Journal of Cyber Security
DOI:10.32604/jcs.2020.011341
Article

Excellent Practical Byzantine Fault Tolerance

Huanrong Tang, Yaojing Sun and Jianquan Ouyang*

School of Cyberspace Security, Xiangtan University, Xiangtan, 411105, China
*Corresponding Author: Jianquan Ouyang. Email: oyjq@xtu.edu.cn
Received: 30 August 2020; Accepted: 19 October 2020

Abstract: With the rapid development of blockchain technology, more and more people are paying attention to the consensus mechanism of blockchain. Practical Byzantine Fault Tolerance (PBFT), as the first efficient consensus algorithm solving the Byzantine Generals Problem, plays an important role. But PBFT also has its problems. First, it runs in a completely closed environment, and any node can't join or exit without rebooting the system. Second, the communication complexity in the network is as high as O(n2), which makes the algorithm only applicable to small-scale networks. For these problems, this paper proposes an Optimized consensus algorithm, Excellent Practical Byzantine Fault Tolerance (EPBFT), in which nodes can dynamically participate in the network by combining a view change protocol with a node's add or quit request. Besides, in each round of consensus, the algorithm will randomly select a coordination node. Through the cooperation of the primary and the coordination node, we reduce the network communication complexity to O(n). Besides, we have added a reputation credit mechanism and a wrong node removal protocol to the algorithm for clearing the faulty nodes in time and improving the robustness of the system. Finally, we design experiments to compare the performance of the PBFT and EPBFT algorithms. Through experimental, we found that compared with the PBFT algorithm, the EPBFT algorithm has a lower delay, communication complexity, better scalability, and more practical.

Keywords: Byzantine fault tolerance; distributed consensus; PBFT; blockchain; PBFT optimization

1 Introduction

In 2008, an article called “Bitcoin: A Peer-to-Peer Electronic Cash System” introduced Bitcoin into people’s field of vision, and has since entered the era of digital currency. The Bitcoin system subsequently developed rapidly, attracting the attention and research of a large number of scholars. Later, the researchers extracted the underlying framework of the Bitcoin system and named it blockchain.

Simply, the blockchain is a decentralized distributed database, and the data in the system needs to be jointly maintained by all nodes. Each node in the blockchain system is equal, and has a copy of the database. Blockchain, as its name, is formed by connecting blocks sequentially, as shown in Fig. 1. Block is the basic data unit of blockchain. It is a special data structure consisting of a block header containing metadata and a block body containing transactions. The structure of the blocks in different blockchain systems is similar, but the content contained in the block header and block body is different. In the Bitcoin system, the block header size is fixed at 80 bytes, and each block body contains at least 500 transactions. The block structure is shown in Tab. 1.

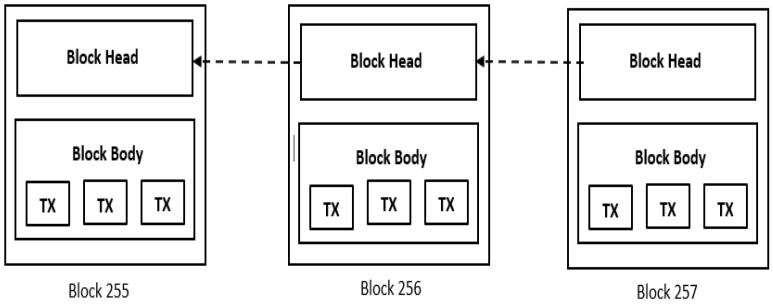

Figure 1: The structure of the blockchain

Table 1: The struct of the node information table

Field	Size	Description
Block Size	4 bytes	The size of block, in byte
Block header	80 bytes	Several fields form the block header
Transaction Counter	1–9 bytes	How many transactions follow
Transaction	Variable	The transaction recorded in this block

The development of the blockchain can be divided into three stages: (1) The use of blockchain technology to release digital currency and realize the point-to-point transmission of value. At this time, it is called Blockchain 1.0, which is a decentralized, everyone-oriented virtual currency system and related applications. Bitcoin is the representative of it. However, the blockchain system can only be applied to the transaction payment field. when applied to other fields, it is necessary to re-adjustment of the underlying structure. (2) Use the blockchain to build an application development platform, so that people can develop their own blockchain applications based on this platform without rebuilding the underlying blockchain system. This is the Blockchain 2.0, use smart contracts to develop pan-finance programmable applications. The representative is Ethereum. However, the consensus mechanism of this type of application has high latency and low throughput and cannot be applied to other fields besides finance. (3) Blockchain 3.0 symbolizes that the blockchain has entered the field of social justice and intelligence. We can use blockchain to confirm, measure and store the property rights of any information that represents value, such as electronic notarization, arbitration, voting, auditing, medical treatment, traceability, etc. The difficulty of blockchain 3.0 is the performance problem, which requires a throughput of up to one million.

With the rapid popularity of cryptographic currency [1], blockchain technology has also been rapidly developed and widely used in various fields, such as finance, medical care [2–3], education, IOT [4], edge computing [5], and so on. The core of blockchain technology, distributed consensus mechanism, has also become a hot topic of research [6].

The distributed consensus mechanism is to solve the Problem of Byzantine failures [7]: When the channel is reliable, how to make the whole system run well and ensure the integrity, reliability, and consistency of stored information with the interference of malicious nodes. Research-based on Byzantine Generals Problem has been going on for a long time. However, most earlier work either concerns techniques designed to demonstrate theoretical feasibility that is too inefficient to be used in practice or assumes synchrony [8–10], i.e., relying on known bounds on message delays and process speeds. Practical Byzantine Fault Tolerance (PBFT) was the first practical algorithm to tolerate Byzantine errors, presented by Castro et al. [11]. PBFT is a state machine replication protocol [12]. It provides liveness and safety properties if the number of malicious nodes in the network does not exceed [(n – 1) / 3], where n is the number of the total replicas of the network. But PBFT also has some problems: (1) The algorithm runs in a completely closed environment, and nodes are not allowed to join or exit freely. (2) It is closed leads to the inability to kick malicious nodes out of the network in the operation of the algorithm, which leads

to repetitive errors. (3) The communication complexity of the algorithm is as high as O(n2). To solve these problems, we propose a better performance Byzantine fault-tolerant algorithm called Excellent Practical Byzantine Fault Tolerance (EPBFT).

EPBFT is a PBFT-based consensus protocol that inherits all of the advantages of PBFT, providing the same safety and liveness as PBFT. At the same time, EPBFT works on the weak synchronization assumption, which makes it applicable to the Internet. For the issues mentioned above, EPBFT has also made some optimizations: it reduces the communication complexity from O(n2) to O(n) and removes the commit phase to reduce the delay. EPBFT has better scalability, allowing nodes to join or exit the network while the system is running. Besides, the credit score mechanism and the error node clearing protocol have been introduced to EPBFT to improve the robustness of the system. The original PBFT protocol uses the C/S request-response mode. that is essentially a manifestation of a centralized mindset. When applied to a blockchain system, it does not match the peer-to-peer network of the blockchain. So in EPBFT, we changed it to the P2P network topology response. Compared with PBFT, EPBFT has better communication performance, lower latency, more stability, and more practical.

Thus, this paper makes the following contributions: (1) Better scalability. When a node joins or exits, the node in the network does not need to be down. They only need to perform consensus verification on the node's join or exit. After passing the verification, they can join or exit the system. (2) Lower communication complexity. In each round of consensus, the algorithm reduces the communication complexity to O(n) by randomly selecting a coordinating node to cooperate with other nodes. (3) High system stability. The algorithm introduces a reputation credit mechanism and a faulty node removal protocol, which cleans up faulty nodes in the system promptly. (4) change the C/S request-response mode in the original PBFT to P2P network topology response mode.

There is some closest work to us. Liu et al. proposed a dynamic authorization PBFT algorithm [13]. It allows the nodes to join the network while the system is running by classifying nodes, but the new nodes do not participate in the consensus immediately. The number of consensus nodes remains unchanged during the consensus process, and the communication complexity is O(n2). Xu Xiao proposed a dynamic BFT [14]. It realizes the dynamic participation of nodes with the help of a reliable central node NodeCA and view-change protocol. But the process of the node joining process is assumed to be too ideal. It requires the new node to send the join request to the network immediately after the network registration is successful. Its communication complexity is also O(n2). Jiang Yanjun proposed an HSBFT algorithm [15], which also realized the active participation of nodes by introducing trusted service providers. Although it reduces the communication complexity to O(n), it requires five stages to reach a consensus. Thus, it increases the delay of the message.

The rest of the paper organized as follows: In Chapter 2, we will give a brief introduction to the system model and some concepts of EPBFT; The EPBFT consensus process, and some of the problems that may arise, are discussed in Chapter 3; In Chapter 4, we will test and analyze EPBFT from latency, communication complexity, and Fault tolerance. Finally, we conclude in Chapter 5.

2 Preliminaries

In this chapter, we will introduce the system model and some important concepts of EPBFT.

2.1 System Model

The system assumption of EPBFT is very similar to PBFT. We assume that there are some malicious nodes in an asynchronous distributed system. The behavior of malicious nodes is arbitrary. They may send error messages, do not reply, delay the communication of the correct nodes, even join forces to attack. But they cannot delay the communication of the correct nodes indefinitely and subvert the cryptographic techniques.

In the system, we use encryption to prevent spoofing, replay attacks and detect message corruption [16]. The encryption technologies used in the system have public-key signatures, message authentication

codes, and message digests produced by collision-resistant hash functions. We indicate the signature of the message m signed by node i is $<m>\sigma_i$, and the digest of the message m is D (m). It should note that we signed the digest of a message and then attached it to the end of plaintext message instead of directly signing the original text ($<m>\sigma_i$, should be explained in this way). Each node is required to know the public key of other nodes.

2.2 Normal-Case Operation of PBFT

Under normal circumstances, the PBFT needs three stages to finish consensus: pre-prepare, prepare, commit. The pre-prepare phases and prepare phases ensure that non-faulty nodes agree on the order of the requests in the same view. The prepare phases and commit phases provide that non-faulty nodes can still reach a consensus on the order of the requests even if the view has changed. Through these three phases, all non-faulty nodes guarantee to perform the same sequence of requests, and the state of the State Machine Replication is consistent.

Fig. 2 shows the operation of the algorithm in the normal case of no primary faults. Replica 0 is the primary, replica 3 is faulty, and C is the client. Dashed lines indicate possible situations.

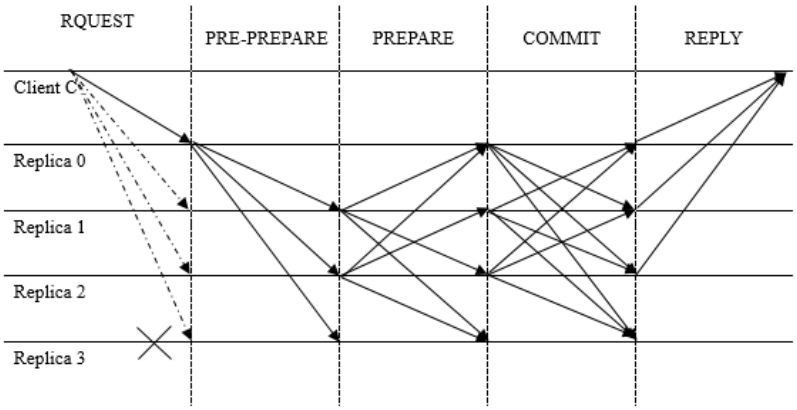

Figure 2: Normal case operation

2.3 Node Information Table (NIT)

In EPBFT, the state and number of nodes are dynamically changing, so each node needs to maintain an NIT to get the information in the network in real-time.

Table 2: The struct of the node information table

Node ID	State	IP	PK	Grade
1	Active	192.169.3.55	…	3
2	Sleep	192.168.66.201	…	2
…	Dead	…	…	0

where PK is similar to “1Ez69SnzzmePmZX3WpEzMKTrcBF2gpNQ55”.

The NIT records the information of each node in the network. The state represents the current state of a node. It has three states: Active, Sleep, Dead. Dead indicates that the node is faulty. Sleep indicates that the node has actively logged out of the network. Active indicates that the node is normal. The Grade indicates [17–18] the times of VERIFY-TIMEOUT of the node when the node is selected as the coordinating node, the initial value is still 3. When it reduced to 0, the node can also be considered as the

faulty node. Besides, the table also includes other information about the node, such as IP, PK. By querying the NIT, a node can obtain the number n of active nodes in the network [19].

2.4 Primary Selection and View Number

In EPBFT, the number of nodes is dynamically changing. So the formula, P = V mod Rn, for calculating the primary, is no longer applicable, where Rn is the total nodes. The new way to select the primary is as follows:

When the view changes, starting from the row where the current primary is located in the NIT, looking for a node, whose state is Active, Grade is greater than 0, as the new primary. If the end of NIT is reached, the condition is still not met, then search again from the first line of NIT.

The view number is also redefined. We use the 'n-nodeId' to represent a view, where n represents the view is the n-th view, which increments from 0; nodeId represents the node number of the new primary node.

2.5 Network Structure

PBFT uses a C/S request-response mode. The client sends a request to the distributed system. After receiving the request, the distributed system sequentially executes the request and returns the result to the client. That is essentially a manifestation of a centralized idea that does not correspond to the decentralized features of blockchain [20]. So in EPBFT, we changed the C/S request-response mode of PBFT to the P2P network topology response mode. We randomly selected a coordination node in the process of consensus. When most nodes approve a new block, the coordination node sends a CONFIRMED message to all active nodes. After receiving the CONFIRMED message, the primary starts to generate the next block. This way can dynamically generate blocks according to the state of the network, which increases the flexibility of the system.

3 Excellent PBFT

In this chapter, we will introduce the algorithm of EPBFT in detail.

Although PBFT has many advantages, its closure and high communication complexity have being criticized.

Therefore, in EPBFT, we propose an Optimized consensus algorithm. We will randomly select a node as the temporary coordination node during each round of consensus. Through the coordination node, we can not only change the C/S request-response model of the PBFT to the P2P network topology response mode but also reduce the communication complexity to O(n). We use backup nodes to represent all non-primary nodes.

3.1 Normal Case Operation

When a node is selected as the primary node or the primary node receives a CONFIRMED message from the coordination node, it takes the transactions from the transaction pool and packs them into a block and then starts a three-phases protocol to finish the consensus. The three stages are proposal, verify, confirmed.

In the proposal phase, the primary node calculates the height of the next block and sends a proposal message with the block message piggybacked to all other active nodes in its NIT. Then the message is added into its log and a timer Tpc start (waiting for a confirmed message).

The format of the proposal message is <<PROPOSAL,v,h,d,t,R>σ_p,block>, where v is the current view, h is the height of the next block, and d is the digest of the block, d = D(block), t is the timestamp when the proposal message was sent, σ_p is the signature of the primary on PROPOSAL, R is the location of the coordination node, It finds the R-th active node from the row where the primary node is in the NIT. R calculate [21] as follow:

$$R = \text{StrToInt}\left(\text{SubStringEnd32}\left(\text{hash}(\text{block})\right)\right)\ mod\ n \quad (1)$$

Where n is the number of active nodes in NIT.

A backup accepts a proposal message provided:

(1) d is the digest of the block, σ_p is correct, and node R is active;

(2) The v in the proposal message is equal to the local v.

(3) it has not accepted a proposal message with same v and h but containing a different d;

(4) h is in the range of [L, H], and L is the last stable checkpoint, H = L + 2 * K, K is a constant;

(5) The transactions in the block are correct;

If backup node i accepts a proposal message, it enters the verify phase and sends a verification message to the node R. Then it starts a timer Tc (waits for a confirmed message) [22], and adds the proposal and verify messages to its log. If the verification fails, nothing will be done. The form of the verify message is: <<VERIFY,v,h,d,P,i>σ_i,block> where the P is the proposal message node i received, i is the node Id.

After receiving a verify message, the node R accepts the verify message and adds it to its log provided the signature of the verify message is correct, the v in message equals the replica's v, the h is between L and H. When the node R receives 2f verify messages from different active nodes with the same v, h, d, it enters the confirmed phase. Then it broadcasts a confirmed message to all active nodes, including the primary node.Theconfirmed message's format is: <<CONFIRMED,v,h,d,t,ψ>σ_r,block>, where ψ is a set containing the 2f correct verify messages received by node R, t is the timestamp in proposal message.

Fig. 3 shows the process of the algorithm in the normal case of no primary faults. Replica 0 is the primary, replica 3 is faulty, and replica 2 is the coordination node of this consensus.

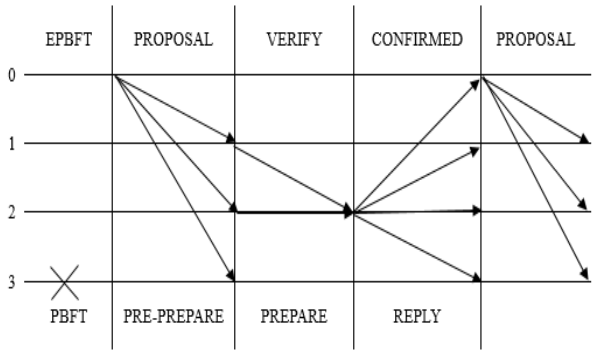

Figure 3: Normal case operation

3.2 Other Case

Under normal circumstances, EPBFT reduces the communication complexity to O(n) through a coordination node. However, in an asynchronous network, it is difficult to avoid the existence of network delay, or the primary and coordination node does evil, which may cause the above process not to be executed correctly. Next, let's discuss the possible anomalies and their solutions.

Case A

If the backup node does not receive a proposal message from the primary within the specified timer Tp, the backup node starts the view-change protocol and switches to the new view.

Case B

If the coordinating node accepts f + 2 (f = (n – 1)/3, n is the total nodes) verify messages from different backup nodes, it will broadcast <VERIFY-FAIL,v,h,t,ψ>σ_r to other nodes, where ψ is the set of f + 2 different verify messages it accepts. When a backup node receives the verify-fail message, it verifies the

signature, and each of the messages in ψ. When they passed the verification, the backup starts the view-change protocol to switch a new view.

Case C

If the backup node does not receive a confirmed message from node R before Tc expired. They resend the verify message to node R + 1 and decrement node R's grace by one. (1) If the backup receives the verify message from node R + 1, it indicates that there is a problem with the coordination node, they will decrement node R' time-out by one. And then continue. (2) If the backup still does not receive the verify messages from node R + 1, it indicates that the primary node has failed, they will start the view-change protocol to change the primary.

Case D

If the primary does not receive the conformed message within Tpc (Tpc > 2Tc), It randomly selects k nodes, and requests conformed messages from them. If most of the responses (with the threshold percentage p) from the k backup nodes are the same. The primary node accepts the conformed message. The reason for this is to prevent that the coordinating node deliberately does not send a message to the primary node to prevent the system from running.

If there are n active nodes in the current system with f faulty nodes, the probability Pr the primary get the correct CONFORMED message is:

$$Pr = \frac{\sum_{i=kp}^{k} C_{n-f}^{i} C_{f}^{k-i}}{C_{n}^{k}} \tag{2}$$

For quantitative analysis, we assume f = 4, n = 13, and the results show in the Tab. 3.

Table 3: Pr with different k and p

k \ P / Pr	$\geq \frac{1}{2}$	$\geq \frac{2}{3}$
1	0.69	0.69
2	0.92	0.46
3	0.79	0.79
4	0.94	0.64
5	0.88	0.48
6	0.97	0.78
7	0.95	0.65
8	1	0.51
9	1	0.82

It can be seen from the table that Pr does not increase linearly as k increases. Even with the increase in p, Pr may decrease. In the example of n = 13, f = 4, a better solution is to select six backup nodes for inquiry. When more than three identical confirmed messages were received, the primary has a 97% probability of getting the correct confirmed message.

3.3 The Scalability of the Algorithm

In the PBFT algorithm, nodes are not allowed to join or exit during system operation, which severely limits its application in a real scene. In this section, we introduce the scalability of the EPBFT.

3.3.1 View Change Protocol

To ensure the liveness of the system, when the backup node suspect that the primary is evil or does not receive the proposal message from the primary within the specified time Tc, they will start the view-change protocol to switch to a new view. In the EPBFT algorithm, the view-change protocol is as follows:

When a node begins the view change protocol for a new view, it will stop accepting other messages (except checkpoint message, view-change message, new-view message), and then it sends a view-change message to the primary of the new view v+1. The format of the view-change message is: <view-change, $v+1,h_{last},C,S,I>\sigma_i$, where v+1 is the new view, hlast is the height of the previous block, C is a set, containing the proposal message and confirmed message of block hlast, S is the NIT of node i, which has to delete the old primary node.

When the new primary receives 2f valid view-change messages for view v+1 from different replicas with the same S, it enters view v+1, multicasts a new-view message to all other active backups and set the local NIT to S. The format of the new-view message is: $<$new-view,$v+1,h_{next},C,V,S>\sigma_p$ where V is a set that contains 2f vailed view-change messages received by the primary, and h_{next} is the most h_{last} in view-change messages in V; C is a set, containing the proposal and confirmed message of $h_{next.}$

A backup accepts a new-view message for v+1 if it is signed properly, if the view-change messages in V are valid, if $h_{next,}$ C, S is matched to V. And then the backup enters view v+1.add the new-view message to its log and sets its NIT to S.

After that, the new primary starts generating blocks from the height h_{next}.

Fig. 4 shows the process of the view-change proposal. Replica 3 is the old primary and is faulty, and replica 2 is the new primary node. The dotted line indicates that the node does not exist.

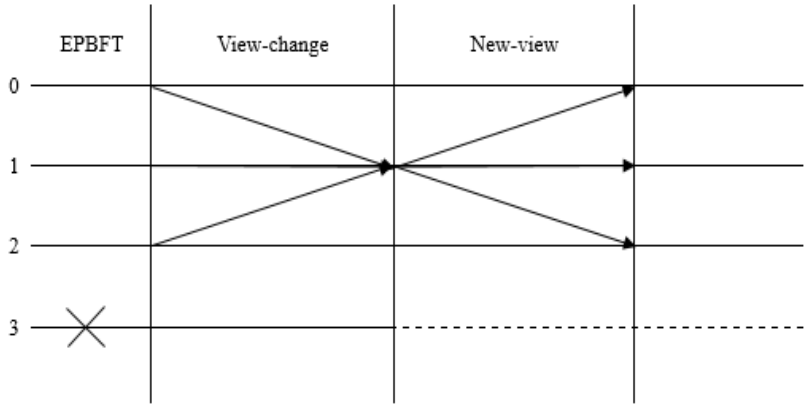

Figure 4: The process of View-Change Proposal

3.3.2 New Node Participation Protocol

When a new node j, wants to join the network, it multicasts a join-req message to all nodes in the network. The format of the join-req message is: $<$JOIN-REQ,M$>\sigma_j$, where M is the basic information of node j, including the IP address, public key.

After receiving the join-req message, the node i checks the M information. If the verification is passed, node i assigns a node number m to node j according to its own NIT. The allocation rules for m are as follows:

(1) If node j is not in NIT, set m to NIT's length plus one;

(2) If the node j is in the NIT, but the status is Dead. the join will be rejected;

(3) If the node j is in the NIT and the state is sleep, then m is set to the original node number of node j.

Then the node i sends a join-rep message to the node j. The format of the join-rep message is: $<$JOIN-REP,v,t,m,S,n,i$>\sigma_j$, where S is the NIT of node i, n is the number of active nodes in S, m is the node number assigned by node i for j, t is the timestamp.

The node j adds the join-rep message to its log if it is signed properly, m, n is matched to S. Node j selects the join-rep message whose n is the largest and has 2 * ((n – 1)/3 + 1) messages as same as it (with the same v, t, S)in the log, as the correct join-rep message. Because node j does not know how many

nodes are in the current network. From the right join-rep messages, node j can obtain the total number of nodes n in the current network, the state S of each node, and the maximum number of faulty nodes (f = ((n – 1))/3). Then, it multicasts a join message to all active nodes in S, and set its NIT to S.

The format of the join message is: <JOIN,v,t,m,V>σ_j, where v is the view, t is the timestamp in the join-rep message, V is a set containing 2f + 1 correct join-rep messages received by the node j.

After receiving the join message, backup i verifies the signature and each message in V, check v, t, m according to the join-rep sent before. If the verification is passed, it needs to add node j to its NIT with the help of a view change protocol. Backup i stop accepting other messages (except checkpoint message, view-change message, new-view message), and then it sends a view-change message to the primary.

The format of the view-change message is: <view-change,v,h_{last},C,J,i>σ_i, where v is the view, h_{last} is the height of the previous block, C is a set, containing the proposal message and confirmed message of block h_{last}, J is the join message from node j.

When the primary receives 2f valid view-change messages from different replicas with the same v, J, it multicasts a new-view message to all other active backups (include the node j) and adds node j to its NIT.

The format of the new-view message is: <new-view,v,h_{next},C,V>σp, where V is a set that contains 2f vailed view-change messages received by the primary, and h_{next} is the smallest h_{last} in view-change messages, in which each of them has another f identical messages as same as it, in V, C is a set, containing the proposal and confirmed message of h_{next}.

A backup accepts a new-view message if it is signed properly, if the view-change messages in V are valid and the J in each the view-change message of V is the same, if h_{next}, C is matched to V. And then the backup adds the new-view message to its log and adds node j to its NIT.

Fig. 5 is the joining process of node j. Node 0 is the primary node, node 3 is faulty, and j is the new node.

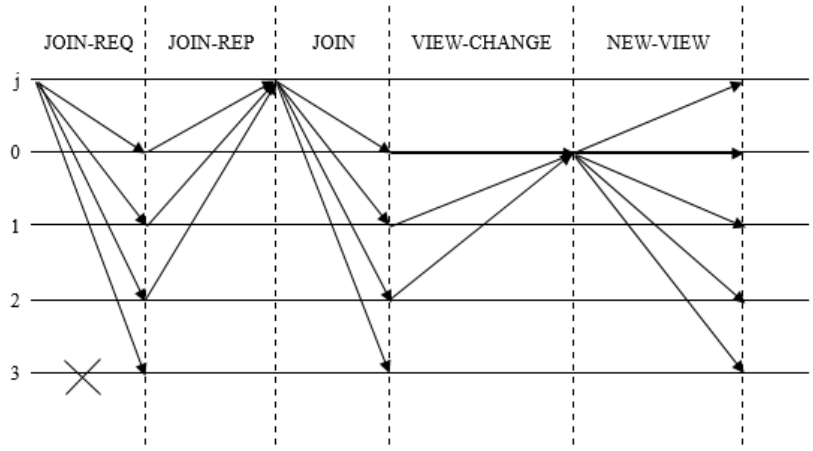

Figure 5: The process of a new node participating

3.3.3 Active Exit Protocol

In EPBFT, when a node e actively requests to log out of the network, the following process is performed.

First, node e multicasts an exit-req message to all other active nodes. The format of the exit-req message is: <EXIT-REQ,v,t,n>σ_e, where v is the view, n is the node number of node e, t is the timestamp and is totally ordered.

When node i receives an exit-req message, it verifies its signature and checks v, t according to its current v. If the verification passes, node i will send an exit-rep message to node e. The format of the exit-rep message is: <EXIT-REP,v,t,n,i>σ_i, where v is the view, n is the node id of node e, t is the timestamp in the exit-req message.

The node e adds the exit-rep message to its log if it is appropriately signed, v, t, n is the same with its

exit-req message. When node e receives 2f correct exit-rep messages from the different backup node with the same v, t, n, it multicasts an exit message to all other active nodes, The format of the exit message is: <EXIT,v,t,n,V>σ_e, where V is a set containing the 2f valid exit-rep messages received by the node e.

After receiving the exit message, backup i verifies the signature and each message in V, check v, t, m according to the exit-req received before. If the verification is passed, it needs to remove node j from its NIT utilizing the view change protocol. Backup i stop accepting other messages (except checkpoint message, view-change message, new-view message), and then it sends a view-change message to the primary. The format of the view-change message is: <view-change,v,h_{last},C,E,i>σ_i, where v is the view, h_{last} is the height of the previous block, C is a set, containing the proposal message and confirmed message of block h_{last}, E is the exit message from node e.

When the primary receives 2f valid view-change messages from different replicas with the same v, E, it multicasts a new-view message to all other active backups (include the node e) and remove the node e from its NIT. The format of the new-view message is: <new-view,v,h_{next},C,V>σ_p, where V is a set that contains 2f vailed view-change messages received by the primary, and h_{next} is the most in view-change messages, V,C is a set, containing the proposal and confirmed message of h_{next}.

A backup accepts a new-view message if it is signed properly if the view-change messages in V are valid and the E in each the view-change message of V is the same if h_{next} C is matched to V. And then the backup adds the new-view message to its log and remove the node j from its NIT.

Fig. 6 shows the active exit protocol in EPBFT. Node 0 is the primary, node 3 is faulty, and node 1 is the exit node e. The dotted line indicates that the node does not exist.

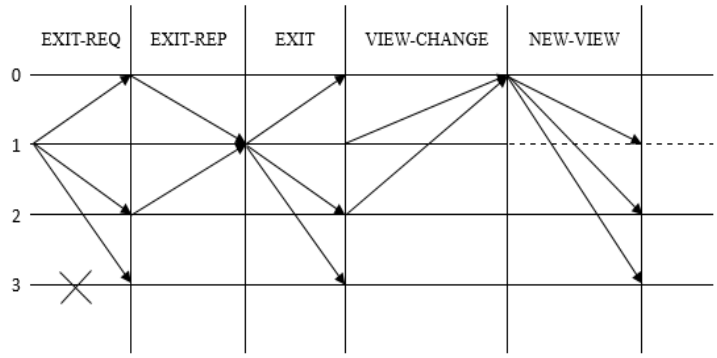

Figure 6: The flow of Active Exit Protocol

3.3.4 Node Recovery Protocol

When a node r that has actively quit applies to join the network again, it performs a node recovery protocol.

Firstly, it multicasts a recover-req message to all nodes in the network. The format of the recover-req message is: <RECOVER-REQ,r,ψ>σ_r, where r is the original id of the node r, ψ is the information of node r, such as IP, PK.

After receiving the recover-req message, node i verifies the signature and checks the n and ψ according to its NIT. If the verification is passed, it sends a recover-rep message to node r. Otherwise, it does nothing. The format of the recover-rep message is: <RECOVER-REP,v,t,r,S,n,i>σ_i, where S is the NIT of node i, n is the number of active nodes in S, m is the original node id of the node r, t is the timestamp.

The node r adds the recover-rep message to its log if it is signed properly, n is matched to S. m is equal to the recover-req message. Node r selects the recover-rep message whose n is the largest and has 2 * ((n – 1) / 3 + 1) messages as same as it (with the same v, t, S) in the log, as the correct recover-req message. Because node r does not know how many nodes are in the current network. From the correct recover-req messages, node r can obtain the total number of nodes n in the current network, the state S of

each node, and the maximum number of faulty nodes (f = (n – 1) / 3). Then, it multicasts a recover message to all active nodes in S, and set its NIT to S.

The format of the recover message is: <RECOVER,v,t,m,V>σ_j, where v is the view, t is the timestamp in the recover-rep message, V is a set containing 2f+1 correct recover -rep messages received by the node r.

After receiving the recover message, backup i verifies the signature and each message in V, check v, t, m according to the recover-rep sent before. If the verification is passed, it needs to change the state of node r in its NIT to ACTIVE. Backup i stop accepting other messages (except checkpoint message, view-change message, new-view message), and then it sends a view-change message to the primary.

The next process is the same as the new node joining process and will not be described in detail here.

Fig. 7 shows the recovery process of node r. Node 0 is the primary node, node 3 is faulty, and node r is the node to be recovered.

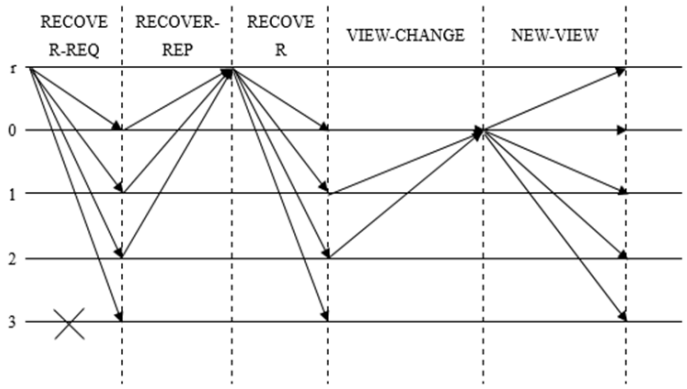

Figure 7: The process of a new node recovering

3.4 Reputation Credit Mechanism and Wrong Node Removal Protocol

To improve the robustness and stability of the system, we should appropriately remove the nodes which are marked as evil. For this, we introduce two new concepts:

Grade: It is the number of times the node has failed. When a coordinating node does not send the confirmed message to other backup nodes in time, its Grade decreases by 1. When Grade drops to 0, the node is considered to be the fault node and should be cleaned up.

Error node removal protocol: When a node finds that the grade of a node f in its NIT becomes 0, it will add the id of the node to the pre-clearing set E, and periodically check whether E is empty. If it is not empty, the view change protocol will start, and these nodes will be cleared from the NIT.

In this chapter, we present a series of protocols for nodes to participate and exit the EPBFT network dynamically. The joining or exiting process of a node does not affect the safety and liveness of the algorithm. Because in each sub-process, we all ensure that at least f+1 honest nodes joined.

3.5 Correctness

In this section, we present an Excellent PBFT that reduces communication complexity and changes the C/S response mode. But in the process of consensus, the node can only enter the next stage with the consent of most nodes. So it still maintains the same safety and activity as PBFT.

To avoid duplication, the backup nodes will retain the last confirmed messages they received and reject the messages whose timestamps less than those.

4 Performance Evaluation

In this chapter, we will test and analyze EPBFT from the four aspects of communication complexity,

delay, fault tolerance, and scalability, and compare it with the original PBFT algorithm to verify the timeliness, robustness, and availability of the improved algorithm.

4.1 Scalability

In Chapter 4, we introduced in detail how the EPBFT consensus algorithm performs node join, exit, and recovery. In this section, we will test these features to ensure their usability and correctness.

When a new node joins (or recovers), the original consensus nodes do not need to stop, and they only need to perform consensus verification on the new node's join (or recovery) request. After the verification is passed, it can be added (or recovered) to the subsequent consensus process of the entire network. In this experiment, we assume that there were originally 7 consensus nodes in the system, and the new node was added to the network as the 8th node.

The test results of consensus node join (or recovery) are shown in Tab. 4. Each experiment was repeated 10 times.

Table 4: The result of adding (or recovering) consensus node

Experiment ID	Experimental scene	Expected results	Actual results
1	The new (or recovery) node is not in the NIT of the consensus node	Node addition (or recovery) success (failure)	As expected
2	The new (recovery) node is in the NIT of the consensus node, and the status is ACTIVE	Node addition (or recovery) failed	As expected
3	The new (recovery) node is in the NIT of the consensus node, and the status is SLEEP	Node addition (or recovery) failed (success)	As expected
4	The new (recovery) node is in the NIT of the consensus node, and the status is DEAD	Node addition (or recovery) failed	As expected

When a member node wants to exit the system, other nodes need to perform consensus verification on the exit request of the existing node. After the verification is passed, all nodes change the consensus parameters and no longer send consensus data to the node in the subsequent consensus process. The node can then disconnect from the network and exit the common consensus. In this experiment, we assume that there were originally 8 consensus nodes, and one node applied to withdraw from the blockchain network. It is worth noting that when the number of nodes is less than 5, nodes are no longer allowed to exit the network.

The test results of consensus node exit are shown in Tab. 5. Each experiment was repeated 10 times, too.

Table 5: The result of consensus node exit

Experiment ID	Experimental scene	Expected results	Actual results
1	The exited node is in the NIT of the consensus nodes, and the status is ACTIVE	Node exited successfully	Node exited successfully
2	The exited node is in the NIT of the consensus nodes, and the status is SLEEP	Node exit failed	Node exit failed
3	The exited node is in the NIT of the consensus nodes, and the status is DEAD	Node exit failed	Node exit failed
4	The exited node is not in the NIT of the consensus nodes	Node exit failed	Node exit failed
5	There are only 4 consensus nodes in the system	Node exit failed	Node exit failed

It can be proved through experiments that the EPBFT consensus algorithm implements the functions of node joining, exiting, and recovering. Compared with the original PBFT algorithm, the improved algorithm has better scalability.

4.2 Performance Analysis

Indicators for evaluating the pros and cons of blockchain applications include throughput, latency, fault tolerance, and traffic. Experiments and statistics on the EPBFT algorithm for these indicators, and compares with the classic PBFT algorithm. The performance and advantages of the EPBFT algorithm are proved experimentally.

4.2.1 Latency

Delay refers to the time interval from the transaction submission to the network until the transaction is determined to take effect. It is a measure of network performance and the runtime of consensus algorithms. In a non-forked blockchain system, transactions are added to the chain to indicate that the transaction is confirmed. In a forked system, it is necessary to wait for no more forks before the transaction is confirmed. The lower the delay, the better the system performance. The delay is calculated as:

$$T_{delay} = T_{tx\ broadcast} + T_{consensus} + T_{block\ broadcast} \quad (3)$$

Ttx broadcast indicates the time from when the transaction is generated until the consensus node accepts the transaction; Tconsensus indicates the consensus time of the transaction; Tblock broadcast indicates the consensus block broadcast time. Neither the PBFT algorithm nor the EPBFT algorithm is forked, so in our system, as long as the transaction is added to the chain, it means that the transaction is determined.

We set a total of 7 nodes in the current system, of which 2 are evil nodes. We take 5, 15, 20, 40, 100 blocks in the current system to propagate and perform 10 experiments on each value, and then use the average value as the transaction delay for each value. And then compare the delay of EPBFT and PBFT is shown in Fig. 8.

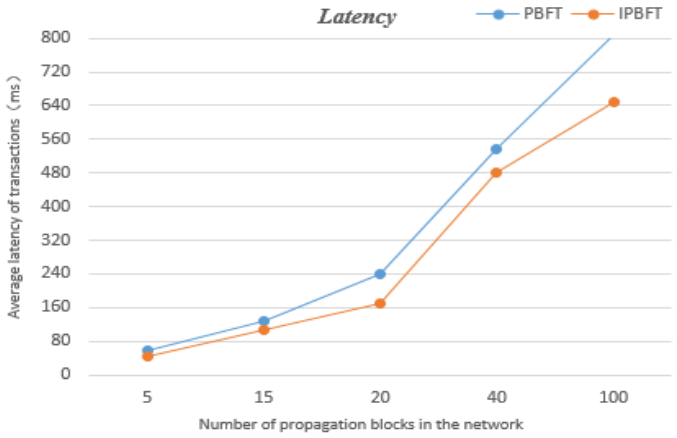

Figure 8: The comparison of PBFT's delay and EPBFT's delay

4.2.2 Fault Tolerance

Fault tolerance indicates the maximum number of error nodes that the system can tolerate. The more the total number of nodes in the system, the more error nodes it can tolerate, but when the number of error nodes exceeds the maximum value that the system can tolerate, consensus cannot be reached. The ability of EPBFT algorithm to tolerate error nodes is the same as that of PBFT algorithm: f = (n–1)/3. We set a total of 7 nodes in the current system and let the number of error nodes are 0,1,2,3,4 to observe the performance of the system. We use transaction delay to determine whether the system can reach consensus, as shown in Fig. 9.

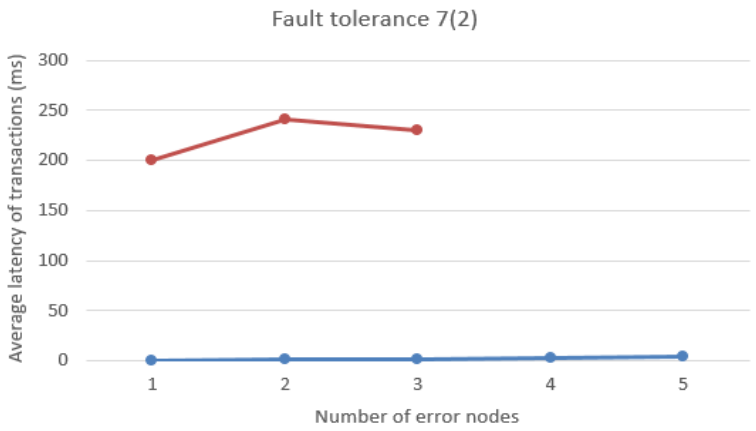

Figure 9: EPBFT's fault tolerance, where the blue dots indicate the number of error nodes, yellow dots indicate the corresponding time required to reach the transaction. It can be seen from the figure that when the error node is greater than or equal to 3, the transaction delay is not within the specified time range (theoretical is infinite), and consensus cannot be reached. So EPBFT and PBFT maintain the same fault tolerance

4.2.3 Communication Complexity

In PBFT, three stages need to be broadcast. In each broadcast process, to ensure the security of the data, each node independently processes the message: Accept the transaction or block, and after it is verified, broadcast to all other nodes in the network. In this process, a large amount of inter-node communication is required. If there are n nodes in the current system, the communication complexity of the system during this consensus process can reach O(n2). To reduce communication complexity, in EPBFT, a coordination node is randomly selected during each consensus process. Although the existence of coordinating nodes reduces the independence between nodes and increases the probability of joint attacks, it greatly reduces the communication complexity of messages.

We set the system need to complete the consensus of 20 blocks. When the total number of nodes is: 7 (2), 13 (4), 19 (6), 31 (10), 46 (15), computing the communication complexity in the network. The number in parentheses indicates the number of error nodes, and the result is shown in Fig. 10.

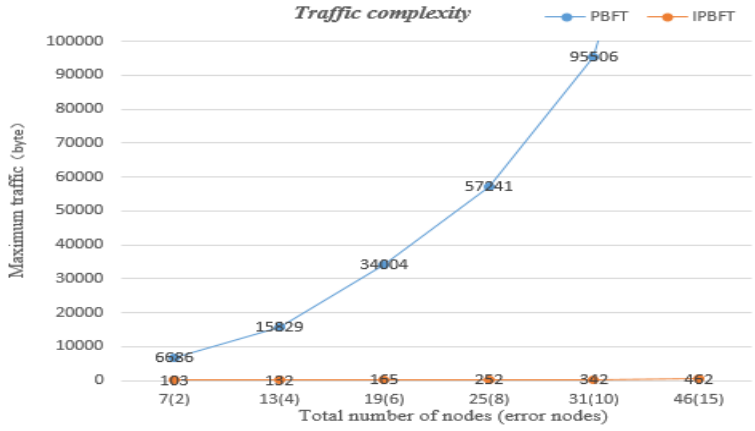

Figure 10: The comparison of PBFT's delay and EPBFT's delay

It can be seen from the figure that as the number of nodes increases, the maximum communication volume of the system in the PBFT algorithm increases rapidly. The EPBFT algorithm, although the maximum communication volume in the system, will increase with the number of nodes, but the increase

is relatively slow. From the analysis of the experimental results, we can see that our improved algorithm greatly reduces the communication complexity in the network.

4.3 Summary

In this chapter, we mainly analyze the function and performance of the EPBFT consensus algorithm through experiments. First, we test the scalability of the EPBFT algorithm by adding, deleting, and recovering nodes to the system. We then examined the delay, fault tolerance, and communication complexity of the EPBFT and PBFT algorithms, and compared them. Through experimental results, it can be found that compared with the PBFT algorithm, the EPBFT algorithm has a lower delay, communication complexity, better scalability, and more practical.

5 Conclusion

EPBFT is a Byzantine fault-tolerant consensus protocol that is scalable, practical, low latency, and low communication complexity. It is primarily used in licensed networks, such as the alliance chain. EPBFT inherits from PBFT, so it guarantees the safety and liveness of the system like PBFT. Besides, EPBFT also solves the fatal problem of PBFT: The number of nodes is immutable, and the communication complexity is high. It allows nodes to freely join or exit the network without introducing centralized components or rebooting the system and reduces the complexity of communication from O(n2) to O(n). We also set up a scoring mechanism for the nodes. Using the scoring mechanism, we can remove the faulty nodes in the network in time, which increases the robustness of the network. To be more suitable for the blockchain system, we also changed the C/S request-response mode of the classic PBFT to the P2P network topology response mode. In short, compared to PBFT, our protocol has better scalability, better communication performance, higher robustness, and better usability.

Funding Statement: This research was supported by Key Projects of the Ministry of Science and Technology of the People's Republic of China (2018AAA0102301), Project of Hunan Provincial Science and Technology Department (2017SK2405), CERNET Innovation Project (NGII20170715), (NGII20180902).

Conflicts of Interest: The authors declare that they have no conflicts of interest to report regarding the present study.

References

[1] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System,* 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf.

[2] H. R. Tang, N. Tong and J. Q. Ouyang, "Medical images sharing system based on blockchain and smart contract of credit scores" in *Proc. 1st IEEE Int. Conf. Hot Inf.-Centric News*, pp. 240–241, 2018.

[3] W. She, Y. Hu, Z. Tian, G. Liu, B. Wang *et al.,* "Secure model of medical data sharing for complex scenarios," *Journal of Cyber Security*, vol. 1, no. 1, pp. 11–17, 2019.

[4] D. Kim, S. D. Min and S. Kim, "A DPN (Delegated Proof of Node) mechanism for secure data transmission in IoT services," *Computers, Materials & Continua,* vol. 60, no. 1, pp. 1–14, 2019.

[5] Y. Yan, Y. Dai, Z. Zhou, W. Jiang and S. Guo, "Edge computing-based tasks offloading and block caching for mobile blockchain," *Computers, Materials & Continua,* vol. 62, no. 2, pp. 905–915, 2020.

[6] R. Song, Y. Song, Z. Liu, M. Tang and K. Zhou, "GaiaWorld: a novel blockchain system based on competitive PoS consensus mechanism," *Computers, Materials & Continua,* vol. 60, no. 3, pp. 973–987, 2019.

[7] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM,* vol. 4, no. 3, pp. 382-401, 1982.

[8] L. Lamport, "The part-time parliament," *ACM Transactions on Computer Systems*, vol. 16, no. 2, pp.133-169, 1998.

[9] L. Lamport, "Paxos made simple," *ACM SIGACT News*, vol. 32, no. 4, pp. 18–25,2001.

[10] D. ongaro and J. Ousterhout, "In search of an understandable consensus algorithm," Ph.D. dissertation, Stanford University, USA, 2014.

[11] M Castro and B Liskov, "Practical byzantine fault tolerance," in *Proc. Proc. of the 3rd Sym. on Operating Systems Design and Implementation,* New Orleans, USA, pp. 173−186, 1999.

[12] A. Bessani, J. Sousa and E. E. P. Alchieri, "State machine replication for the masses with BFT-SMART," in *Proc. 2014 44th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks*, Atlanta, GA, USA, pp. 355-362, 2014.

[13] X. F. Liu, "Research on blockchain performance improvement of byzantine fault-tolerant consensus algorithm based on dynamic authorization," Ph.D. dissertation, Zhejiang University, 2017.

[14] H. Xu, Y. Long, Z. Q. Liu, Z. Liu and D. W. Gu, "Dynamic practical byzantine fault tolerance," in *Proc. 2018 IEEE Conf. on Communications and Network Security (CNS),* Beijing, China, pp. 1-8, 2018.

[15] Y. Jiang and Z. Lian, "High performance and scalable byzantine fault tolerance," in *Proc. 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conf. (ITNEC)*, Chengdu, China, pp. 1195-1202, 2019.

[16] C. Li, G. Xu, Y. Chen, H. Ahmad and J. Li, "A new anti-quantum proxy blind signature for blockchain-enabled internet of things," *Computers, Materials & Continua*, vol. 61, no. 2, pp. 711–726, 2019.

[17] X. Wang, J. W. Li and J. Chai, "The research on the incentive method of consortium blockchain based on practical byzantine fault tolerant," in *Proc. 2018 11th Int. Sym. on Computational Intelligence and Design (ISCID),* Hangzhou, China, pp. 154-156, 2018.

[18] Z. L. Xu, H. M. Feng and B. Liu, "Study of highly efficient PBFT consensus mechanism based on credit," *Application Research of Computers,* vol. 10, no. 10, pp. 1-2, 2019.

[19] X. Jiang, M. Liu, C. Yang, Y. Liu and R. Wang, "A blockchain-based authentication protocol for WLAN mesh security access," *Computers, Materials & Continua,* vol. 58, no.1, pp. 45–59, 2019.

[20] G. Sun, S. Bin, M. Jiang, N. Cao, Z. Zheng *et al*., "Research on public opinion propagation model in social network based on blockchain," *Computers, Materials & Continua,* vol. 60, no. 3, pp. 1015–1027, 2019.

[21] K. Li, H. Li, H. Hou, K. Li and Y. Chen, "Proof of vote: a high-performance consensus protocol based on vote mechanism & consortium blockchain," in *Proc. 2017 IEEE 19th Int. Conf. on High Performance Computing and Communications,* Bangkok, Thailand, pp. 466-473, 2017.

[22] G. S. Veronese, M. Correia, A. N. Bessani and L. C. Lung, "Spin one's wheels? byzantine fault tolerance with a spinning primary," in *Proc. 2009 28th IEEE Int. Sym. on Reliable Distributed Systems*, Niagara Falls, NY, Canada, pp. 135-144, 2009.