An efficient Byzantine fault-tolerant consensus mechanism based on threshold signature

Wenxuan Jiang, LiquanChen, Yu Wang, Sijie Qian
School of Cyber Science and Engineering, Southeast University

一等奖

Abstract

In the application scenario of PKI system, the amount of written data is huge and the consensus will consume a lot of resources. In order to reduce the communication complexity and optimize the master node election process, this paper proposes Threshold Signature Practical Byzantine Fault Tolerant(TS-PBFT), an efficient Byzantine Fault Tolerant consensus mechanism based on the Threshold Signature. This consensus mechanism firstly adds the Threshold Signature technique to the PBFT algorithm with Byzantine Fault Tolerant, which reduces the communication complexity to O(n) level; secondly, it introduces the external monitoring mechanism and node trustworthiness index to improve the master node election strategy; Finally, it adds the batch processing mechanism to improve the performance of the consensus process. Compared with the PBFT algorithm, the TS-PBFT algorithm increases the throughput, reduces the delay, and decreases the out block time, which improves the performance of the algorithm.

Algorithm

1. Model of TS-PBFT Scheme

The entities involved in the mechanism improved in this paper include the Client, the Consensus Network (Quorum), the master node (Primary), the Commit-Message Signer (C-Signer), the Execute-Message Signer (E-Signer). The TS-PBFT consensus mechanism distributes some of the functions of the master node (Primary) to the signer node (C-Signer, E-Signer). The total number of nodes for the TS-PBFT consensus mechanism's threat model is n=3f+2c+1, which contains f Byzantine error nodes (malicious nodes) and c non-Byzantine error nodes (fault nodes).

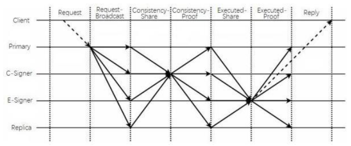

Fig. 1 Overall model of the TS-PBFT consensus mechanism

2. Quick consensus protocols (normal)

The quick consensus protocol process for TS-PBFT is as follows.

1) *Request phase:* The client C sends a transaction request r to master node P in consensus network Q with format *<Request,o,t,C>*.

2) *Request-Broadcast phase:* The master node P assigns a sequence number s upon the request, waits until at least *n>=b* requests has been received, and then assembles the n received requests into a transaction group $R=(r_1,\ldots,r_n)$. Finally, broadcast the request distribution message *<Request-Broadcast,s,v,R>* to all consensus nodes.

3) *Consistency-Share & Consistency-Proof phase:*

a) The backup node i verifies the received message and accepts. Calculate hash value *h=H(s,v,R)*. Sign the hash using a subkey to get $\xi(h)_i$. Send message <*Consistency-Share,s,v,*$\xi(h)_i$> to C-Signer.

b) C-Signer receives, checks and accepts message. Once C-Signer has received 3f+c+1 messages, it calculates overall threshold signature $\xi(h)$ and broadcasts a validation certificate message <Consistency-Proof,s,v,ξ(h)>. The backup node calculates hash and verify the overall threshold signature, execute the transaction group R if passes.

4) *Executed-Share & Executed-Proof Phase:*

a) The state update from state χ_{s-1} to χ_s after it completes all transactions in group R. The backup node I computes a hash $d=H(\chi_s)$ and perform a partial threshold signature $\pi(d)_i$ on d.

b) Send message <*Executed-Share,s,*$\pi(d)_i$> to the E-Signer. The E-Signer receives the message and verifies the partial threshold signature. If passed, log it to the local cache. Once E-Signer receives f+1 messages, calculate its overall threshold signature $\pi(d)$. Generate and broadcast execution confirmation message <*Executed-Proof,s,*$\pi(d)$>. Validates the signature and ends consensus.

5) *Reply phase:* E-Signer sends an execution confirmation message <*Executed-Done,o,l,s,*χ_s,$\pi(d)$*,Merkle*> to inform that the request o was executed successfully. The client C receives the message and checks its signature, and then does Merkle proof <*Verify,o,l,s,*χ_s,$\pi(d)$*,Merkle*> on it, and acknowledges if the message meets the expectation.

3. Linear Consistency Protocol (Exceptions)

If C-Signer does not collect 3f+c+1 shared signature message <*Sign-Share,s,v,*$\xi(h)_i$> within the specified time, the system enters a linear mode. The differences are as follows.

1) *Request-Broadcast phase:* Once receiving 3f+c+1 shared signature messages, The C-Signer (the master node) generates and broadcasts a request distribution message <*Request-Broadcast-Slow,s,v,*$\tau(h)$>.

2) *Consistency-Share-Slow & Consistency-Proof-Slow phase:* The backup node receives, verifies and accepts the message. Then generate a shared signature message <*Consistency-Share-Slow,s,v,*$\tau_i(\tau(h))$> and sent it to C-Signer. Once C-Signer receives 2f+c+1 message, calculates its overall threshold signature $\tau(\tau(h))$ and broadcasts message <*Consistency-Proof-Slow,s,v,*$\tau(\tau(h))$>. The backup node i calculates the hash, verify threshold signatures and executes the transaction group R.

4. View change protocols

TS-PBFT consensus mechanism has two modes. Therefore, TS-PBFT designs a new view-switching protocol.

1) *Master node election:* In this chapter, the Reliability Level (RL) is designed to measure the trustworthiness of a single node, which is used as a reference to elect a master node. The iterative calculation of the Reliability Level indicator for nodes is as follows:

$$RL_v = RL_{v-1} - \alpha * if\_leader - \beta * num\_audit + \gamma \quad (1)$$

With this algorithm, a node with higher credibility will have a higher probability of becoming a master node at each election; it also avoids the situation where a single node becomes a master node multiple times; and finally, it can effectively punish nodes for inaction by receiving external monitoring.

With this algorithm, a node with higher credibility will have a higher probability of becoming a master node at each election; it also avoids the situation where a single node becomes a master node multiple times; and finally, it can effectively punish nodes for inaction by receiving external monitoring.

2) *Launching a new view:* The backup node i generates a view switch message <*View-Change,v,*l_s,$\chi(l_s)s$,$\chi(l_{s+1}),\ldots,\chi(l_{s+win})$> and sends it to the new master node. The new master node receives at least 2f+2c+1 messages then sets the new view number *v*=v+1*, and sends the previously collected set of message I to all backup nodes in a new view message *<New-View,v*,I>*.

3) *Receive the new view:* Backup node i extracts message group I and processes the 2f+2c+1 messages one by one according to the sequence number size from lowest to highest. For each of these * messages, if a record of the fast-mode and the line-mode commit proof message is found in the local cache, then recognizing its state digest signature.

Experimentation

The Hyperledger Fabric v0.6 implementation is chosen to compare the performance differences between TS-PBFT and the PBFT consensus mechanism. The consensus is written in the Go language. The performance metrics are throughput, latency, and out-block time, respectively.

TS-PBFT has a higher throughput in fast mode (f=0) than in linear mode (f=1) . The throughput also decreases when there is a faulty node (c=1) in the consensus network.

Both PBFT and TS-PBFT have long processing times (between 40 and 120 seconds) at high concurrent requests. However, TS-PBFT has a communication complexity level in both two modes(f=0,1) .

As the number of concurrent requests gradually increases, both PBFT and TS-PBFT show a significant increase in the out-block time. At 100,000 concurrent requests, TS-PBFT continues to operate normally.

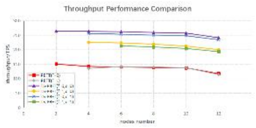

Fig. 2 Comparison of throughput performance

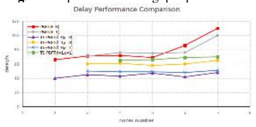

Fig. 3 Delay performance comparison

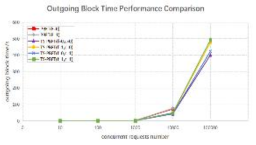

Fig. 4 Performance Comparison of Outgoing Block Time

Conclusion

In order to adapt to the PKI scenario, the functional requirements of communication complexity reduction, master node election optimization and consensus request batch processing are then proposed. This paper proposes an efficient Byzantine fault-tolerant consensus mechanism based on threshold signature TS-PBFT, and analyzes and compares the original PBFT consensus mechanism and the improved TS-PBFT consensus mechanism. The experimental results show that TS-PBFT outperforms PBFT in terms of throughput, latency and out-block time.

For the Byzantine fault-tolerant consensus mechanism, although the TS-PBFT proposed in this paper makes a great improvement in the efficiency of the consistency protocol, there is still a large room for improvement in the view switching protocol and the checkpoint protocol. At the same time, it is also a worthwhile optimization direction to integrate the BFT class consensus mechanism with the token-based consensus mechanisms such as PoW, PoS and DPoS to make up for the shortcomings of the small number of nodes that can be accommodated by the BFT class consensus mechanism with their high scalability.

Main References

[1] Zhen Y, Yue M, Zhong-Yu C, Chang-bing T and Xin C. Zero-determinant strategy for the algorithm optimize of blockchain PoW consensus. 36th Chinese Control Conference(CCC), 2017: 1442-1447.
[2]Košťál K, Krupa T, Gembec M, Vereš I, Ries M and Kotuliak I. On Transition between PoW and PoS. International Symposium ELMAR, 2018:207-210.
etc.

Contact Information

Contact Person：Wenxuan Jiang
Mobile Phone：15850535690
Mailbox：947796931@qq.com