBFT, we have implemented an efficient Byzantine fault-tolerant algorithm, and combined with Tendermint to achieve the effective robustness of the protocol. In the master node election algorithm, according to the longest chain election rule, the master node is replaced at the beginning of each round to reduce the possibility of the Byzantine leader node doing evil. And to simplify the communication structure of the PBFT core commit phase to reduce the communication complexity to $O(N)$. Combining the empty block mechanism of Tendermint's locking mechanism to merge the view-change process of PBFT into the normal consensus, effectively improving the robustness of the protocol. Finally, the performance of the protocol has not been greatly improved. In the future, the work content of the protocol will be further optimized for better performance.

ACKNOWLEDGMENT

The authors of this paper were supported by NSFC through grants U20B2053, U19B2023, U19B2036, and Key Research and Development Project of Hebei Province through grant 20310101D.

REFERENCES

[1] drdobbs. The Byzantine Generals Problem[J]. Acm Transactions on Programming Languages & Systems, 1982, 4(3):382-401.
[2] Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance.In Proceedings of the Third Symposium on Operating Systems Design and Implementation, OSDI '99, pages 173–186, Berkeley, CA, USA,1999. USENIX Association.
[3] Ethan Buchman. Tendermint: Byzantine fault tolerance in the age of blockchains. PhD thesis, 2016.
[4] Miller A , Xia Y , Croman K , et al. The Honey Badger of BFT Protocols[C]// Acm Sigsac Conference on Computer & Communications Security. ACM, 2016:31-42.
[5] Cynthia, Dwork, Nancy, et al. Consensus in the presence of partial synchrony[J]. Journal of the Acm, 1988.
[6] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. J. ACM, 32(2):374–382, 1985.
[7] Distler T , Rüdiger Kapitza. Increasing performance in byzantine fault-tolerant systems with on-demand replica consistency[C]// Conference on Computer Systems. ACM, 2011.
[8] M. Burrows. The Chubby lock service for loosely coupled distributed systems. In Proceedings of the 7th Symposium on Operating Systems Design and Implementation, pages 335–350, 2006.
[9] M. Correia, N. F. Neves, and P. Ver´ıssimo. How to tolerate half less one Byzantine nodes in practical distributed systems. In Proceedings of the 23rd International Symposium on Reliable Distributed Systems, pages 174–183, 2004.
[10] J. Cowling, D. Myers, B. Liskov, R. Rodrigues,and L. Shrira. HQ replication: A hybrid quorum protocol for Byzantine fault tolerance. In Proceedings of the 7th Symposium on Operating Systems Design and Implementation, pages 177– 190, 2006.
[11] R. Guerraoui, N. Kne˘zevi´c, V. Qu´ema, and M. Vukoli´c. The next 700 BFT protocols. In Proceedings of the EuroSys 2010 Conference, pages 363–376, 2010.
[12] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: speculative Byzantine fault tolerance. In Proceedings of the 21st Symposium on Operating Systems Principles,pages 45–58, 2007.
[13] R. Rodrigues, M. Castro, and B. Liskov. BASE: Using abstraction to improve fault tolerance. In Proceedings of the 18th Symposium on Operating Systems Principles, pages 15–28, 2001.
[14] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Computing Survey, 22(4):299–319, 1990.
[15] S. Sen, W. Lloyd, and M. J. Freedman. Prophecy: Using history for high-throughput fault tolerance. In Proceedings of the 7th Symposium on Networked Systems Design and Implementation, 2010.
[16] Yee Jiun Song and Robbert van Renesse. Bosco: One-step byzantine asynchronous consensus. In Distributed Computing, 22nd International Symposium, DISC 2008, Arcachon, France, September 22-24, 2008. Proceedings, pages 438–450, 2008.
[17] Victor Shoup. Practical threshold signatures. In Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding, pages 207–220, 2000.
[18] HariGovind V. Ramasamy and Christian Cachin. Parsimonious asynchronous byzantine-fault-tolerant atomic broadcast. In Principles of Distributed Systems, 9th International Conference, OPODIS 2005, Pisa, Italy, December 12-14, 2005, Revised Selected Papers, pages 88–102, 2005.
[19] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. J. ACM, 27(2):228–234, 1980.
[20] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II, pages 3–33, 2018.
[21] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II, pages 643–673, 2017.
[22] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. https://bitcoin.org/bitcoin.pdf, 2008.
[23] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. J. Comput.Syst. Sci., 75(2):91–112, 2009.
[24] Guy Golan-Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael K. Reiter, Dragos- Adrian Seredinschi, Orr Tamir, and Alin Tomescu. SBFT: a scalable decentralized trust infrastructure for blockchains. CoRR, abs/1804.01626, 2018.
[25] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017, pages 51–68, 2017.
[26] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. ACM Trans. Comput. Syst., 20(4):398–461, 2002.
[27] Y. Gilad et al. "Algorand: Scaling Byzantine Agreements for Cryptocurrencies", 2017.
[28] Dou, Yingtong, et al. "Enhancing graph neural network-based fraud detectors against camouflaged fraudsters." Proceedings of the 29th ACM International Conference on Information & Knowledge Management. 2020.