2021 IEEE 37th International Conference on Data Engineering (ICDE) | 978-1-7281-9184-3/20/$31.00 ©2021 IEEE | DOI: 10.1109/ICDE51399.2021.00165

Meepo: Sharded Consortium Blockchain

Peilin Zheng [†], Quanqing Xu [#], Zibin Zheng [†], Zhiyuan Zhou [#], Ying Yan [#], Hui Zhang [#],
[†]*School of Computer Science and Engineering, Sun Yat-sen University,*
[#]*Blockchain Platform Division, Ant Group*
zhengpl3@mail2.sysu.edu.cn, xuquanqing.xqq@antgroup.com, zhzibin@mail.sysu.edu.cn,
{wenzhang.zzy, fuying.yy, shengchu.zh}@antgroup.com

***Abstract*—Blockchain performance cannot meet the requirement nowadays. One of the crucial ways to improve performance is sharding. However, most blockchain sharding research focuses on public blockchain. As for consortium blockchain, previous studies cannot support high cross-shard efficiency, cross-contract flexibility, shard availability, and strict transaction atomicity, which are the essential requirements but also the challenges in consortium blockchain systems. Facing these challenges, we propose *Meepo*, a systematic study on sharded consortium blockchain. Meepo enhances cross-shard efficiency via the cross-epoch and cross-call. Moreover, a partial cross-call merging strategy is designed to handle the multi-state dependency in contract calls, achieving cross-contract flexibility. Meepo employs a replay-epoch to ensure strict transaction atomicity, and it also uses a backup algorithm called shadow shard based recovery to improve the shard robustness. We implement Meepo on the AliCloud, using 32 shards in maximum, achieving more than 120,000 cross-shard TPS under the workload of 100,000,000 asset transactions.**

***Index Terms*—blockchain, sharding, smart contract**

I. Introduction

Blockchain and blockchain-based smart contracts have wide applications nowadays, especially in consortium blockchain. Consortium blockchain systems are maintained by all the consortium members, carrying out the business such as cross-border exchange, goods tracing, and so on [22]. However, the performance of consortium blockchain cannot meet the requirement of applications [6]. For example, e-commerce shopping festivals require more than 100,000 TPS (Transaction Per Second) at the peaks, which is very large and cannot be supported by mainstream consortium blockchain systems. Therefore, improving the performance of consortium blockchain is urgent.

Sharding is one of the optional methods to improve blockchain performance. There are some studies that focus on blockchain sharding, such as Elastico [9], RapidChain [20], Monoxide [18], Dang et al. [5], Tao et al. [15]. The common idea of these studies is to divide the transactions and servers into different shards. Generally, each shard is a blockchain, of which the transactions are validated by several corresponding servers. These shards can communicate with each other through the cross-shard protocols. Thus the throughput of the entire network can be regarded as the sum of all shards, which has been improved.

However, the previous sharding studies focus little on consortium blockchain, and they cannot meet the following practical requirements of consortium blockchain: **(1) High cross-shard efficiency:** The transaction that needs to be executed on multiple shards is called a cross-shard transaction. In previous public blockchain sharding research, the cross-shard efficiency is sacrificed for security thanks to the incompleteness of the ledger. However, consortium blockchain requires complete ledger in practice and higher cross-shard efficiency than public blockchain. New cross-shard approaches need to be designed. **(2) Flexible smart contract calling:** Previous studies focus on the simple payment transactions or simple payment contracts on the blockchain. Nevertheless, most consortium blockchain applications are running on complex smart contracts. One contract on consortium blockchain might call the state in several shards and contracts, which cannot be supported well by previous work. **(3) Strict transaction atomicity:** The business models on consortium blockchain need strict transaction atomicity [10], and this atomicity should be ensured within one block. Previous sharding studies cannot ensure strict atomicity in such a short time within a block. **(4) Shard availability:** The data on consortium blockchain is read and written by the members far more frequently than public blockchain, requiring higher availability. Improving the complexity of the system will reduce availability. Thus a new method needs to be designed to enhance availability.

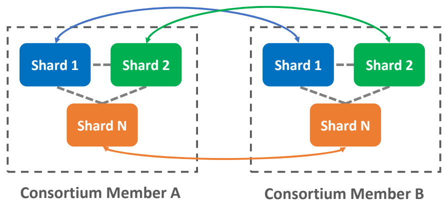


Fig. 1. Topology of Servers in Sharded Consortium Blockchain

To solve the above problems in consortium blockchain sharding, we propose *Meepo*[1], as its idea is to use Multiple Execution Environments Per Organization. The topology of servers in Meepo is shown in Figure 1, where each consortium member holds all the shards. In Meepo, we provide four solutions to the above challenges:

Cross-epoch and Cross-call is the main idea of this paper. It is an internal cross-shard protocol for the consortium

[1]It is also a legendary hero who can clone himself to obtain more power.

member. In Meepo, after the consensus of each block ends, each shard begins to communicate across shards. This process is so-called cross-epoch. Cross-call is the message from one shard to another, resulted from the cross-shard transactions, such as cross-shard payment. Meepo aggregates cross-calls into several cross-epochs in order. In this way, cross-shard communication can be done efficiently and orderly within each consortium member, ensuring the consistency of them.

Partial Cross-call Merging Strategy is used to improve the smart contract flexibility in sharded environments. Meepo allows smart contracts to send the cross-calls that are with partial parameters from different shards, then merges these partial cross-calls into a full one to execute on one shard, like a callback function. In this way, using only one transaction, the state of different shards can be parallelly operated or passed into one shard for complex smart contracts.

Replay-epoch is designed to ensure the transaction atomicity. Once any transaction exception occurs, all the shards will replay the transactions without the error, namely replay-epoch. Only the successful transactions will be executed, and the wrong ones will be reverted. In this way, Meepo provides atomic guarantees for cross-shard transactions within the duration of one block.

Shadow Shard Based Recovery is proposed to enhance the availability of the consortium member. It is a hot backup method. We equip the shards with several backup servers called shadow shards. Once any shard is down, the consortium member can switch to the corresponding backup to continue the execution of blockchain transactions. Thus it significantly reduces the impact of a single point of failure on system availability.

In summary, the contributions of this paper include the following:

- We propose Meepo, as a sharded consortium blockchain system, with a cross-epoch based cross-shard protocol designed for consortium blockchain, achieving higher cross-shard efficiency than the previous studies.
- We design a partial cross-call merging strategy to enable smart contracts to make flexible calling to multiple shards, meeting the requirement of complex business models on consortium blockchain systems.
- We propose a replay-epoch based method to cancel the operation of error transactions, ensuring strict transaction atomicity in the time of one block.
- We propose a hot backup method for sharded consortium blockchain, namely shadow shard, to improve the robustness of the consortium member.

To evaluate the feasibility and performance of Meepo, we develop Meepo based on Go-ethereum and carry out the experiment on AliCloud, with 32 shards in maximum. Under the workload of more than 100,000,000 transactions among 3,276,800 accounts, Meepo can achieve more than 120,000 cross-shard TPS, which is 20x than that in Go-ethereum using the same consortium consensus protocol.

II. Background

In this section, we will introduce the background with the basic concepts of blockchain, smart contract, and sharding.

The idea of blockchain was first proposed in Bitcoin [11]. According to peer permissions, blockchain systems can be divided into public blockchain, consortium blockchain, and private blockchain. As for consortium blockchain, although the network condition could be better than public blockchain thanks to fewer peers, the requirement of throughput is also higher. The data on consortium blockchain can be very large. And, the very large transaction data cannot be processed by the mainstream blockchain systems with high throughput and low latency. This paper focuses on improving the performance of consortium blockchain. It is worth noting that our method can also be used in private blockchain, to some extent.

A blockchain-based smart contract [14] is an event-driven program defined by its contract code stored with the blockchain [21]. Taking Ethereum [19] blockchain as an example, each smart contract has a state. The state can be changed by functions defined in the contract code. And the functions are called by the blockchain transactions [19]. The contract can be called by a user in the way of sending a blockchain transaction to the peers. The transaction consists of the address of the contract, the calling function, and the parameters [4]. A smart contract can also call another smart contract. Rather than sending a transaction, this calling is done by the message delivered in the contract execution environment [19].

Sharding is a common method in database systems [12] to break the bottleneck of limited resources. Therefore, the workload of blockchain can also be divided into different shards. Generally, blockchain sharding methods can be summarized into network sharding, state sharding, transaction sharding, and full sharding, as they accordingly partition the workload of the network, storage, and computation of the blockchain systems. In this paper, we use the sharding method that is close to the full sharding. However, the difference is that, in consortium blockchain, one consortium member needs to hold all the shards for the business data. And this will also provide the opportunity to achieve high cross-shard efficiency.

III. Meepo

A. Goals and Challenges

In most business scenarios in consortium blockchain, each consortium member needs to hold the complete ledger and state, since the business data is needed by all the consortium members. In other words, in practice, although the transactions are divided into different shards, the state of shards is best to be possessed by every consortium member to improve the authority and availability of the ledger. In this case, the topology of the sharded consortium blockchain is shown in Figure 1. The challenges are as follows:

C1: High cross-shard efficiency. Cross-shard transaction is a transaction executed in more than one shards. In sharded consortium blockchain, the cross-shard transactions might be very frequent. However, we cannot set up the account

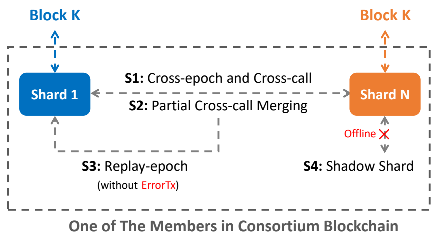

Block K
Block K
Shard 1
S1: Cross-epoch and Cross-call
S2: Partial Cross-call Merging
Shard N
Offline
S3: Replay-epoch
(without ErrorTx)
S4: Shadow Shard
One of The Members in Consortium Blockchain

Fig. 2. Overview of Cross-shard Solutions in Meepo

distribution to reduce the cross-shard communication, since the future user behaviors cannot be predicted. In this case, the overall sharded performance is limited to the cross-shard communication, which is the key challenge.

C2: Flexible smart contract calling. Previous blockchain sharding research focuses on a transaction model that is just from one shard to another, such as a payment. However, smart contracts are not always as simple as the payment, especially in consortium blockchain. For an e-commerce example, if a customer wants to order several items that are on different shards, his transaction needs to get prices and stocks from different shards. In consortium blockchain, smart contracts are more complicated due to the aggregation of more data and more complex business models. Thus that can be a long way in the network of sharded consortium blockchain, leading to inefficiency and challenge.

C3: Strict transaction atomicity. Previous studies can be summarized into two common methods to ensure atomicity. One is the asynchronous method, the other is a Two-phase Commit method (e.g.lock-commit), which is close to some concurrency control protocols [17]. Unfortunately, these two methods cannot work very well in sharded consortium blockchain. The first method cannot provide immediate confirmation, while the second method locks all the related accounts, preventing them from other transactions in this duration. Therefore, previous studies cannot be used on sharded consortium blockchain to ensure the strict transaction atomicity in acceptable short time, which is also an important challenge.

C4: Shard availability. In consortium blockchain, the availability is essential. As mentioned before, consortium members need to read and write the data of shards frequently. In addition, cross-shard communication of Meepo also needs the shards to be online (will be proposed in the next subsection), resulting in very high requirements for shard availability, which becomes a problem in sharded consortium blockchain. A simple hot backup of data cannot solve this problem, since the synchronization and switching of backups need to maintain communication and consistency with other shards.

B. Our Solution

Facing the above challenges, we provide the corresponding solutions in Meepo, as the overview shown in Figure 2.

S1: Cross-epoch and Cross-call.

Approach. To solve the problem in **C1**, we propose a cross-shard protocol based on the cross-epochs and cross-calls.

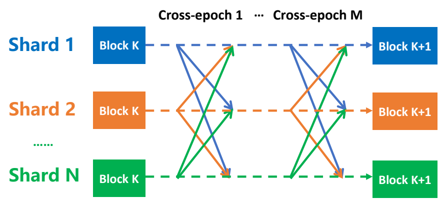


Fig. 3. Cross-epochs after Blocks as Cross-shard Communication in Meepo

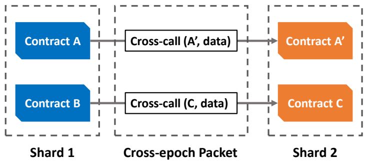


Fig. 4. Cross-calls in Cross-epoch of Meepo

The main idea is to import several cross-epochs between the blocks, as shown in Figure 3. After the block propagation and validation are completed, the shard communicates and executes cross-shard transactions. As shown in Figure 3, an arrow represents a data package from one shard to another. Each data package consists of a set of messages, which are the cross-calls to the same shard, executed by the target shard. The cross-calls are shown in Figure 4. When there are no cross-calls generated, the cross-epochs finish, and the shards begin to generate the next blocks. To help understanding, we provide the pseudocode example of an asset transferring smart contract, as shown in Algorithm 1.

Advantage. This cross-shard protocol handles the cross-shard communication within the consortium member, in order to improve the cross-shard efficiency. Since the communication just happens in the local network of the consortium member, the network speed can be fastened than the previous studies that are in the Internet environment. Moreover, in one member, different shards can trust each other because they belong to the same consortium member. However, in public blockchain, shards need to sacrifice efficiency to complete some kind of encryption proof. Therefore, by making use of the network condition and trustworthiness within the consortium member, the cross-shard efficiency can be improved.

S2: Partial Cross-call Merging Strategy.

Approach. Facing the challenge **C2**, we propose a partial cross-call merging strategy. As shown in Figure 5(a), Meepo allows the shards to send the cross-call with only partial parameters, namely partial cross-call, then merges them into one full call to execute on the target shard. Given the example of e-commerce, suppose a customer of *Shard 1* wants to buy the items of *Shard 2* and *Shard N*. In *Cross-epoch 1*, *Shard 1* sends the request (contract calling) to reduce the stocks and get the prices of the items. And then, *Shard 2* and *Shard N* respectively return the price in a partial call, without knowing the others' price. Finally, *Shard 1* merges their parameters by the hash value of the original transaction to deduct the customer's balance. We also provide a pseudocode example

Algorithm 1: Pseudocode of Example of Asset Transferring Contract Using Cross-call In Meepo

```
public function transfer (to, value):
    require(balanceOf[msg.sender]>=value);
    balanceOf[msg.sender] -= value;
    CrossCall(shardOf[to], this, "add", to, value);
private function add (to, value):
    balanceOf[to] += value;
```

Algorithm 2: Pseudocode of Example of E-commerce Shopping Contract Using Partial Cross-call in Meepo

```
public function buy (item1, item2):
    CrossCall(shardOf(item1), this, "getPrice1", item1);
    CrossCall(shardOf(item2), this, "getPrice2", item2);
private function getPrice1 (item1):
    CrossCall(origin, this, "settle", priceOf[item1], _);
private function getPrice2 (item2):
    CrossCall(origin, this, "settle", _, priceOf[item2]);
private function settle (price1, price2):
    balanceOf[tx.origin] -= (price1+price2);
```

of an e-commerce shopping contract using the partial cross-call in Meepo, as shown in Algorithm 2.

Advantage. This strategy improves the flexibility of smart contracts, enabling smart contracts to operate the state on different shards, even in parallel calls. Furthermore, it also enhances the cross-shard efficiency of some complex contracts. Suppose a smart contract depends on the state from *N* shards, without this strategy, it should go through a long way for *N* epochs. However, using the partial cross-call merging strategy, the number of epochs can be reduced to two, as it just calls the state parallelly and gets the callback.

S3: Replay-epoch.

Approach. To solve the problems in **C3**, we import replay-epochs among the cross-epochs to revert the state operation by error transactions. As Figure 5(b) shows, after the contract execution in one cross-epoch, if there are any errors, the shards will enter the so-called replay-epoch. In this epoch, the hash values of the error transactions will be propagated to all the shards, to remove the transaction from the block, then replay the block and the cross-epochs. In this way, the final executed blockchain transactions will all be the successful ones, and the error transactions will do nothing to the state of contract. It means that the state operations of any cross-shard transaction, will all succeed or fail.

Advantage. The most important advantage of the method is that it ensures the transaction atomicity in the duration of one block. This is also benefited from the high efficient cross-shard communication. As mentioned, public blockchain cannot achieve this since the network delay is too long for the error transactions to be propagated, resulted in the cross-block atomicity or lower availability of the locked state. However, as for consortium blockchain, where each member is holding all the state, the error transactions can be propagated in the local network, giving the time for the shards to replay the correct transactions. Our experiment shows that this method costs an acceptable time to replay the block, in exchange for the atomicity of the transaction, which will be evaluated in Section IV.

S4: Shadow Shard Based Recovery.

Approach. Facing the challenge of shard availability in **C4**, Meepo proposes a recovery approach based on the hot backup shard, namely *Shadow Shard Based Recovery*. The processes are shown in Figure 5(c). Meepo equips each shard with a shadow shard as a backup. Each pair of shards is like a master-slave backup architecture. The shadow shard copies the block and state from its master shard, without joining in the consensus or communication with other shards, but is known by others. When any shard crashes, the corresponding shadow shards will be invoked by others, restoring the state to prevent the entire consortium member cluster from downtime.

Advantage. This hot backup approach helps the consortium member to improve its shard availability. It uses some acceptable redundancy in exchange for the high availability of the system. Moreover, for some applications that require extreme reliability, the shards can be equipped with more than one shadow shards to enhance availability and reliability.

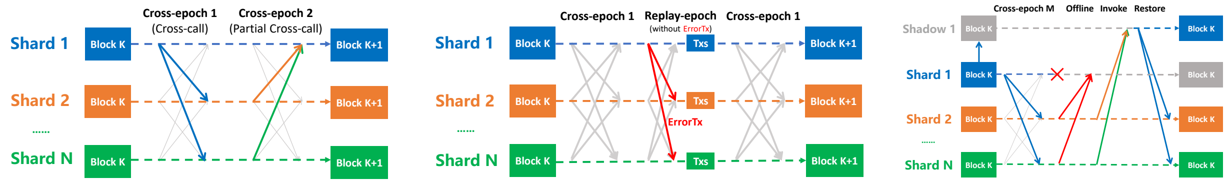
Security. Meepo does not change the consensus protocol. The fault tolerance of each shard is determined by the origin consensus protocol, such as PBFT [3]. Suppose there are *N* shards and *M* members in this network, $N \times M$ nodes in total. Suppose Meepo can tolerate d_m malicious members, there are $d_m \times N$ malicious nodes. Thus Meepo is resilient to Byzantine faults from up to $F_{meepo} = \frac{d_m \times N}{N \times M} = \frac{d_m}{M}$ fraction of nodes. Therefore, Meepo keeps the security the same as the original consensus protocol.

IV. Implementation and Evaluation

In this section, We implement Meepo as a consortium blockchain system to answer the following research questions: **RQ1:** How is the cross-shard efficiency in different shard scales? **RQ2:** How is the consumption of the replay-epoch, handling the transaction atomicity? **RQ3:** How is the availability using the shadow shard?

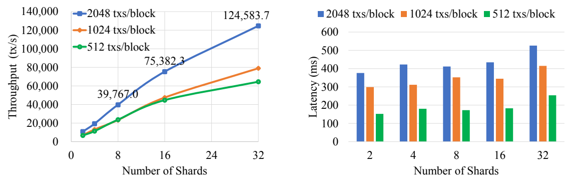
Implementation. We implement Meepo on the Go-Ethereum[2], with Proof of Authority (PoA) protocol that can be used for consortium blockchain. We add an interface for batch submission of transactions to facilitate testing and improve performance. We run the Meepo-Geth on the experimental environment on Elastic Cloud Server (ECS), provided by Al-iCloud. Each blockchain peer is equipped with an Intel Xeon Platinum 8269CY CPU, 8 cores, 32GB RAM, and 900GB NVME SSD. The other configurations of the benchmark will be introduced case by case in the experiments.

[2]https://github.com/Ethereum/go-ethereum



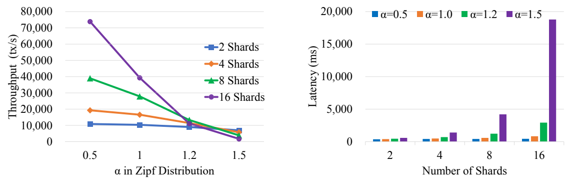
(a) Partial Cross-calls (b) Replay-epoch (c) Shadow Shard

Fig. 5. Cross-shard Communications of S2, S3, S4 in Meepo



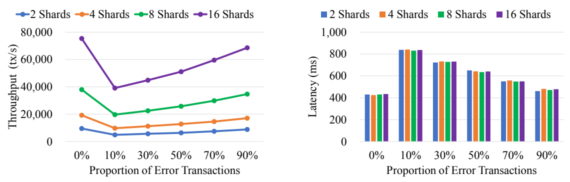
(a) Throughput (b) Latency

Fig. 6. Cross-shard Efficiency of Meepo-Geth in Different Number of Shards, under the Workload of 104,857,600 Transactions in Maximum



(a) Throughput (b) Latency

Fig. 7. Performance Evaluation of Meepo-Geth under the Workload of Zipf Distributed Account Transaction Frequency



(a) Throughput (b) Latency

Fig. 8. Performance Evaluation of Meepo-Geth with Different Proportions of Error Transactions, with 52,428,800 Transactions in Maximum

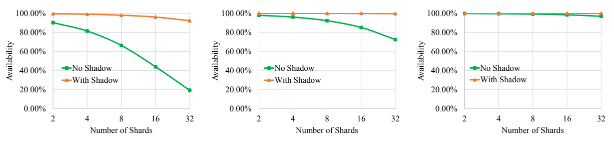
RQ1: Cross-shard Efficiency. To evaluate the cross-shard efficiency of Meepo, we use the mentioned *Asset Transferring Contract*. Since the consortium consensus protocol in Geth is PoA, of which the consensus efficiency is less affected by the number of consortium members, we use four consortium members, and each one is equipped with shards from 2 shards to 32 shards per member, case by case. And the results are shown in Figure 6. In the configuration of 32 shards, the throughput is 124,583.7 TPS. In this configuration, the number of processed cross-shard transactions is $32 \times 2048 \times 1600 = 104,857,600$ in total, transferring the assets among $2048 \times 1600 = 3,276,800$ accounts. As for the original Geth, we also test its performance in the same configuration, and the result is 6,343.2 TPS. Therefore, compared with the throughput of original Geth in the same number of consortium members and workload, Meepo-Geth delivers 20x throughput over the original consortium system. As for network consumption, if the consortium blockchain requires 124,583.7 TPS and the corresponding latency of 526.04 ms, the network bandwidth has to be able to transfer the 13 MB data in 0.5s. Figure 6 also shows that the number of transactions of the block also matters. When running with more blocks, the Garbage Collection (GC) in Golang becomes more significant, leading to a decrease in performance. GC optimization or using other languages (C++/Rust) could solve this problem in future work.

In the real-world asset transferring, the frequency distribution of real-world accounts shows a Long-tailed distribution [7]. We use $P(r) = \frac{C}{r^{\alpha}}$ to generate the Zipf [1] distributed workload, where r represents the rank of an account, P is the probability of the account, C is a constant number, and α is a parameter. The results are shown in Figure 7. When the α grows, the scalability of shards declines, as the throughput drops faster with more shards. This is the so-called hot-spot account problem, which might be solved by upper load balancing in future work.

RQ2: Consumption of Transaction Atomicity. To evaluate the consumption of handling the transaction atomicity, we run Meepo-Geth with some error transactions that will throw exceptions during the cross-epoch. During the workload generation, we insert the error transactions in different percentages of the total transactions. The percentage of errors is set to 0%, 10%, 30%, 50%, 70%, and 90%, respectively. The throughput and latency are shown in Figure 8.

As shown in Figure 8(a), with 10% transactions that throw the exceptions, Meepo-Geth achieves around 40,000 TPS in 16 shards. Compared to the throughput that is 75,382 TPS of all successful transactions, the throughput is reduced by 46%. However, it should be noted that this cost of time is worthy. In the asset transferring of consortium blockchain, even though the throughput might be reduced by half, the total throughput of the system is still high. And, the error transaction does not happen all the time. On the contrary, it can hardly ever happen in consortium blockchain because the upper application can check the transaction before submission. In this case, in a long time, the system will only have a very short time to reduce throughput due to error transactions.

RQ3: System Availability. We simulate the availability of consortium member in Meepo, in the different number of shards and different configurations of single peer reliability. Each shard is equipped with a shadow shard, and we suppose the switching time in the hot backup method can be ignored.



(a) λ=0.95 (b) λ=0.99 (c) λ=0.999

Fig. 9. Availability of Meepo Consortium Member

We mark the reliability of the single peer as λ , as mentioned before. In this evaluation, the configurations are set to $\lambda = 0.95$, $\lambda = 0.99$, and $\lambda = 0.999$, respectively. The results are shown in Figure 9. As shown in Figure 9(a), with the number of shards increases, the availability of the "No Shadow" decreases to less than 20%, which is an unacceptable level. With the shadow shard, the availability can be enhanced by more than 90%, of which the improvement is very significant. However, if the consortium member is using the cloud server with high reliability, it might have no needs to equip the shadow shard, as Figure 9(c) shows no significant difference between using the shadow shard or not.

V. Related Work

Luu et al. [9] firstly introduce the sharding technology in the database to the blockchain and propose Elastico. After that, more sharding strategies of blockchain are proposed. Kokoris-Kogias et al [8] proposed OmniLedger. OmniLedger proposes Atomix, a Byzantine shard atomic submission protocol, to ensure that each transaction is fully committed or eventually canceled. Zamani et al. [20] propose RapidChain with full sharding of blockchain transactions. Similar to the full sharding in RapidChain, Monoxide [18] provides the concept of asynchronous consensus zones and uses Chu-ko-nu mining to defend the attack from malicious peers. Dang et al. [5] propose a sharding method for permissioned blockchain with a 2PC (Two-phase Commit) method to solve the distributed transactions in the sharded blockchain. Rana et al. [13] propose Free2Shard, using a dynamic node allocation algorithm to enhance the sharding security. Besides the sharding approaches proposed by academia, there are also other sharding studies that are from cryptocurrency communities, such as Zilliqa [16] and Ethereum 2.0 [2]. From the above, we can learn that most sharding studies are concentrated on public blockchain, with less attention to consortium blockchain, and a single participant only keeps part of the ledger. Because of the delay of propagation, cross-shard communication is usually across the blocks or done by the clients. The significant difference between these studies and Meepo is that, in Meepo, consortium members always hold all the shards they need, thus cross-shard communication can be efficient in the local network of the consortium members.

VI. Conclusion

In this paper, we propose Meepo, a systematic study on the sharded consortium blockchain systems. It uses a cross-epoch based protocol to achieve high efficient cross-shard communication. Furthermore, we provide a partial cross-call merging strategy to support the multi-state calling of complex smart contracts. We also preserve the strict transaction atomicity as the original blockchain via a replay-epoch based cross-shard protocol. Additionally, we add shadow shards as the backup for the shards to prevent the system from being blocked by a single point of failure. The maximum measured cross-shard throughput is more than 120,000 TPS.

Acknowledgment

The research is supported by the National Key R&D Program of China (No. 2020YFB1006001), the National Natural Science Foundation of China (No. 62032025), and Alibaba Group through the Alibaba Innovative Research (AIR) programme. Zibin Zheng is the corresponding author.

References

[1] R. L. Axtell. Zipf distribution of us firm sizes. *science*, 293(5536):1818–1820, 2001.
[2] V. Buterin. Ethereum 2.0 mauve paper. 2016.
[3] M. Castro, B. Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.
[4] W. Chen, Z. Zheng, J. Cui, E. Ngai, P. Zheng, and Y. Zhou. Detecting ponzi schemes on ethereum: Towards healthier blockchain technology. In *WWW*, pages 1409–1418, 2018.
[5] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi. Towards scaling blockchain systems via sharding. In *SIGMOD*, pages 123–140, 2019.
[6] T. T. A. Dinh, J. Wang, G. Chen, R. Liu, B. C. Ooi, and K.-L. Tan. Blockbench: A framework for analyzing private blockchains. In *SIGMOD 2017*, pages 1085–1100, 2017.
[7] A. B. Downey. Evidence for long-tailed distributions in the internet. In *Proceedings of the 1st ACM SIGCOMM Workshop on Internet Measurement*, pages 229–241, 2001.
[8] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford. Omniledger: A secure, scale-out, decentralized ledger via sharding. In *2018 IEEE Symposium on SP*, pages 583–598. IEEE, 2018.
[9] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena. A secure sharding protocol for open blockchains. In *Proceedings of CCS*, pages 17–30. ACM, 2016.
[10] S. Mehrotra, R. Rastogi, Y. Breitbart, H. F. Korth, and A. Silberschatz. Ensuring transaction atomicity in multidatabase systems. In *Proceedings of 17th ACM SIGACT-SIGMOD*, pages 164–175, 1992.
[11] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.
[12] M. T. Özsu and P. Valduriez. *Principles of distributed database systems*, volume 2. Springer, 1999.
[13] R. Rana, S. Kannan, D. Tse, and P. Viswanath. Free2shard: Adaptive-adversary-resistant sharding via dynamic self allocation. *arXiv preprint arXiv:2005.09610*, 2020.
[14] N. Szabo. The idea of smart contracts. 1997.
[15] Y. Tao, B. Li, J. Jiang, H. C. Ng, C. Wang, and B. Li. On sharding open blockchains with smart contracts. In *ICDE*, pages 1357–1368. IEEE, 2020.
[16] Z. Team et al. The zilliqa technical whitepaper. 2017.
[17] A. Thomson, T. Diamond, S.-C. Weng, K. Ren, P. Shao, and D. J. Abadi. Calvin: fast distributed transactions for partitioned database systems. In *Proceedings of SIGMOD*, pages 1–12, 2012.
[18] J. Wang and H. Wang. Monoxide: Scale out blockchains with asynchronous consensus zones. In *16th NSDI*, pages 95–112, 2019.
[19] G. Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum Project Yellow Paper*, 151, 2014.
[20] M. Zamani, M. Movahedi, and M. Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of CCS*, pages 931–948. ACM, 2018.
[21] P. Zheng, Z. Zheng, J. Wu, and H.-n. Dai. Xblock-eth: Extracting and exploring blockchain data from ethereum. *IEEE Open Journal of the Computer Society*, 1:95–106, 2020.
[22] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang. An overview of blockchain technology: Architecture, consensus, and future trends. In *IEEE International Congress on Big Data*, pages 557–564. IEEE, 2017.