High Performance and Scalable Byzantine Fault Tolerance

Yanjun Jiang*
School of Computer Science
Beijing University of posts and telecommunications
Beijing,China
Jiangyanjun0718@bupt.edu.cn

Zhuang Lian
School of Computer Science
Beijing University of posts and telecommunications
Beijing,China
lianzhuang@bupt.edu.cn

***Abstract*—With the great success of cryptocurrencies, people have paid great attention to blockchain technology and Byzantine fault tolerance(BFT) for consensus. Practical Byzantine Fault Tolerance protocol (PBFT) proposed by Castro and Liskov in 1999 is recognized as the most classic protocol for solving the Byzantine generals problem, but it still has many fatal disadvantages. Firstly, its communication complexity in the consensus process has reached O(n^2), so that it can only be applied in a small-scale system. Secondly, it works in a completely enclosed environment, nodes are not allowed to participate or exit the network without restarting the system, making it difficult to get applied in the actual system. In order to solve these problem, this paper proposes the High Performance and Scalable Byzantine Fault Tolerance, an improved BFT protocol, called HSBFT. In normal case operation, the HSBFT's communication complexity is reduced to O(n), and it also has the ability to be scalable, making it adaptable to more diverse practical scenarios.**

Keywords—distribute consensus; Byzantine Fault Tolerance; Byzantine generals problem; communication networks; message passing; PBFT; High Performance Byzantine Fault Tolerance; Scalable Byzantine Fault Tolerance.

I. INTRODUCTION

The blockchain technology, proposed by Satoshi Nakamoto in 2008, has achieved great success in recent years [1].BlockChain has been widely used to provide decentralized data protection in the field of digital currency [2],cloud computing [3], smart contract [4], and finance [5,6].The distributed consensus protocol, one of the cornerstones of the entire blockchain technology, has also become a research hotspot.

The distributed consensus protocol is to solve the problem of how to make the right decision in a distributed network in the case of a few malicious nodes exhibiting arbitrary behavior, called Byzantine generals problem [7]. Research based on Byzatine generals problem has been going on for a long time. However,most ealier work [8-12] either concerns techniques designed to demonstrate theoretical feasibility that are too inefficient to be used in practice, or easy for adversaries to violate their assumptions, such as synchrony [13] in the network.

PBFT,proposed by Castro and Liskov in 1999 [14], is the first practical Byzantine fault tolerance protocol. PBFT is a protocol for state machine replication and provides liveness and safety [15]. It implements Byzantine fault tolerance through three-stage message passing in the case where the number of malicious nodes in the network does not exceed $\lfloor\frac{N-1}{3}\rfloor$, where N is the total node number of the network. but it still has some fatal disadvantages. Firstly, the communication complexity of PBFT in the consensus process is O(n^2), so it can only be applied in small-scale systems. Secondly, the PBFT algorithm works in a completely enclosed environment, making it difficult to get applied in the actual system (such as P2P or blockchain network). In order to solve these problems of PBFT, a High Performance and Scalable Byzantine Fault Tolerance protocol, called HSBFT, is proposed in this paper.

HSBFT is a PBFT-based protocol that inherits all the advantages of PBFT, providing the same safety and liveness as PBFT. It is also a protocol based on weak synchrony assumptions[16], which is an important property to make it practical in Internet. HSBFT addresses the two fatal disadvantages of PBFT mentioned above. In normal case operation, HSBFT reduces the communication complexity of the two phases of prepare and commit from O(n^2) to O(n), which means that the total communication complexity of the entire protocol is O(n). At the same time, HSBFT brings the scalability to the network. nodes can freely participate or exit the consensus network, making it possible to apply in practical distributed systems (such as consortium blockchain network [17]). In addition, HSBFT gives a possible solution for kicking malicious nodes out of the network, and perfects the data synchronization algorithm, called node recovery protocol, which is not specified in PBFT. Compared with PBFT, HSBFT is more practical and adaptable in a actual system.

Thus, this paper makes the following contributions:

- The communication complexity of prepare phase and commit phase is reduced from O(n^2) of PBFT to O(n) without affecting the safety and liveness.
- It proposed an scalable Byzantine Fault Tolerance protocol for dynamically changing the number of nodes at runtime.
- A node recovery protocol is proposed to synchronize network data and state when some nodes are out of sync.

The closest work to us is the improvement of scalability proposed by Xiaofei Liu[18] and X.Hao[19].The former introduces a series of alternate nodes in the network. When a consensus node fails, one alternate node will replaces the fail node. In fact, this method does not realize the scalability of the network. The total number of consensus nodes in the network is unchanged, but it has improved the robustness of the network to some extent. The latter's assumptions are too idealistic, requiring the newly joined node to send a request to the network immediately after registering its information, otherwise the network consensus will be destroyed. At the same time, due to it use view-change protocol to achieve node's exit, the cost of node exit protocol is too high.

The rest of the paper is origanized as follows.We begin by describing our system model and some newly introduced definitions in chapter II. Chapter III describes the performance improving in HSBFT. Chapter IV details the implementation details and performance advantages of scalable BFT. Finally we conclude in chapter V.

II. Preliminaries

In this chapter,we will introduce the system model and outline of HSBFT.

A. System Model

The system assumption of the HSBFT is similar to the PBFT.We assume that there is an asynchronous network with malicious nodes. The malicious node can exhibit arbitrary behavior at any time during the running of the system. They can send error messages, delay the network communication of the correct node, and jointly initiate the destruction behavior with other malicious nodes. However, we assume that the malicious node cannot subvert the cryptographic techniques, nor can it delay the network indefinitely. HSBFT is an improvement based on PBFT, so the assumptions and system behavior of HSBFT are completely consistent with PBFT in the absence of special instructions.

There are two kinds of nodes in HSBFT: candidate and follower:

- The candidate nodes have stronger network communication capability and have a larger bandwidth. It can be candidated as a primary node when the view-change protocol is executed.
- The follower node only participates in the consensus but cannot be selected as primary.

The concept and role of candidate and follower will be detailed in Chapter III.

B. Node State Table (NST)

In HSBFT, the number and state of nodes in the network are allowed to change dynamically, so all nodes and clients need to maintain a node state table called NST, as shown in Table 1.

TABLE I: The Structure of node state table

node number	isCandidate	state	IP&PK	primary times
1	TRUE	ACTIVE	...	2
2	FALSE	EXITED	...	-

NST records information about all nodes in the network. State represents the current state of the node. It has three values: active, exited, and punished. Exited indicates that the node has voluntarily exited the network, punished indicates that the node's health score is too low to be kicked out of the network. isCandidate indicates whether the node is a candidate node, and primary times indicates the number of times the node has been worked as a primary. In addition, NST also saves basic node information such as IP address and public key. By looking up NST, We can get N, the number of active nodes currently participating in the consensus, and calculate the fault tolerance capability of the network through formula $f=\lfloor\frac{N-1}{3}\rfloor$.

C. NST Center

In HSBFT, the number and state of nodes in the network will change at any time. When a new node or client wants to join the network, it must obtain the node state in the current network. This process cannot be achieved only through the consensus between nodes. Therefore, the NST Center needs to be introduced to maintain the NST information. While introducing a central node in a distributed network environment is not a good solution. In order to maximize the maintenance of distributed decision characteristics of network nodes, the role of NST Center will be minimized in HSBFT. NST Center only used for new node or new client to get the network's node state. We assume that all nodes, clients and NST Center know each other's IP address, public key and other basic information when the network is initialized.

NST Center is a security service provider,only acts as an information storage node and does not participate in any consensus process. In the modern computer industry, it is easy to find a trusted node that all nodes recognize and do not exhibit error behavior.

D. Primary Selection and View Number

Due to the scalable property of HSBFT, the total node number n in the network changes at any time, so the primary selection method $p=v \bmod |\mathcal{R}|$ in PBFT is no longer applicable. The primary selection strategy in HSBFT is as follows:

When view-change occurs, starting from the row where the current primary is located in NST, look for the first node that satisfies its isCandidate property is true and its state is active as the new primary node. If the end of NST is reached, the condition is still not met , then search again from the first line of NST.

At the same time, the representation of the view number is also redefined. View number is represented as p-t in HSBFT, where p is the primary's node number of current view, t indicates that p is the t -th working as the primary, and the value of t is stored in NST(see Table I). For example, the view number at a certain time is 7-4, indicating that node 7 is the primary node, and node 7 is the fourth time working as the primary. In this way, it is ensured that no two views have the same view number throughout the entire process.

III. High Performance BFT

In this chapter, we will introduce the performance improvement of HSBFT compared to PBFT.

The high communication complexity of PBFT has always been criticized. The communication complexity of both prepare and commit stages have reached O(n^2) in PBFT, which makes it difficult to apply in a practical system.

In HSBFT, a new improvement idea is proposed. The primary is not only the initiator of the new message, but also the coordinator of the consensus process. In normal case operation, the communication complexity of prepare and commit stage will all be reduced to O(n), which makes it possible to form a larger consensus network. HSBFT is an improvement based on PBFT, so only the improvements in HSBFT are explained below. For convenience, we use backup to represent non-primary nodes.

A. Normal Case Opreation

When the primary p receives the message m with correct signature from the client, the three-phase protocol of HSBFT starts.The three phases are *pre-prepare,prepare*,and *commit*.

In the pre-prepare phase,the primary assigns a sequence number n to the request,multicasts a pre-prepare message with m piggybacked to all active backups, append the meaasge to its log, and starts the timer T_1. the message has form <<pre-prepare, v, n, d, *s-t*, Ω>$_{\sigma p}$, m> where v indicates the view number in which m is being sent, d is the digest of m, Ω is the local NST of the primary, *s-t* indicates the timestamp when the pre-prepare message was sent, σ_p is the signature of primary on the pre-prepare message.

A backup accepts a pre-prepare message provided:

- the signatures in the request and the pre-prepare message are correct and d is the digest for m ;
- it is in view v ;
- its local NST is same as the NST in the pre-prepare message ;
- it has not accepted a pre-prepare message with the same v, Ω,and n but containing a different diges d ;
- the sequence number in the pre-prepare message is between a low water mark, h, and a high water mark, $\mathcal{H}$. The definition of h and $\mathcal{H}$ is the same as in PBFT.

If backup i accept the pre-prepare message, it enters the prepare phase, replies a prepare message to the primary and adds message to its log.the prepare message has form <prepare, v, Ω, n, d, i >$_{\sigma i}$.

At the same time, the primary continuously accepts the prepare message from backups and performs the prepare message verification.After the primary node receives $2f$ correct prepare messages from different active backups,it multicasts the <prepare-collect, Φ, v, Ω, *s-t-collect*>$_{\sigma p}$ message to all active backups, which Φ represents the set of $2f$ correct prepare messages received by the primary.

After receiving the prepare-collect message, the backup i verifies the signature and check whether v, Ω is correct.then it extracts Φ, and performs prepare message verification for each of the messages. When $2f$ different messages pass the verification, the backup i enters the commit phase.

The prepare message <prepare, v, Ω, n, d, i >$_{\sigma i}$ passes the prepare message verification if node i is active, the signature is correct ,the view num and Ω are all the same as current node, and the digest and sequence num is same to the pre-prepare message the node received before.

The scheme of the commit phase is similar to the prepare phase. The same optimization idea is adopted. After the node i enters the commit phase, a <commit, v, Ω, n, d, i >$_{\sigma i}$ message is sent to primary,where d is the digest of m. After the primary receive $2f+1$ commit messages (one of them is generated by primary) with the same digest, NST,view number and sequence number from different backups, the <commit-collect, Φ, v, Ω, *s-t-collect*> $_{\sigma p}$ message is multicast to all active backups,where Φ is the set of $2f+1$ correct commit message and *s-t-collect* is the sending time of the message, then the primary perform the request and reply to the client. After receiving the prepare-collecct message of the primary, the backup execute the request and reply to the client if the commit-collect message verification succeeds.

When the client receives $f+1$ replies with the same result from different nodes, the request completes. Otherwise, if the client waits until timeout and still fails to collect $f+1$ identical replies, the request will be re-created by the client.

Figure 1 shows the operation of the HSBFT in the normal case of no primary faults. node 0 is the primary, node 3 is faulty,and C is the client.

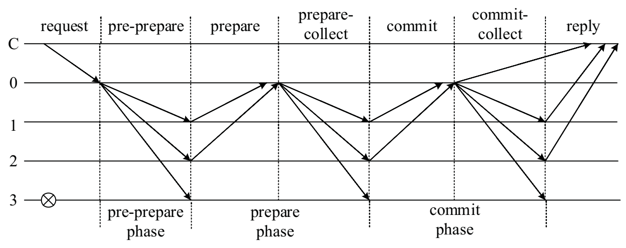

Fig. 1. Normal Case Operation of high Performance BFT

B. Other Case

With normal case operation, the communication complexity of both the prepare and commit phases is reduced to O(n) . However, in an asynchronous network, it is difficult to avoid the existence of network delay or the primary does evil, which may cause the above process to not be executed correctly.The following takes the prepare phase as an example to enumerate the abnormal situations that may be encountered and give solutions.

- Case A

If the primary receives $f+1$ prepare messages that are not accepted, it multicasts a <prepare-fail, v, Ω>$_{\sigma p}$ message directly to all active backups. After receiving the prepare-fail message, the backup starts the view-change protocol to switch to the next view.

- Case B

If the timer T_1 timeout and the primary still not receive $2f$ correct prepare messages, it multicasts <prepare-timeout, Φ, v, Ω>$_{\sigma p}$ message to all active backups, where Φ is the set of all prepare messages received by the primary.The backups extract the prepare message in Φ for prepare verification. If the verification finds that there are $2f$ correct prepare messages in

Φ, it is proved that the primary is incorrect, and the view-change protocol is initiated to switch to the next view. Otherwise, the consensus at prepare stage degenerates to the same scenario as PBFT. all nodes multicast prepare messages to other active nodes to re-consensus. In this case, the network communication complexity is $O(n^2)$.

- Case C

If a backup does not receive the prepare-collect message from the primary until the time of $s\text{-}t + T_2$ ($T_2 > T_1$),the backup starts the view-change protocol to switch to the next view.

In addition, the primary may maliciously delay the consensus in Case B. The priamry may deliberately discards a large number of prepare messages to achieve the illusion of prepare-timeout, forcing the consensus to degenerate and the entire phase re-consensus, which increases the network synchronization time and increases the communucation complexity of the protocol. To solve this problem, each backup will maintain a timeout-counter for the primary.When the times of prepare-timeout reaches the specified threshold, the view-change protocol will be run to change the primary.

The abnormal situation handling method of the commit phase is similar to the prepare phase, and detailed description is omitted here. In the prepare phase, backups use $s\text{-}t$ as the starting point of the timer to minimize the impact of pre-prepare message delay on the network consensus, making the network consensus more synchronous, reducing the delay of the view-change initiation process in Case C.

C. Candidate and follower

The HSBFT algorithm puts higher requirements on the bandwidth of the primary, so that the primary must have stronger network capabilities. But compared to PBFT, the communication pressure of backups is relatively lower. This results in different performance requirements for each node in the system. In order to reduce network costs, we divide the nodes in the network into two categories, candidate and follower (described in chapter II).

In PBFT, a series of proofs are carried out using $\mathcal{N}=3f+1$ as a special case. In the HSBFT, it is generalized to the general case of $\mathcal{N}\geq 3f+1$. Assuming that the current system is designed to be f-tolerant, the number of nodes $\mathcal{N}$ in the system should satisfy $3f+1\leq\mathcal{N}<3(f+1)+1$, so except for f macilious nodes, up to two correct nodes are allowed to be down, that is, the number of invalid nodes in the system may be up to $f+2$. This requires at least $f+3$ candidate nodes to ensure that at least one node in the network can act as the primary at any time. Therefore, only about one third of the nodes in the network need to be the candidate node, which ensures that the network is running correctly. After the introduction of the candidaite, the network cost is greatly reduced.

The above is the entire flow of the high performance BFT. the PBFT protocol adopts a conservative synchronization strategy, which requires a fully distributed multicast strategy in all phases, so that the communication complexity of the protocol is up to $O(n^2)$. Since the probability of high delay in the network is relatively low in the actual system, HSBFT adopts a more optimistic consensus strategy. In normal case operation, the communication complexity can be reduced to $O(n)$, and even in the worst case, the communication complexity is guaranteed to be the same as PBFT. Othermore, in HSBFT the primary acts as a coordination hub for the consensus process, which reduces the synchronization progress difference between nodes, the amount of data that needs to be transmitted is greatly reduced during view-change.

D. Correctness

In this chapter, we present a high-performance BFT optimized for both the prepare and commit phases. However, these optimizations are carried out internally in these two phases. The node still needs to receive $2f+1$ correct messages to reach the consensus of every stage. so it does not affect the safty and liveness of the original PBFT protocol.

IV. SCALABLE BFT

Because PBFT can not dynamically add or remove nodes during the network operation, it is difficult to be effectively applied in the actual system. This chapter will introduce a practical scalable Byzantine fault-tolerant protocol.

A. View Change Protocol

In PBFT,The view-change protocol provides liveness by allowing the system to make progress when the primary fails [14].

HSBFT follows the same idea, and the process and role of view-change protocol are basically the same as in PBFT. However, due to the scalable property of HSBFT, the NST is also added as a judgment evidence of the view-change. In HSBFT,view-change protocol can be used not only for view switching but also for NST updating. In the following two cases, node i will execute the view-change protocol:

- The primary is abnormal and node i wants to switch to the next view. Such as the timer of node i expires.
- The network's NST needs to be updated, such as the node's participation and exit that will be discussed next.

The view change protocol in HSBPF is discribed as follows (only a brief description is given, see[14] for detailed proof).

When a node i in view v starts a view-change protocol, it stop accepting messages (other than checkpoint messages, view-chang messages, and new-view messages) and send a $\langle$view-change, $v^*, \Omega^*, n, \mathcal{C}, \mathcal{P}, i\rangle_{\sigma i}$ message to the primary of view v^*. Here v^* is the view num of the view that node i wants to switch to, Ω^* is the NST that node i wants to update to, n is the sequence number of the latest stable checkpoint s known to i, $\mathcal{C}$ is a set of $2f+1$ valid checkpoint messages proving the correctness of s ,and $\mathcal{P}$ is a set containing a set $\mathcal{P}_m$ for each request m that prepared at i with a sequence number higher than n. Each set $\mathcal{P}_m$ contains a valid pre-prepare message and $2f$ matching, valid prepare messages signed by different backups with the same view number, NST, sequence number, and the digest of m.

When the primary p of view v^* receives $2f$ valid view-change messages for view v^* from different active nodes, it enters view v^*, update its local NST to Ω^* and multicasts a $\langle$new-view,v^*,Ω^*,$\mathcal{V}$, $\mathcal{O}\rangle_{\sigma p}$ message to all active backups,where $\mathcal{V}$ is a set containing the valid view-change messages received by the primary plus the view change message for v^* the primary sent (or would have sent), and $\mathcal{O}$ is a set of pre-preare messages created from all messages from sequence *min-s* to *max-s*. *min-s* is the sequence number of the latest stable checkpoint in $\mathcal{V}$ and *max-s* is the the highest sequence number in a prepare message in $\mathcal{V}$. The definition of $\mathcal{O}$ is similar to that in PBFT.

A backup accepts a new-view message for view v^* if it is signed properly, if the view-change messages it contains are valid for view v^* or for Ω^*, and if the set $\mathcal{O}$ is correct; it verifies the correctness of $\mathcal{O}$ by performing computation similar to the one used by the primary to create. Then the backup enters view v^* and update its local NST to Ω^*, and reconsensus the pre-prepare messages in set $\mathcal{O}$.

Different from PBFT,view-change can be used not only for view switching but also for NST updating in HSBFT. Therefore, view-change can be divided into the following three categories in HSBFT. The format of the message is:

- The $\langle$view-change, v^*, Ω, n, $\mathcal{C}$, $\mathcal{P}$, $i\rangle_{\sigma i}$ message indicates a switch to the next view.
- The $\langle$view-change, v, Ω^*, n, $\mathcal{C}$, $\mathcal{P}$, $i\rangle_{\sigma i}$ message indicates that NST is to be updated to Ω^*.
- The $\langle$view-change, v^*, Ω^*, n, $\mathcal{C}$, $\mathcal{P}$, $i\rangle_{\sigma i}$ message means to update NST to Ω^* while switching view to v^*.

The correctness of view-change protocol has been fully proved in PBFT. In HSBFT, the NST is added as a judgment evidence, view number and NST together play the role of the view number in PBFT. it does not affect the correctness of the view-change protocol.

In the scalable BFT , view-change protocol will play a very important role, especially in the update of NST.

B. *New Node Participation Protocol*

New node participation protocol is divided into two phase: *newnode-apply* and *newnode-sync*.

When a new node, called j, wants to join the network, it request the latest NST from the NST Center firstly. Then j multicasts $\langle$newnode,$\mathcal{A}$,$\Psi\rangle_{\sigma j}$ message to all active nodes, where $\mathcal{A}$ indicates the network authentication data, in different systems, it can be network authentication code or manual auditing or other personalized solutions. Ψ indicates the basic information of j, including the IP address,public key, whether it is a candidate node, and so on.

After receiving the new-node message, the node i check the correctness of the signature according to the public key in Ψ, and perform audit according to $\mathcal{A}$. If the audit is passed, a node number n is assigned to j, otherwise the node i do nothing. The allocation rules for n are as follows:

- Node i find j in the local NST firstly. If the j 's state is active or punished, the request will be rejected(the node in the punished state is not allowed to join the network again).
- If j is in exited state, it will still be assigned the original node number stored in NST.
- Otherwise, a node number n is assigned to j, and the value of n is the maximum node number value in the current NST plus one.

Then the node i sends a $\langle$newnode-callback, n, Ω, $i\rangle_{\sigma i}$ message to the j , where n indicates the node number assigned by node i for j, and Ω indicates the local NST of node i.

The j starts collecting newnode-callback message after multicasting the newnode message. When the j receives $2f+1$ correct newnode-callback message from different active nodes (signature verification is successful, Ω is same,node i is active,the assigned node number n is all same), it multicasts $\langle$newnode-collect, n, $\mathcal{O}\rangle_{\sigma j}$ messages to all active nodes, where n indicates the node number of j, and $\mathcal{O}$ is a set of $2f+1$ correct newnode-callback message from different node.

After receiving the newnode-collect message, the node i first verifies that the newnode-collect message's signature is correct, and then extracts the callback message in $\mathcal{O}$, if $\mathcal{O}$ contains $2f$+1 newnode-callback message with correct signature from different nodes, indicating that the nodes in the network have reached a consensus on the participation of j. Thereafter, the node i stop accepting messages (other than checkpoint, view-change, and new-view message), and adds node j to the local NST officially, sends $\langle$NST-add, n, $\Psi\rangle_{\sigma i}$ message to all clients and the NST Center. Then node i enters the newnode-sync phase. Clients and NST Center update their local NST after receiving $2f+1$ matching NST-add messages. The j 's certification and consensus process was completed during the newnode-apply phase.

In the newnode-sync phase, the request message that has not yet entered the checkpoint is redone with the help of the view-change protocol. Firstly, all nodes send a $\langle$view-change, v , Ω^*, n, $\mathcal{C}$, $\mathcal{P}$, $i\rangle_{\sigma i}$ message to the primary. According to the view-change protocol, the primary multicasts the $\langle$new-view, v, Ω^*, $\mathcal{V}$, $\mathcal{O}\rangle_{\sigma p}$ message to all active nodes(including j), after receiving $2f$ +1 view-change messages (where the j is still excluded when calculating f). when j receives the new-view message, it create its local NST through the $2f+1$ view-change messages in $\mathcal{V}$ and finds the current minimum legal stable checkpoint n according to the content of $\mathcal{V}$ as its first checkpoint, so that even if the next view-change happens, j can still participate in it .At the same tine, the current view numer v is also got from $\mathcal{V}$. And then j works with other nodes in the network redo the protocol for the pre-prepare message in $\mathcal{O}$ until the protocol finished. In the newnode-sync phase, the NST of the network changed, and j can provide all the functions of a node ,becomimg a formal node in the network.

Figure 2 shows the participation of j in HSBFT. node 0 is the primary,node 3 is faulty,C&C is client and NST Center, and j is the new node.

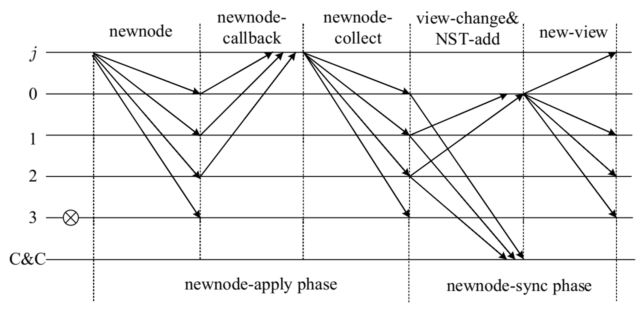

Fig. 2. The flow of New Node Participation Protocol

If on the newnode-sync phase,backup i does not receive the new-view message from the primary util timeout, the view-change protocol is initiated to switch to the next view.the backup i send <view-change, v^* , Ω^*, n, $\mathcal{C}$, $\mathcal{P}$, i $>_{\sigma i}$ message to the primary of view v^*, and update the NST while switching the view.

The new node participate protocol separates the admission consensus of j from the data synchronization. All the consensus protocols in the network are still in progress before the node enters the newnode-sync phase. Even if the consensus in newnode-apply phase fails or times out, only node j needs to resend the request without affecting the consensus process of the existing nodes in the system. By decoupling the new node admission consensus and the new node data synchronization process, all network delays in the newnode-apply phase and even node j malicious intrusion into the network will not have any impact on the existing consensus in the system (including correctness and performance).

C. Active Exit Protocol

When a node in the HSBFT network, called e, actively requests to exit the network, the following process is executed.

First, e multicasts an <exit, $\mathcal{R}$, $n>_{\sigma e}$ message to all active nodes, where $\mathcal{R}$ is the reason why the node e exits the network, and n is the node number of e. After receiving the exit message, node i reviews the exit request for e according to $\mathcal{R}$. After the audit is passed, node i replies a <exit-callback, i, $n>_{\sigma i}$ message to node e. After e receives $2f$ correct exit-callback messages from different active nodes, it multicasts the <exit-collect, $\mathcal{O}$, n $>_{\sigma e}$ message to all active nodes, where $\mathcal{O}$ is a set contains $2f+1$ correct exit-callback messages (one of which is generated by e itself), n is the node number of e .

After node i receives the exit-collect message from e, it first verifies the correctness of the signature, and then extracts each exit-callback message in $\mathcal{O}$. If $\mathcal{O}$ contains $2f+1$ matching exit-callback messages from different active nodes, it proves that all active nodes in the current network has reached a consensus on e's exit request from the network. Node i updates its local NST to set e's state to exited, after which all messages(other than newnode message) from e will be ignored,and messages in the network that have not yet reached consensus will make consensus based the new NST.At the same time, node i sends <NST-exit, n, i $>_{\sigma i}$ message to NST Center and all clients to synchronizes their local NST, where n represents the node number of e.. At this point, the active exit protocol ends, node e leave the network.

Figure 3 shows the active exit protocol in HSBFT. Node 0 is the exit node e,node 3 is faulty, and C&C is the clients and NST Center.

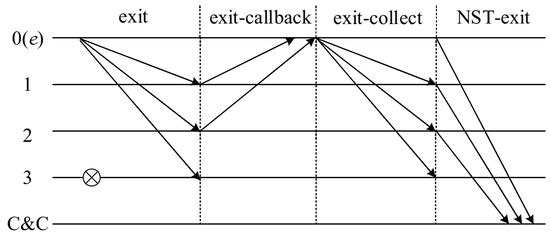

Fig. 3. The flow of Active Exit Protocol

Similar to the node participation, the active exit protocol have not any impact on the ongoing consensus until node e receives $2f$ correct exit-callback message. Even if the network delay occurs during the node exit process, it is only node e is affected because it needs to resend the exit request. The network cost of such a node exit protocol is very low.

D. Evil Exit Protocol

If a node continues to exhibit wrong behavior during the network consensus process, we can identify it as a malicious node. Removing malicious nodes is very beneficial to ensure the robustness of the network, but it may have to pay some performance cost. This paper gives a possible solution and analyzes the performance pressure brought by it. The reader can carefully choose whether to apply this solution for the specific application environment.

In order to constantly monitor the health of each node in the network, we introduce the concept of *health score*. When the system is initialized or a new node joins the network, the node's default health score is $\mathcal{H}_{init}$. In the process of the entire network consensus, each node is responsible for monitoring the health of other nodes in real time. For any node i in the network, each time it completes a message consensus, it need to check whether other nodes are also involved in this consensus. If a node is correctly involved in the consensus, the node's health score is increased by 1(no more than $\mathcal{H}_{init}$), otherwise its health score is reduced by 1. When node i detects that the health score of node m drops to 0, node i updates the local NST to set the state of node m to Punished and sends a <view-change, v , Ω^*, n, $\mathcal{C}$, $\mathcal{P}$, i $>_{\sigma i}$ message to the primary. If node m happens to be the primary, node i sends <view-change, v^* , Ω^*, n, $\mathcal{C}$, $\mathcal{P}$, i $>_{\sigma i}$ message instead to the primary of the next view. According to the view-change protocol, when the primary receives $2f+1$ matching view-change messages, it multicasts the new-view message to all active backups. After receiving the new-view message, node i sends messages to node m, all clients, and NST center to notify the node status changed. After the view-change protocol ends, node m are removed from the network, and it is no longer allowed to join the network again in general.

Figure 4 shows the evil exit protocol in HSBFT. node 0 is the malicious node m, node 1 is primary, and C&C is clients and NST Center.

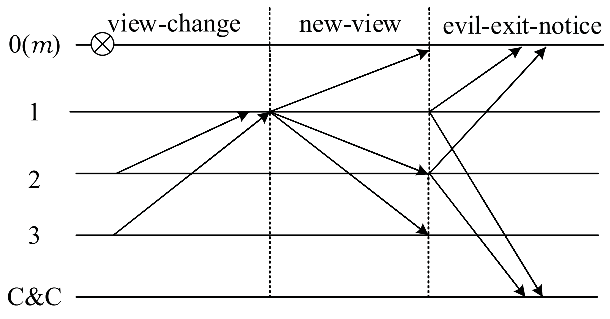

Fig. 4. The flow of Evil Exit Protocol

The following takes the commit phase as an example to describe the performance cost of the above protocol. Whether in PBFT or HSBFT, as long as a node receives $2f+1$ commit messages, the consensus can be completed. However,in order to implement the above evil exit protocol, it is necessary to continuously accept the all commit messages to monitor the health of every node. In the HSBFT protocol, the primary only needs to receive $2f+1$ commit messages and then execute commit-collect in the commit phase.While in the evil-exit protocol, the primary must wait for messages from all nodes until timeout, which will bring greater system bandwidth pressure and longer consensus delay. Therefore, in the actual environment, using this protocol requires more caution.

For the performance loss caused by the above global monitoring, it can be considered to only monitor the primary. When the view-change protocol execution causes the system to enter the next view,the primary's health score is reduced; when the consensus is successful, the primary's health score is increased. Since the primary does the most impact on the performance of the network (the view-change protocol needs to stop the entire consensus process, taking up a large bandwidth), this approach is a good solution to compromise performance and security to some extent.

E. Node Recovery Protocol

For a f fault-tolerant network, the total number of active nodes $\mathcal{N}$ in the network should satisfy, $3f+1\leq \mathcal{N}<3(f+1)+1$. Taking $\mathcal{N}=3f+2$ as an example, even if there is a correct backup (called b) down,the network can still work normally. However, when b restarts, it may not be able to participate the consensus correctly because its local state may not be synchronized with other nodes, such as with different view. Since the other correct nodes can still achieve synchronization, the view-change initiated by the b will not be supported, and b will continuously initiate the view-change protocol but will not get any response. To solve this problem, a node recovery protocol is proposed.

If b does not succeed in initiating view-change for f consecutive times, it proves that b itself has some problem. b executes the node recover protocol as follows.

Firstly, b requests the latest NST from the NST Center, and then multicasts $\langle$recover, $n, \mathcal{R}\rangle_{\sigma b}$ messages to all active nodes, where n indicates the node number of b, and $\mathcal{R}$ is the request reason of b.After receiving the message, node i verifies the signature and decides whether to allow it to execute the node recovery according to the $\mathcal{R}$. If node i supports this request, it replys a $\langle$recover-callback, $n, i\rangle_{\sigma i}$ message to b, where n is the node number of b. After node b receives $2f$ correct recover-callbak messages, it multicasts the $\langle$recover-collect, $\mathcal{O}\rangle_{\sigma b}$ message to all active nodes, where $\mathcal{O}$ is $2f+1$ correct recover-callback messages node b received(one of which is proposed by b itself).

After receiving the recover-collect message and verifying that $\mathcal{O}$ contains $2f+1$ correct callback messages with the correct signature, node i sends $\langle$view-change, v^*, Ω, n, $\mathcal{C}$, $\mathcal{P}$, $i\rangle_{\sigma i}$ message to the primary of the next view. Through the view-change protocol ,node b can get the correct state of the current network and the common latest stable checkpoint is also recovered from the network.

The node recovery protocol is similar to new node participation protocol, except that the former switches to a new view during view-change, while the latter updates the NST.

F. the Client

When a client newly joins the network or re-enters after leaving for a while,it needs to obtain the current network status from the NST Center and register its own information (like IP address,public key etc.) to the NST Center firstly. After this, when a node receives a request from an unfamiliar client, it can obtain the client's information from the NST Center.

Since the client is notified when the network node is added or deleted, clients already in the network do not need to update their local NST through NST Center. But In this way,it requires the node to record the information of all clients that has been connected to the network, which is very high performance requirements. Therefore, nodes in the network will delete clients that have not been connected for a long time from their local buffer.

G. Correctness

In this chapter, we present a series of protocols for nodes to dynamically participate and exit the HSBFT network. These protocols essentially complete the NST update through the consensus between the nodes. In the consensus process of each protocol, we ensure that least $2f+1$ active nodes agree. This also ensures that at least $f+1$ correct nodes agree the NST update(for the macilious nodes is no more than f). This means that the vast majority of the correct nodes have reached a consensus and the correctness of these protocol have been proven.

V. Conclusion And Future Work

HSBFT is a high-performance, scalable, practical Byzantine fault-tolerant consensus protocol based on PBFT for a permissioned network, like consortium blockchain [20]. HSBFT inherits all the advantages of PBFT and provides the same safety and liveness as PBFT. Morever, HSBFT has stronger communication performance and solves the fatal flaw that the number of nodes cannot be changed during the operation of the PBFT protocol. HSBFT reduces the communication complexity of prepare and commit phase from O(n^2) to O(n) and allows node to dynamically participate or exit whitout restarting the network, which makes the protocol more practical. In addition, HSBFT proposes a node recovery protocol that enables anomalous nodes to recover to the correct state through consensus.Therefore, compared with PBFT, HSBFT protocol has higher application value in practical systems.

In order for the newly joined node to obtain the network's state, HSBFT introduces a NST Center to maintain the node state table. But introducing a central node in a distributed network is not a perfect solution. Improvements in this area require more in-depth research in the future.

ACKNOWLEDGMENT

This work was supported by The National Key R&D Program of China (Project Number: 2016YFB0502404) .

REFERENCES

[1] Nakamoto, S. (2008). Bitcoin: a peer-to-peer electronic cash system. Consulted.

[2] Defrawy, K. E., & Lampkins, J. (2014). Founding digital currency on secure computation. 1-14

[3] Kõlvart, M., Poola, M., & Rull, A. (2016). Smart Contracts. The Future

[4] Kamhoua, C., Kwiat, K., Njilla, L. Liang, X., Shetty, & S., Tosh, D. (2017). ProvChain: A Blockchain-based Data Provenance Architecture n Cloud Environment with Enhanced Privacy and Availability.Ieee/acm International Symposium on Cluster, Cloud and Grid ompuing (pp.468-477). IEEE Press.

[5] Blockchain Technology in Finance Treleaven, Philip1; Brown, Richard Gendal2; Yang, Danny3 Source: Computer, v 50, n 9, p 15-17, 2017

[6] Treleaven Philip, Brown Richard Gendal,Yang Danny, Blockchain Technology in Finance, IEEE Computer Society,2017,50(9):15-17.

[7] Lamport L, Shostak R, Pease M. The Byzantine generals problem[J]. ACM Transactions on Programming Languages and Systems (TOPLAS), 1982, 4(3): 382-401.

[8] J. Garay and Y. Moses. Fully Polynomial Byzantine Agreement for n>3t Processorsin t+1 Rounds. SIAM Journal of Computing,27(1), 1998.

[9] D. Malkhi and M. Reiter. Unreliable Intrusion Detection inDistributed Computations. In Computer Security Foundations Workshop, 1997.

[10] R. Canetti and T. Rabin, "Optimal asynchronous byzantine agreement," Symposium on Theory of Computing, 1993.

[11] M. K. Reiter, "A secure group membership protocol," Software Engineering IEEE Transactions on, vol. 22, no. 1, pp. 31–42, 1994.

[12] K. P. Kihlstrom, L. E. Moser, and P. M. Melliar-Smith, "The securering protocols for securing group communication," in Hawaii International Conference on System Sciences, 1998, p. 317.

[13] L. Lamport, "Using time instead of timeout for fault-tolerant distributed systems." ACM Transactions on Programming Languages and Systems,vol. 6, no. 2, pp. 254–280, 2016.

[14] Castro M, Liskov B. Practical Byzantine fault tolerance. In: Proceedings of the 3rd Symposium on Operating Systems Design and Implementation. New Orleans, USA: OSDI, 1999. 173−186

[15] B. Liskov and M. Castro. A Correctness Proof for a Practical Byzantine-Fault-Tolerant Replication Algorithm. Technical Memo MIT/LCS/TM-590, MIT Laboratory for Computer Science, 1999.

[16] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of bft protocols," in ACM Sigsac Conference on Computer and Communications Security, 2016, pp. 31–42.

[17] Z. Zheng, S. Xie, H. Dai, X. Chen and H. Wang, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends," 2017 IEEE International Congress on Big Data (BigData Congress), Honolulu, HI, 2017, pp. 557-564.

[18] Liu Xiaofei. Research on blockchain performance improvement of Byzantine fault-tolerant consensus algorithm based on dynamic authorization[D].Zhejiang University, 2017.

[19] X. Hao, L. Yu, L. Zhiqiang, L. Zhen and G. Dawu, "Dynamic Practical Byzantine Fault Tolerance," 2018 IEEE Conference on Communications and Network Security (CNS), Beijing, 2018, pp. 1-8.

[20] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, and E. G. Sirer, "On scaling decentralized blockchains," in International Conference on Financial Cryptography and Data Security, 2016, pp. 106–125.

Authors' background

Your Name	Prefix*	Research Field	Email
Zhuang Lian	master student	BlockChain	lianzhuang@bupt.edu.cn
Yanjun Jiang	associate professor	Computer Network	jiangyanjun0718@bupt.edu.cn