Consensus in Lens of Consortium Blockchain: An Empirical Study

Hao Yin[1], Yihang Wei[1], Yuwen Li[1], Liehuang Zhu[2(✉)], Jiakang Shi[3], and Keke Gai[2(✉)]

[1] School of Computer Science and Technology, Beijing Institute of Technology, Haidian District, Beijing 100081, China
yinhao@bit.edu.cn,weiyongjian200708@126.com,yw.li@hotmail.com
[2] School of Cyberspace Security, Beijing Institute of Technology, Beijing 100081, China
{liehuangz,gaikeke}@bit.edu.cn
[3] China Mobile Research Institute, Xichen District, Beijing 100032, China
shijiakang@chinamobile.com

Abstract. Blockchain emerges as a public decentralized ledger system in recent years. Compared to the traditional distributed database, the blockchain realizes trustless property over the distributed network but consumes more computing resources and processing time. In blockchain, the consensus algorithm is the key component that guarantees such a significant property. To reach better performance, people adjust the network assumptions and classifies the blockchain into three types of the public, consortium, and private. Since consortium blockchain is prevalent studied and more practical, in this paper we investigate various representative consensus in the lens of consortium blockchain. By comparative analyzing those consensus algorithms, we find that deterministic consensus can speed up the transaction process in consortium blockchain. The related experiments are also conducted to understand what mainly causes the consensus delay. The results show that communication complexity seriously influences the algorithm performance. From this empirical study, we suggest that the message transmission path can be an optimized method to make research in future work.

Keywords: Consortium blockchain · Consensus algorithm · Byzantine fault tolerance · Performance · Empirical study

1 Introduction

Blockchain [21] establishes a decentralized database over a peer-to-peer (P2P) network, in which users do not need to trust each other nor a centralized organization. To build such a system, users run the open-source program code on the local computer to become a node. The blockchain can help users to spread the transactions and make an agreement via a consensus protocol over the whole

M. Qiu (Ed.): ICA3PP 2020, LNCS 12454, pp. 282–296, 2020.
https://doi.org/10.1007/978-3-030-60248-2_19

network. Since every transaction is witnessed by a majority of nodes in the network, blockchain builds a *trustless* [26] system from scratch. This property can cut down lots of costs in production activities, but blockchain's performance is not ideal for most practical applications.

Technically, we can optimize the consensus protocol to improve blockchain performance in creating blocks and processing transactions [7,32,33]. In general, we classify the type of blockchain according to the permission to make different assumptions for the nodes in the consensus protocol. Blockchain [6,34] is roughly divided into public, consortium, and private, where the nodes have trends to be stable and trustworthy. The weaker trust assumption leads to a simple consensus protocol thus getting better performance. It is a trade-off between performance and trustless. From a practical view, consortium blockchain takes both of them into account and very adopts to a real production environment.

However, existing consensus protocols often consider the scalability of public blockchains, or just apply the traditional distributed consistent algorithm into the consortium blockchain. It is rarely based on the features of consortium blockchain to reach optimal performance. In this paper, we revisit various representative consensus protocols and comparatively analyze them in the lens of consortium blockchain. From a highly abstract perspective, we can understand the key factors that affect the performance of the consensus protocol. Under this inspiration, we give a piece of advice in consensus optimization for future work. The contributions in our paper are as follows.

1. We synthesize the representative consensus algorithms and classify them into three types. In this way, we try to revisit these consensus algorithms without too many details, helping understand the essential design better.
2. We investigate those consensus algorithms in the lens of consortium blockchain, elaborating the process of reaching consensus under a high-level framework. Some features can be used to optimize performance.
3. We demonstrate the main reason for the performance bottleneck and take experimental verification. Such an empirical study inspires that communication path optimizing may improve the performance of consortium blockchain.

The rest of the paper is organized as follows. We give a revisit of the representative consensus in Sect. 3. Then we make a comparison and discussion in Sect. 4 from the perspective of consortium blockchain. Section 5 shows the experiments we conduct to verify the performance bottleneck. Finally, we conclude this paper in Sect. 6.

2 Related Work

The consensus is the main factor of performance bottleneck in transaction processing. There are a lot of works attempting to design different blockchain consensus against some specific scenarios [9,20], which can improve the performance for practical usage. Some survey works collect those schemes and discuss various views, making it easy to know about the study progress.

Bano et al. [1] conducted a systematic and comprehensive study of blockchain consensus protocols, where the consensus is classified into three types: proof-of-work (PoW), proof-of-X (PoX), and hybrid protocols. Xiao et al. [30] introduced a framework to analyze fundamental differences of various blockchain consensus protocols. Five core components are identified, including block proposal, block validation, information propagation, block finalization, and incentive mechanism, to make evaluations. Garay et al. [10] presented a roadmap for studying the consensus problem. They perform a landscape of consensus research in the Byzantine failure model, aiming to present a new class of blockchain consensus protocols.

Sankar et al. [28] discussed these proposed well-optimized Byzantine fault tolerant consensus protocols and analyzed their feasibility and efficiency in meeting the characteristics. Nguyen et al. [24] presented a review of the Blockchain consensus algorithms by categorizing them into two main groups, which are proof-based consensus and voting-based consensus. Salimitar et al. [27] took focus on a typical internet of things (IoT) network, which consists of several devices with limited computational and communications capabilities. Gudgeon et al. [12] structured the rich research on layer-two transactions, categorizing the research into payment and state channels as well as commit-chains. While the blockchain is used only as a recourse for disputes. Wang et al. [29] investigated sharding protocols used in blockchain, providing a systematic and comprehensive review of blockchain sharding techniques.

In our work, we make a classification of blockchain consensus protocols according to the ways of reaching an agreement. For blockchain, the longest chain principle is a type of probabilistic consensus while the message interaction is a type of deterministic consensus. We review these existing consensus protocols from the perspective of consortium blockchain for practical applications.

3 Consensus Mechanism

The core of blockchain consensus protocols is to select a leader for bookkeeping. The input of the consensus protocol is a sequence of transactions generated by nodes, and the outputs are the encapsulated data block appending to the blockchain. According to the ways of outputting block, we can classify blockchain consensus protocols into three categories: probabilistic consensus algorithms, deterministic consensus algorithms, and hybrid consensus algorithms.

3.1 Probabilistic Consensus

This type of consensus usually relies on the chain structure reaching a final distributed agreement. All peers in the blockchain network adopt the same strategy to determine which block should be appended to the current blockchain. Although temporary forks sometimes happen, they eventually converge to a particular chain to reach a consensus.

Proof of Work. Each node in a bitcoin [23] system solves a complicated but easy-to-verify puzzle. Once the majority of the whole network approves the solution, the node will obtain the bookkeeping rights for the block appending this block to the end of the blockchain. Such a difficult puzzle can be described as searching a random number to make the hash value of block less than or equal to the target value. The bitcoin system regulates the average block generation time to about 10 min by adjusting the search difficulty. If the average block generation time is less than 10 min observed from the previous 2016 blocks, the target value will be reduced to increase the difficulty.

Proof of Stake. Peercoin [15] first proposed an alternative blockchain consensus protocol called proof-of-stake (PoS) to address the problem of computing resource waste in PoW. The nodes with higher equity obtain the bookkeeping rights with higher probability. The PoS protocol does not require external physical input, which is more environmentally friendly than PoW. Furthermore, the 51% attacks in consensus protocols refer to the stake all miners hold instead of computing power. The advantage of PoS is that it can shorten the time needed to reach a consensus. It is also highly efficient and saves energy because it does not need a lot of computing power to solve problems.

Proof of Capacity. Like the PoW algorithm, the miners in Burstcoin [19] also need to solve complex problems to compete for mining rights. But the difference is that the most complex calculation can be cached through the design of proof-of-capacity (POC) consensus. It can consume storage space to replace with computing time. In POC, the miner needs to store results on the hard disk, and then the miner finds the required data in the previously generated cache data only to generate a new block. The more cached data the miner stores, the greater the chance of getting the bookkeeping right, so this consensus can encourage that the miner chooses larger storage space rather than greater computing power.

Proof of Elapse Time. This consensus method [5] was proposed by Intel in 2016 and is regarded as the core consensus algorithm of Hyperledger Sawtooth. The implementation of this consensus method needs the support of a trusted execution environment (TEE). Specifically, the proof-of-elapse-time (PoET) consensus algorithm used in Sawtooth is based on Intel's SGX (software guard extension) technology to ensure that trusted code is executed reliably. By running this consensus protocol, a new node downloads the trusted algorithm code and loads it into the environment of SGX. To simulate computing PoW solution, the node waits for the appropriate random waiting time, and then packs a block and broadcasts the block and the proof to network.

Proof of Burn. The allocation of mining rights by proof-of-burn [13] (PoB) is based on the user's initiative to "burn" the token. During this process, tokens that are intentionally destroyed are used as a way to "invest" in the blockchain

to prove investment in the network. The more tokens a user destroys in the system, the more likely they are to be selected as the next leader. PoB consensus ensures similar security to PoW without consuming computing resources. In the algorithm, a token is burned by sending it to a verifiable public "devourer" address without a corresponding private key. Therefore, the token sent to this address will not be used by anyone and cannot be circulated anymore.

3.2 Deterministic Consensus

This type of consensus applies a traditional distributed agreement protocol to the blockchain area. Each block is treated as a batch of requests that is ready to decide on the distributed network. The difference from traditional consistent algorithms is that there is a chain to link the sequential blocks and no fixed participants in the network.

Practical Byzantine Fault-Tolerance. Miguel Castro and Barbara Liskov [4] in 1999 proposed a practical Byzantine fault-tolerant algorithm (PBFT) to solve the efficiency in byzantine general problem. This algorithm reduces the message complexity from exponential level to polynomial level and makes it feasible in a practical system. The PBFT algorithm usually assumes that the network has $n = 3f+1$ nodes with tolerate fault f. At any time, there are only a master node and other slave nodes. The master node drives the protocol of message exchange to reach an agreement on at least $f+1$ honest nodes. Every node runs under the same configuration from the protocol, which is called a *view*. The view change is triggered when the master node failed to make a consensus.

Paxos. The Paxos [18] algorithm is a distributed consistency algorithm based on message passing. The algorithm solves the problem of distributed consistency under non-byzantine assumptions. Nodes may encounter network delay or even shut down without any response. The algorithm runs in an asynchronous network, which can tolerate message loss, delay, disorder, and repetition. There is also a master node that writes a consistent result to other slave nodes using two-stage message multicast. The algorithm uses a majority approach to ensure a f fault tolerance under the network scale of $2f + 1$.

Raft. The Raft [25] algorithm adopts a more simple requirement than the Paxos, which only considers a single proposer to write results. The algorithm must ensure that all nodes execute the same sequence of instructions and reach a consistent state after every round of the protocol. The nodes are divided into three states: leader, candidate, and follower. The state of nodes transition among these three states, which determined by an election procedure. After the leader node's election is completed, other nodes will transition to follower state and set the election timer. The leader node sends heartbeat packets to other nodes, resetting the follower nodes' timer. In this state, the raft algorithm will copy logs from the leader to followers.

Kafka. The consensus in a Kafka [17] is the leader-follower model where the heartbeat detection is implemented by Zookeeper. Followers will follow the leader to copy and remain consistent with the leader. If the leader fails, a new leader will be selected from the followers. The Kafka algorithm adopts the publish-subscribe model to sync messages. The producer of the message generates the message and submits it to the Kafka cluster. After sorting and consensus by the Kafka cluster, it is obtained by the subscribers. Kafka divides multiple topics according to the message type, while each topic can contain multiple partitions for redundancy. Kafka cluster saves messages for a period of time or until the number of messages exceeds a threshold. Consumers are required to actively poll for new messages.

3.3 Hybrid Consensus

This type of consensus combines the structure of blockchain and traditional distributed agreement protocols. Generally, it uses the probabilistic consensus to elect potential blocks and applies deterministic consensus to decide which block should be the final result.

Algorand. The Algorand [11] consensus is mainly divided into two steps. First, a generation group is randomly selected to generate new blocks, and then each new block is signed and broadcast. Second, a verification group is randomly selected to verify the new block broadcast, and in the verification group through an improved Byzantine protocol for consensus. In this process, the key thing is to ensure that all participants in each group are randomly and fairly selected. To solve the problem, the Algorand uses a verifiable random function (VRF) to designate the valid members. It is a hash function with asymmetric key technology, where the output is pseudorandom and can be publicly verified.

Elastico. Elastico [22] is a method based on the idea of fragmentation that divides the nodes in the network into several smaller committees. And then the consensus uses a byzantine fault tolerance protocol to reach consensus in each committee and deal with disjointed transaction sets. The node establishes its own identity through computing a PoW solution and joins the committee. By mixing the PoW, sharding, and BFT technologies, Elastico achieves a linear expansion of throughput.

Ouroboros. The Ouroboros [14] designed the Verifiable Secret Sharing protocol to reliably generate multi-party true random numbers. This method solves the problem that some nodes in the multi-party random number protocol fail to generate random numbers due to malicious or dropped lines. This algorithm forms a Merkle tree with all the rights and interests, where the leaf node of the tree is the equity value of an equity owner. The weight of the non-leaf node is the sum of the rights and interests of the left and right subtrees. According to

the random sequence, we start from the root of the Merkle tree, select the left and right subtrees, and finally reach the leaf node to select the block node.

Snow White. In the Snow White [2], there is a hash function that uses a random number seed to determine whether the members of the committee were leaders at each step. At the beginning of each era, there is redistribution for committees. First, the node finds the latest block with a timestamp of 2ω, where the prefix of this block will be used to confirm the members of the next era committee. Then, the node finds the latest block in the local blockchain with a timestamp before ω. Through the prefix of this block, we can get the random number seed of the hash function in the next era.

4 Comparison and Discussion

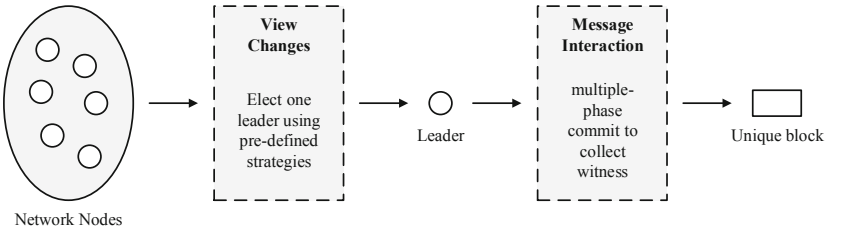
In essence, blockchain is a distributed database system that aims to reach an agreement on transaction log over the distributed network. The chain structure of blockchain brings new benefits and also challenges to the traditional distributed consistent algorithm. We comparatively analyze these types of consensus algorithms mentioned above.

4.1 Deterministic v.s. Probabilistic

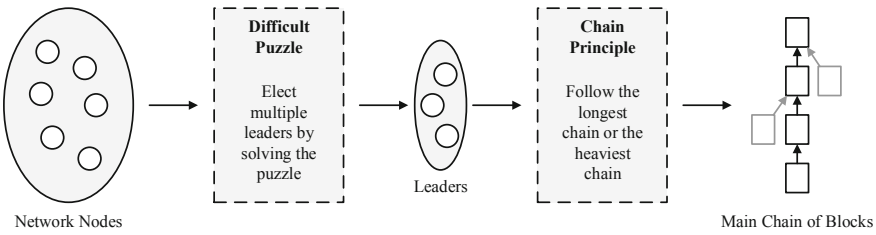
Deterministic consensus and probabilistic consensus are the two types of consensus with the biggest difference. The former ensures that each block output by the algorithm is valid, while the latter usually relies on the longest chain principle to make a new block valid with high probability.

To reach an agreement over the distributed network, there must exist a message (i.e. a new block in blockchain) proposed by a leader. In deterministic consensus, there is only one leader at each unit time. The leader broadcasts a new block and executes several rounds of message interaction to guarantee that most nodes indeed record the same block. We assume the network scale is $n = 3f + 1$, and the leader collects $n - f$ confirmation messages from all other nodes. We need al least $(n - f) - f > f$ confirmations to make a decision. In probabilistic consensus, there are many leaders at the same time. This type of consensus pre-define a puzzle to limit the number of leaders. The consensus relies on the longest chain principle to get confirmation as many as possible. The longest chain principle ($n > 2f$) implies that the proposed blocks in the longest chain have reached an agreement with a high probability as long as a majority of nodes, i.e. $n - f > f$, adopt the same strategy.

As shown in Fig. 1(a), since only one leader in each unit time, the deterministic consensus has to change the views from the network to choose a new leader when the current leader encounter faults. Such a leader election method only supports the network with a quorum. In terms of message interaction, consensus protocols use two-phase (e.g. Raft) or three-phase (e.g. PBFT) commit to decide on the proposed single block. As shown in Fig. 1(b), the probabilistic consensus



(a) The process in determinisitc consensus



(b) The process in probabilistic consensus

Fig. 1. The difference between deterministic consensus and probabilistic consensus.

lets the nodes solve a difficult puzzle to prove that they are valid leaders. In this process, consensus protocols consume different resources to design the puzzle. For example, PoW uses computing power, PoS uses virtual tokens, PoC uses local storage. The elected leaders simply broadcast those proposed blocks, relying on the chain principle (e.g. the longest chain or the heaviest chain strategies) to choose a block to append into the blockchain.

From the above comparison, we can find that using chain principle rather then message interaction can also design a consensus protocol although it is probabilistic. The chain structure provides an option that we can move the communication cost in the second process to the computing cost in the first process. It allows us to get stronger fault tolerance in a more dynamic network. Especially, when the scale of the network is big enough, such a probabilistic consensus can show better performance because of no message interaction.

4.2 Probabilistic v.s. Hybrid

Probabilistic consensus has great potential in a large-scale network because it gives up the complex message interaction. Instead, the actual agreement process is solved by a chain principle, which is fast in outputting a new block. But incurring forks in the chain structure is an unstable factor of consensus.

To let public blockchain more practical, the hybrid consensus replaces the chain principle with the original message interaction. It first selects a group of

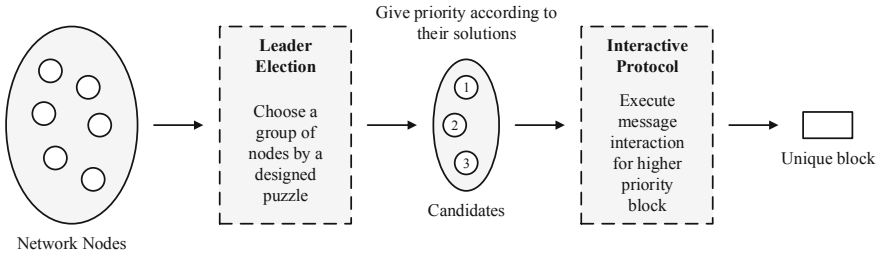


Fig. 2. The process in hybrid consensus.

candidates and then executes multiple-rounds interactive protocol in the group. However, an effective hybrid consensus cannot be designed by simply combining the two processes. Two key obstacles need to address. 1) Message interactive protocol in the deterministic consensus only handles a single block at one unit time. Thus, a leader election algorithm needs to decide the prepared block over the selected candidates. 2) Leader election algorithm in the probabilistic consensus substitutes static view changes strategy, which forces a message interactive protocol to allow the competition of several proposed blocks.

For better understanding, we use a similar framework to depict hybrid consensus in Fig. 2. Different from the probabilistic consensus that relies on chain principle to finally commit a block, the hybrid consensus can decide which block to commit after the first process completes. Due to the second process actually does the consensus work, we re-introduce the types of node faults into the hybrid consensus, resulting in that the ability to fault tolerance is consistent with the used message interactive protocol. To maintain this fault tolerance, the algorithm needs to securely narrow down a consensus group by randomly sampling the candidates over the distributed network. For example, Algorand adopts VRF to select a consensus group, where the size of the group can be adjusted by some parameters. Elastico leverages PoW to divides the network into several shardings. Likewise, the size of the sharding can be adjusted.

Compared to probabilistic consensus, hybrid consensus indeed has a better performance in reaching an agreement. There are neither forks in the blockchain nor too many message interactions in the network. For the public blockchain, the hybrid consensus is a good choice because it keeps the feature of a large-scale dynamic network. However, consortium blockchain can clearly specify a group of consensus nodes, skipping the leader election process.

4.3 Deterministic Consensus

The byzantine assumption lets each pair of nodes keep a point-to-point communication channel against the attacks from malicious nodes. To learn the difference between from non-byzantine fault assumption, we use an ideal model in secure multi-party computation (SMPC) to build a broadcast communication channel, assuming an ideal center that delivers messages to all nodes.

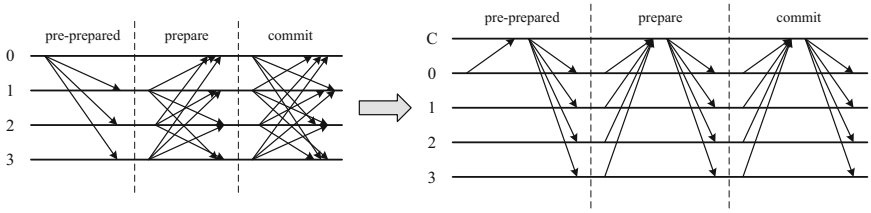
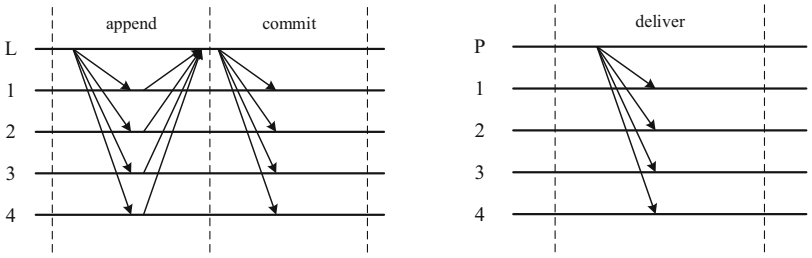


Fig. 3. The process of PBFT in the ideal model.



(a) The process in paxos or raft (b) The process in kafka

Fig. 4. The process of non-byzantine fault consensus.

As shown in Fig. 3, we omit the phases of request from users and the reply for users, simulating the process of PBFT consensus protocol in the ideal model. The ideal center becomes a hub to handles the votes from different nodes, sending the next phase's messages to each node. The only difference is that the complexity of the message is reduced because of the ideal center. The first two phases convince a node that a majority of nodes witness the same message, and the last phase notifies that a majority of nodes keep the same state for this message. Thus, we can safely reach an agreement over most of the honest nodes.

In contrast, it is easier for the consensus protocol with non-byzantine fault. We also illustrate the process of two-phase consensus protocol (e.g. Paxos and Raft) in Fig. 4(a). Since messages in the non-byzantine fault assumption cannot be tampered with, nodes only check that if a majority of nodes witness the same message. Moreover, Kafka uses zookeeper service to inspect whether the nodes are online or not, thus discarding the check phase. As shown in Fig. 4(b), Kafka more likes a backup system that syncs messages to replica nodes.

Although consortium blockchain organizes a relatively fixed group of nodes using a permission mechanism, only less trust to be needed between different organizations. A blockchain system is ought to establish trust based on the weaker trust assumption, which is why we are more interested in the blockchain instead of cloud computing [8]. As we discussed before, chain structure can benefit consensus in a probabilistic way. To improve performance, we need to

validate the bottleneck reason in PBFT and get some inspirations from it to optimize the consensus integrated with the chain structure in future work.

5 BFT Consensus Evaluation

BFT-based consensus protocols are the best choice of consortium blockchain, meeting both performance and application requirements. We investigate open-source code implementations and select the BFT-SMART [3] for deployment, evaluating the performance at different request sizes and network scales. Finally, we analyze the key influenced factors and give some advice in performance.

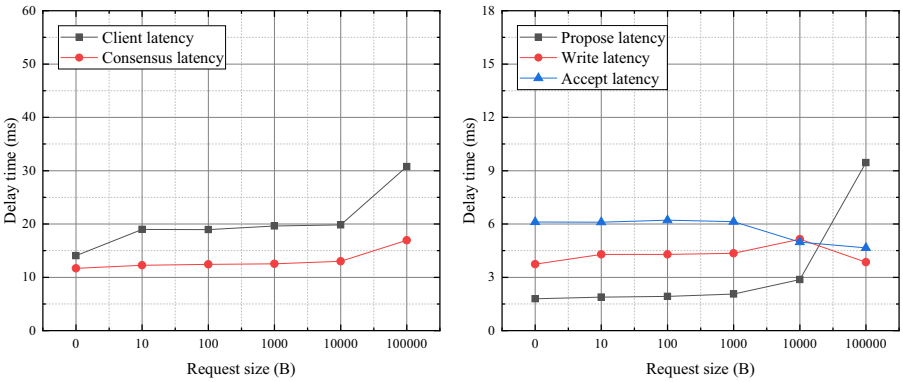
5.1 Benchmark Configuration

Generally, there are two performance indexes to evaluate the blockchain consensus, which are transaction throughput and response delay. The transaction throughput is controlled by the batch size and the batch timeout, which is hard to reflect the real performance on consensus protocol. We prefer to adopt the response delay to evaluate how each phase delays the system.

BFT-SMART is a high-performance Byzantine fault-tolerant algorithm library developed in Java language (with support for version 1.8 and above). This library executes the consensus under the network with $n = 3f + 1$ servers, following the three phases PROPOSE, WRITE, and ACCEPT to multicast messages. Only the PROPOSE phase contains the request from clients. The latter two phases only send a summary of the request, namely the hash value. We adjust some software configuration to measure the specific delay as follows. 1) The batch timeout is set to a negative value, which means that a new round of consensus starts right now after the last round completes. 2) The batch size is set to 1, which means that each block has only one request message, where servers do not cache requests. 3) the number of reply threads is set to 0, which means that servers use the main thread to reply consensus results to clients. Based on these configurations, we let a client constantly feeds one request to the consensus network, collecting the overall delay on average.

Table 1. Host configuration

Host operating system	Windows 10 64 bit
Host processor	Intel(R) Core(TM) i5-5250U CPU @1.60 GHz
Docker CPU	2 CPUs
Docker memory	2 GB
Docker container	library/java:latest



(a) overall delay (b) consensus phases delay

Fig. 5. The latency when increasing request size.

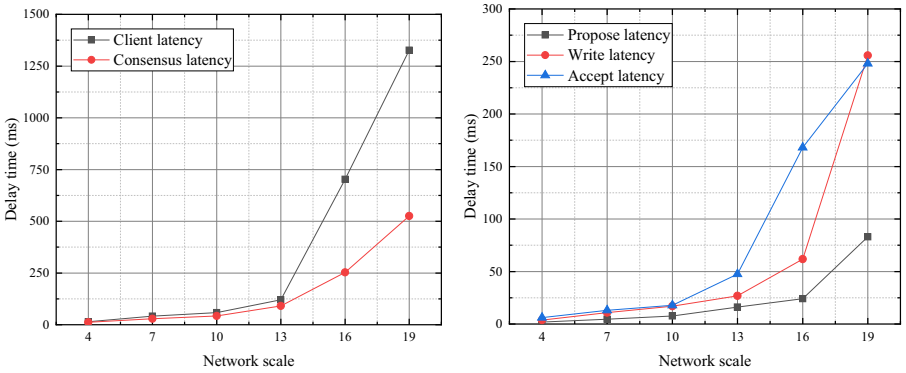
5.2 Latency Experiments

Setup. Table 1 shows the main configuration applied. We use a docker container to build the consensus network because it is easy to deploy with good compatibility. The BFT-SMART library is added to the running container to execute the benchmark programs and output results. We conduct our experiments on a laptop that installed the latest docker engine.

In our experiments, the severs are connected consisting of a network while a client just connects to one of them to send requests every an interval of time. Here we let the client send 1000 requests in total and the interval time is 1 second. Each server collects 100 samples to compute the average latency of the consensus process. In this way, the client can output the response latency and the servers can output the consensus and each phase latency. We learn the performance bottleneck at the two aspects of overall delay and consensus phases delay.

First, we conduct the latency experiments when increasing the request size from the client. As shown in Fig. 5(a), the overall delay is not so large when the request size is small enough. But if the request message is too large, it will cost too much time in transmitting the data. As a result, the latency raises a lot when the request size over 100 KB. Especially for client latency, it needs to collect at least $f + 1$ replies, causing the latency grows faster than consensus latency. Figure 5(b) shows the latency of each phase under the same situation. These three phases together consist of the whole consensus latency, where request size has mainly effect on the Propose phase.

Second, we also conduct the latency experiments when adjusting the network scale, where the request size is set to 0 in all experiments. We let the number of byzantine fault nodes be 1 to 6, thus having the network scale $n = 3f + 1$ from 4 to 19. As shown in Fig. 6(a), the overall delay significantly increases when the network scale becomes large. The message interaction in consensus causes large communication complexity. Especially when the network scale over 13 in



(a) overall delay (b) consensus phases delay

Fig. 6. The latency when increasing network scale.

our experiment, each server needs to collect enough votes from others, which leads to a network burden. Figure 6(b) shows the latency of each phase under the same situation. In the propose phase, servers only receive the message that ready to agree. In the last two phases, the pairwise servers need to exchange messages and thus causing the main latency of consensus.

The experimental results verify that the latency of each phase and overall delay present the same effect even though we run the servers in parallel. The massive protocol messages in the network cause a large latency. Thus, reducing message complexity is indeed a way to optimize consensus performance. In future work, we will study potential methods and consider some features of consortium blockchain to dynamically optimize the consensus in different cases. First, we can adopt an aggregate signature, such as Byzcoin [16], to cut down on extra communication, where a well-designed optimal routing path can be applied. Second, the chain structure can help accelerate the rate of block generation, like Hotstuff [31]. The phases in consensus can be parallelized. Third, we can supervise the consensus nodes in consortium blockchain, using the technique of proof of authority (POA) to reduce the risk of byzantine faults.

6 Conclusion

Consensus is an essential component in the blockchain, which can build a trustless system over the distributed network. This paper does an empirical study on the blockchain consensus algorithms. We revisit the representative consensus algorithms via classifying them into three types. Then, we carefully compare and discuss the difference between the types of consensus algorithms. Compared to probabilistic consensus, consortium blockchain is more adapted to deterministic

consensus for better performance. To find out the main reason for the performance bottleneck, we choose BFT-SMART library to make an evaluation for BFT consensus algorithm. The experimental results verify that message complexity leads to the delay in consensus. Finally, we conclude three aspects of improvement thoughts, which are an aggregate signature, chain structure, and proof of authority, to optimize the consensus for future work.

Acknowledgements. This work is supported by Ministry of Education - China Mobile Research Fund Project (Grant No. MCM20180401). Natural Science Foundation of Beijing Municipality (Grant No. 4202068), National Natural Science Foundation of China (Grant No. 61972034, 61902025), Natural Science Foundation of Shandong Province (Grant No. ZR2019ZD10), Guangxi Key Laboratory of Cryptography and Information Security (No. GCIS201803), Henan Key Laboratory of Network Cryptography Technology (Grant No. LNCT2019-A08), Beijing Institute of Technology Research Fund Program for Young Scholars (Dr. Keke Gai).

References

1. Bano, S., et al.: Consensus in the age of blockchains. arXiv preprint arXiv:1711.03936 (2017)
2. Bentov, I., Pass, R., Shi, E.: Snow white: provably secure proofs of stake. IACR Cryptology ePrint Archive, p. 919 (2016)
3. Bessani, A., Sousa, J., Alchieri, E.E.P.: State machine replication for the masses with BFT-SMART. In: IEEE/IFIP International Conference on Dependable Systems & Networks (2014)
4. Castro, M., Liskov, B., et al.: Practical Byzantine fault tolerance. In: OSDI, pp. 173–186 (1999)
5. Chen, L., Xu, L., Shah, N., Gao, Z., Lu, Y., Shi, W.: On security analysis of proof-of-elapsed-time (PoET). In: Spirakis, P., Tsigas, P. (eds.) SSS 2017. LNCS, vol. 10616, pp. 282–297. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69084-1_19
6. Gai, K., Wu, Y., Zhu, L., Qiu, M., Shen, M.: Privacy-preserving energy trading using consortium blockchain in smart grid. IEEE Trans. Industr. Inf. **15**(6), 3548–3558 (2019)
7. Gai, K., Wu, Y., Zhu, L., Xu, L., Zhang, Y.: Permissioned blockchain and edge computing empowered privacy-preserving smart grid networks. IEEE Internet Things J. **6**(5), 7992–8004 (2019)
8. Gai, K., Guo, J., Zhu, L., Yu, S.: Blockchain meets cloud computing: a survey. IEEE Commun. Surv. Tutorials **PP**(99), 1 (2020)
9. Gai, K., Wu, Y., Zhu, L., Zhang, Z., Qiu, M.: Differential privacy-based blockchain for industrial internet-of-things. IEEE Trans. Industr. Inf. **16**(6), 4156–4165 (2019)
10. Garay, J., Kiayias, A.: SoK: a consensus taxonomy in the blockchain era. In: Jarecki, S. (ed.) CT-RSA 2020. LNCS, vol. 12006, pp. 284–318. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-40186-3_13
11. Gilad, Y., Hemo, R., Micali, S., Vlachos, G., Zeldovich, N.: Algorand: scaling Byzantine agreements for cryptocurrencies. In: Proceedings of the 26th Symposium on Operating Systems Principles, pp. 51–68 (2017)
12. Gudgeon, L., Moreno-Sanchez, P., Roos, S., McCorry, P., Gervais, A.: SoK: off the chain transactions. IACR Cryptol ePrint Arch, p. 360 (2019)

13. Karantias, K., Kiayias, A., Zindros, D.: Proof-of-burn. In: Bonneau, J., Heninger, N. (eds.) FC 2020. LNCS, vol. 12059, pp. 523–540. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51280-4_28
14. Kiayias, A., Russell, A., David, B., Oliynykov, R.: Ouroboros: a provably secure proof-of-stake blockchain protocol. In: Katz, J., Shacham, H. (eds.) CRYPTO 2017. LNCS, vol. 10401, pp. 357–388. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63688-7_12
15. King, S., Nadal, S.: PPCoin: peer-to-peer crypto-currency with proof-of-stake. Self-published paper, 19 August 2012
16. Kokoris-Kogias, E., Jovanovic, P., Gailly, N., Khoffi, I., Gasser, L., Ford, B.: Enhancing bitcoin security and performance with strong consistency via collective signing. Appl. Math. Model. **37**(8), 5723–5742 (2016)
17. Kreps, J., Narkhede, N., Rao, J., et al.: Kafka: a distributed messaging system for log processing. In: Proceedings of the NetDB, vol. 11, pp. 1–7 (2011)
18. Lamport, L., et al.: Paxos made simple. ACM SIGACT News **32**(4), 18–25 (2001)
19. Larsson, T., Thorsén, R.: Cryptocurrency performance analysis of Burstcoin mining (2018)
20. Li, H., Gai, K., Fang, Z., Zhu, L., Xu, L., Jiang, P.: Blockchain-enabled data provenance in cloud datacenter reengineering. In: ACM International Symposium on Blockchain and Secure Critical Infrastructure, pp. 47–55 (2019)
21. Li, X., Jiang, P., Chen, T., Luo, X., Wen, Q.: A survey on the security of blockchain systems. Future Gener. Comput. Syst. **107**, 841–853 (2020)
22. Luu, L., Narayanan, V., Zheng, C., Baweja, K., Saxena, P.: A secure sharding protocol for open blockchains. In: The 2016 ACM SIGSAC Conference (2016)
23. Nakamoto, S., Bitcoin, A.: A peer-to-peer electronic cash system. Bitcoin (2008). https://bitcoin.org/bitcoin.pdf
24. Nguyen, G.T., Kim, K.: A survey about consensus algorithms used in blockchain. J. Inf. Process. Syst. **14**(1), 101–128 (2018)
25. Ongaro, D., Ousterhout, J.: In search of an understandable consensus algorithm. In: {USENIX} Annual Technical Conference, pp. 305–319 (2014)
26. Pilkington, M.: Blockchain technology: principles and applications. In: Research Handbook on Digital Transformations. Edward Elgar Publishing (2016)
27. Salimitari, M., Chatterjee, M.: A survey on consensus protocols in blockchain for IoT networks. arXiv preprint arXiv:1809.05613 (2018)
28. Sankar, L.S., Sindhu, M., Sethumadhavan, M.: Survey of consensus protocols on blockchain applications. In: 2017 4th International Conference on Advanced Computing and Communication Systems (ICACCS), pp. 1–5. IEEE (2017)
29. Wang, G., Shi, Z.J., Nixon, M., Han, S.: SoK: sharding on blockchain. In: 1st ACM CAFT, pp. 41–61 (2019)
30. Xiao, Y., Zhang, N., Lou, W., Hou, Y.T.: A survey of distributed consensus protocols for blockchain networks. IEEE COMST **22**(2), 1432–1465 (2020)
31. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: HotStuff: BFT consensus in the lens of blockchain. arXiv preprint arXiv:1803.05069 (2018)
32. Zheng, Z., Xie, S., Dai, H., Chen, X., Wang, H.: An overview of blockchain technology: Architecture, consensus, and future trends. In: 6th IEEE International Congress on Big Data (2017)
33. Zhu, L., Wu, Y., Gai, K., Choo, K.: Controllable and trustworthy blockchain-based cloud data management. Future Gener. Comput. Syst. **91**, 527–535 (2019)
34. Zhu, L., Gai, K., Li, M.: Blockchain Technology in Internet of Things. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21766-2