Aeolus: Distributed Execution of Permissioned Blockchain Transactions via State Sharding

Peilin Zheng, *Member, IEEE*, Quanqing Xu, *Senior Member, IEEE*, Xiapu Luo, *Member, IEEE*, Zibin Zheng, *Senior Member, IEEE*, Weilin Zheng, Xu Chen, Zhiyuan Zhou, Ying Yan, and Hui Zhang, *Senior Member, IEEE*

***Abstract*—Blockchain has attracted lots of attention in recent years. However, the performance of blockchain cannot meet the requirement of massive Internet of Things (IoT) devices. One of the important bottlenecks of blockchain is the limited computing resources on a single server while executing transactions. To address this issue, we propose Aeolus blockchain to achieve the distributed execution of blockchain transactions. There are two key challenges to achieving this for IoT blockchain: transaction structure and state consistency. Facing these challenges, we first propose a distributed blockchain transaction structure, which imports extra parameters to divide the transaction execution into different stages to enable distributed execution. Second, we propose distributed state update sharding, which equips each blockchain peer with its own master and shard servers. In this way, each blockchain peer can be considered as a cluster that distributes the transaction to shorten the processing time and reach the consensus finally. We implement Aeolus on Go-Ethereum to evaluate its feasibility, on a testbed including 132 cloud servers. Our system runs stably for more than 8 h under the workload of 190 000 000 real-world user transactions. Experimental results show the efficiency that Aeolus can achieve more than 100 000 transactions/s of blockchain transactions, which is 15.6 times the throughput of the original blockchain.**

***Index Terms*—Distributed computing, parallel processing.**

Manuscript received 31 December 2021; revised 1 March 2022 and 16 March 2022; accepted 23 March 2022. Date of publication 4 April 2022; date of current version 30 September 2022. This work was supported in part by the National Key R&D Program of China under Grant 2020YFB1006005, in part by the National Natural Science Foundation of China under Grant 62032025, in part by the Alibaba Group through the Alibaba Innovative Research Program, in part by the Hong Kong Innovation and Technology Fund Project under Grant GHP/052/19SZ, and in part by the Hong Kong Research Grants Council Project under Grant PolyU15222320. Paper no. TII-21-5989. *(Corresponding author: Zibin Zheng.)*

Peilin Zheng, Zibin Zheng, Weilin Zheng, and Xu Chen are with the School of Computer Science and Engineering, Sun Yat-Sen University, Guangzhou 510275, China (e-mail: zhengpl3@mail2.sysu.edu.cn; zibinzheng2@yeah.net; zhengwlin@mail2.sysu.edu.cn; chenx397@mail2.sysu.edu.cn).

Quanqing Xu, Zhiyuan Zhou, Ying Yan, and Hui Zhang are with the Blockchain Platform Division, Ant Group, Hangzhou 310000, China (e-mail: xuquanqing@gmail.com; wenzhang.zzy@antgroup.com; fuying.yy@antgroup.com; huizhang@nec-labs.com).

Xiapu Luo is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong (e-mail: csxluo@comp.polyu.edu.hk).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TII.2022.3164433.

Digital Object Identifier 10.1109/TII.2022.3164433

I. Introduction

Blockchain can enhance the interoperability of the industrial Internet of Things (IoT) devices [1], [2] from different manufacturers. However, the throughput of existing blockchain systems is too low to process the transactions from massive IoT devices. Matured IoT platforms need high throughput. For example, the AliCloud IoT Platform [3] allows millions of devices to communicate with the IoT platform at the same time. However, current blockchain systems cannot handle such a high workload. Improving blockchain performance becomes an urgent research problem, and it is also the key motivation of this article.

Recent studies [4] have pointed out that the major bottlenecks of blockchain systems lie in *network delay* and *transaction execution*. Existing blockchain systems can be divided into permissionless blockchain and permissioned blockchain according to the permission of peers. Permissionless blockchain systems (e.g., Bitcoin [5] and Ethereum [6]) are usually running on top of unstable networks with low bandwidth and low throughput due to low entry barriers for validating peers. Hence, network delay is their major bottleneck because the block and transaction propagation are slow. In contrast, permissioned blockchain systems (e.g., Hyperledger Fabric [7]) are usually deployed on the network with less number of peers and good network conditions. Thus, their performance bottleneck is the execution of transactions, which is the target bottleneck in this article, evaluated in Section VI-C.

By now, earlier studies have been proposed in sharding to improve blockchain performance, e.g., Elastico [8], Zilliqa [9], OmniLedger [10], Monoxide [11], Rapidchain [12], etc. They divide the network into different shards, where the peer validates only partial transactions. These solutions can be summarized as "network sharding." They break the bottleneck of network delay, especially for the permissionless blockchain. However, they cannot be applied to the permissioned blockchain due to the following reasons: 1) *low data availability and security:* permissioned blockchain requires higher data availability and security than permissionless blockchain, because the data are frequently read and written for the business. However, the aforementioned solutions usually let the peer hold partial data of

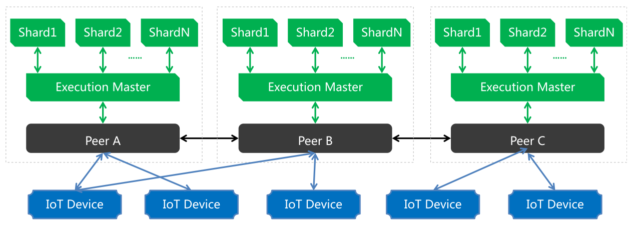

Fig. 1. Architecture of Aeolus blockchain with IoT devices.

the blockchain, and thus, the data availability and security may be jeopardized; and 2) *low cross-shard efficiency:* a transaction in sharded blockchain may depend on more than one shards; then, it is called a cross-shard transaction. When processing cross-shard transactions, the blockchain needs to spend more time in cross-shard communication. Generally, the cross-shard efficiency reflects on the throughput and latency. The aforementioned solutions usually sacrifice cross-shard efficiency in exchange for security. However, the permissioned blockchain has better promise for security and needs lower latency of cross-shard transactions.

In summary, existing solutions were designed for the blockchains with low network bandwidth. Exploiting better network conditions (e.g., higher bandwidth and lower delay), a new algorithm is needed to enhance the performance of blockchain systems. In this article, we focus on breaking the bottleneck of the transaction execution instead of network delay for permissioned blockchains.

Existing blockchain systems usually execute the transactions on one server, and thus, the limited computing resources become the bottleneck of execution [4]. An intuitive approach to accelerate the transaction execution is extending one single server to a cluster, which is the main idea of this article. However, toward this main idea, we need to address two challenges.

1) *C1: Transaction structure:* The transactions relevant to smart contracts, which are autonomous programs running on top of blockchains, cannot be executed in a distributed way. Although some research (e.g., ParBlockchain [13] and FISCO-BCOS [14]) tries to achieve parallel execution, the limited disk I/O on a single server still restricts the performance. Hence, a new transaction structure for distributed execution is needed.
2) *C2: State consistency:* The distributed execution of blockchain transactions makes it hard to deliver messages among the servers of one cluster and maintain consistency across different clusters.

Motivated by the above challenges, we propose *Aeolus* (Aeolus symbolizes fast transaction execution), which achieves the distributed execution of blockchain transactions through state sharding. Fig. 1 shows the architecture of the Aeolus blockchain network. Different IoT devices connect to a permissioned blockchain peer, which handles the consensus protocol computation with other peers. Each peer uses its equipped cluster to execute (validate) the transactions.

Based on this architecture, Aeolus proposes the following solutions to address the mentioned challenges.

1) *S1: Distributed blockchain transaction structure* is a new structure of blockchain transactions. The main idea is to import extra parameters to divide the smart contract execution into different stages according to different shards. The stages on the same shard will be executed sequentially for maintaining the order and consistency of blockchain transactions. Furthermore, the stages of different shards have no conflicts with each other so that they can be executed simultaneously. This solution is different from traditional distributed database transactions. The significant difference is that blockchain transaction is used for triggering complex smart contracts, but database transaction is used for operating tables.
2) *S2: Distributed state update sharding* is an algorithm to divide the state into different shards and update them by a “Master-Shards” architecture. The master handles the block validation and then distributes the transactions to different shard servers for execution. Different shard servers deliver the messages through the master for the cross-shard transactions. Finally, the master fetches the execution result (state root, which is similar to traditional blockchain systems such as Ethereum and Fabric), which is compared with other blockchain clusters to keep the consistency.

We emphasize that Aeolus changes the transaction structure but not the block structure. In blockchain systems, many popular consensus protocols (e.g., proof-of-work (PoW) [5], proof-of-stake (PoS) [15], proof-of-authority (PoA) [16], practical byzantine fault tolerance (PBFT) [17], etc.) reach the consensus of blocks by verifying the headers. The header verification is related to the block structure rather than the transaction structure; thus, it is decoupled with the transaction execution. Therefore, Aeolus can be adapted with existing consensus protocols without changing the security assumptions and fault tolerance of peers.

We implement Aeolus based on Go-Ethereum (Aeolus-Geth) and carefully evaluate its performance as well as compare it with other blockchains. The experimental results show that Aeolus gets the performance improvement. On a testbed including 132 AliCloud servers, our system can deliver more than 100 000 transactions/s (TPS) using 32 shards per peer. Aeolus-Geth runs stably for more than 8 h, under the benchmark workloads from real-world Ethereum transactions, including more than 190 000 000 ERC20 transactions.

The major contributions of this article include the following:

1) We propose Aeolus to accelerate blockchain transaction execution via a distributed architecture, motivated by the evaluated performance bottleneck of transaction execution in the permissioned blockchain.
2) We propose a new distributed blockchain transaction structure that enables the transaction to be executed on multiple servers to accelerate the transaction execution.
3) We propose a novel distributed state update sharding algorithm as a “Master-Shards” architecture, which makes full use of the resources on shard servers and reaches consensus in the blockchain network.
4) We implement the new solutions in Aeolus and conduct extensive experiments to evaluate them. The experimental

results show that Aeolus outperforms existing platforms, and it can achieve more than 100 000 TPS in 32 shards.

The rest of this article is organized as follows. Section II describes the background of blockchain and smart contracts. Section III proposes an overview of Aeolus. Section IV introduces the detailed design of the distributed blockchain transaction structure. Section V proposes the mechanism of distributed state update sharding. Section VI shows the implementation of Aeolus and the comparison between Aeolus and common blockchain systems. Sections VII and VIII discuss the related work of scaling the blockchain systems. Finally, Section IX concludes this article.

II. Background

Blockchain is first proposed by Bitcoin [5]. In a peer-to-peer network, a peer packs the transactions into a block in a period. This chain-like data structure is called the blockchain. The protocol that determines which peer to generate the block is called consensus protocol. Since the peers execute the transactions in their local computing environment, the results can be trusted by each peer. In this article, we focus on improving the transaction execution, which is independent of the consensus protocol. Hence, the fault tolerance of the original blockchain system is not changed.

Smart contracts are autonomous programs [18] running on top of blockchains, which define the operation of the states [19]. All peers in the same blockchain will execute the smart contracts in the same way. Ethereum and Hyperledger Fabric are the typical blockchain systems that support smart contracts. All the peers in a blockchain will execute the smart contracts in the same way and reach the same final state. In one block, different transactions may operate the same state. Thus, it is hard to execute the transaction in a parallel and distributed way. To address this problem, a change of transaction structure and state operation is needed, which is proposed in this article.

Permissionless versus permissioned: Blockchain can be divided into permissionless blockchain and permissioned blockchain according to the permission. The permissionless blockchain allows any peers to join it to keep its decentralization. Thus, a permissionless blockchain is usually developed with a low entry barrier of hardware so that more peers can run it (e.g., Bitcoin and Ethereum). The permissioned blockchain requires that peers in the network should have permission. Previous research has pointed out that the major performance bottleneck of the permissionless blockchain is the network situation [4]. However, the network of permissioned blockchain is usually stable with low delay and large bandwidth, and thus, its performance bottleneck often lies in the time cost of executing transactions. We aim at breaking this bottleneck in this article.

III. System Overview

This section introduces Aeolus from an overview perspective. The details will be described in Sections IV and V.

Architecture: The architecture of Aeolus is shown in Fig. 1. As mentioned in Section I, in order to improve the computing resources during transaction execution, Aeolus extends the single blockchain peer into a cluster. The main idea is to make full use of the computing resources in the cluster, and then, the time of transaction execution will be shortened. Hence, it improves the performance of blockchain. In the following, we will propose the roles in the topology in Fig. 1 and, then, give an example of transaction execution flow in Aeolus.

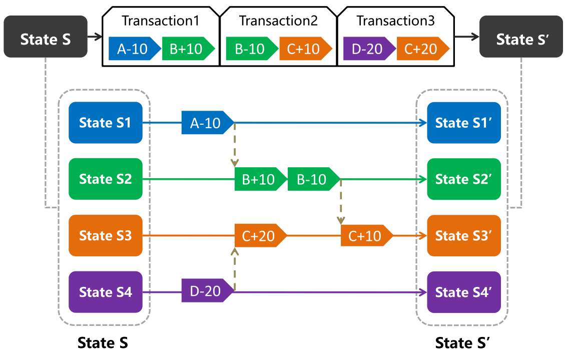

Fig. 2. Transaction execution flow in Aeolus.

Roles: Traditional blockchain systems take one server as a peer. In Aeolus, each *Peer* is equipped with an *Execution Master* and several *Shard Servers*.

As shown in Fig. 1, all these roles are servers, but have different functions as follows. *Peer* is still the node as it is in the traditional blockchain, which commits the block, validates the block, and joins into the blockchain consensus with other peers in the network. However, the transaction execution environment is not on this server, but on the *Execution Master* and *Shard Servers*. *Execution Master* can be considered as a router for the blockchain transactions among the *Shard Servers* to execute the transactions and return the execution result to the *Peer*. *Shard Server* is the exact executor of blockchain transactions. The state is divided into different shards that are handled by different shard servers. Each shard server executes the transactions that operate the shard state on it. As the green area in Fig. 1 shows, each peer has a local execution environment of blockchain transactions, which is only served to themselves independently. Moreover, the green area is only used for transaction execution (validation), where future work can be done to make it join into the consensus protocol (e.g., computing hash code for PoW). In this way, each cluster can be regarded as a blockchain peer, and the increased complexity does not break the equivalence of blockchain peers. Each blockchain peer can reach consensus as before.

Transaction execution flow: Based on the above architecture, the transactions in one block can be executed. Fig. 2 shows an example of the transaction execution flow on Aeolus. In this figure, *State S* is changed by several transactions one by one. The areas of different colors represent different stages of the transaction, and different stages operate on different state shards. Taking a specific example in Fig. 2, *State S* is divided into four state shards. The accounts `A`, `B`, `C`, and `D` are, respectively, on the state `S1`, `S2`, `S3`, and `S4`. *Transaction1* is a transaction that transfers \$10 from `A` to `B`; thus, it needs to deduct the balance of `A` (`A-10`) and then add it to `B` (`B+10`). And so do Transaction2 and Transaction3. In one transaction, different stages should follow the order in the transaction. Nevertheless, in different transactions, different stages (shown indifferent colors) operating different states are mutually exclusive. Thus, they can be executed in parallel.

In this example, the traditional blockchain will execute six state operations one by one, shown as the upper part of the figure. But Aeolus only executes them as follows. First, `S1` and `S4` execute the `(A-10)` and `(D-20)` in parallel. These two operations result in the next step. Second, `S2` and `S3` execute the `(B+10)` and `(C+20)` in parallel. After that, `S2` and `S3` execute the `(B-10)` and `(C+10)` in serial. In total, despite the communication time, only four steps are needed for six operations. Therefore, the execution time can be shortened. As for the dependence between Transaction1 and Transaction2, in a permissioned blockchain, it can be handled by the sender (developer) through existing concurrency control studies [20], [21], which is not the major contribution of Aeolus. Future work could be done for handling dependence in the master.

Based on this architecture, there are two key technical challenges.

1) *C1: How to divide the smart contract execution into stages by transaction structure?* On the permissioned blockchain, the applications are always based on smart contracts. Different from the simple payment transaction, it is difficult for the master to determine which stage and which shard for the smart contract transactions since the master does not execute transactions. The *cross-shard transactions* in previous studies cannot be applied to Aeolus, since they need to change the consensus protocol, discussed in Section VII. Our solution to this challenge is the *distributed blockchain transaction structure* (see Section IV).
2) *C2: How to divide the state and keep the consistency among different clusters?* Aeolus expands one blockchain peer into a cluster. First, it is a problem to determine how to divide the state into shards. Second, it is important to ensure that the transaction execution sequence is consistent among clusters. However, the network and computing resources in different clusters can be different. Therefore, it is challenging to keep the consistency of state in different clusters with high execution efficiency. Our solution to this challenge is the *distributed state update sharding* (see Section V).

IV. Distributed Blockchain Transaction Structure

From Section III, we learn that, after dividing the state into different shards, we need to execute the transaction on the corresponding shards. Moreover, the transaction atomicity should also be handled. Thus, the structure of blockchain transactions should be improved. In Aeolus, the transaction can be represented as a tuple of *<PubKey(from), Contract(to), Input, Signature, Shard, Stage>*. In this structure, *<PubKey(from), Contract(to), Input, Signature>* is inherited from the traditional blockchain, such as Ethereum and Fabric. Aeolus improves the blockchain transaction structure by additional parameters: *Shard* and *Stage*.

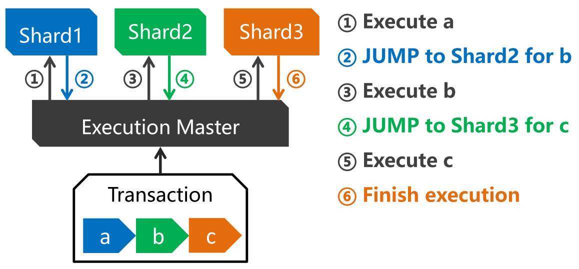

Fig. 3. Cross-shard message delivery By JUMP operation.

A. Shard Flag and Stage Flag

In Aeolus, transactions operating different state shards are executed on different shard servers, as shown in Fig. 2. *Shard Flag* is used to show which shard the transaction should be executed on. The execution of a transaction is divided into different stages. Each stage is marked by a flag. That is the *Stage Flag*. The flag is used to represent which stage the transaction is in. The operation in one stage is fully executed on one shard server. Hence, one stage consists of the state operation only on one shard server. Moreover, as defined in the smart contract, different stages can be executed on the same shard, if needed. The reason to set *Shard* and *Stage* is that, in this way, the execution master can deliver the transaction to the exact shard server without any execution or complex parsing. From the perspective of users and smart contract developers, the differences between Aeolus and traditional blockchain transactions are twofold.

1) The smart contract developer should use *Stage Flag* to divide the contract function. The example of Aeolus smart contracts in handling cross-shard function and atomicity will be given in the next subsections.
2) The transaction sender is required to indicate the first shard server that the transaction will be delivered by setting the *Shard* field in the transaction. After the execution on the first shard server, the transaction could be delivered to other exact shard servers by a "JUMP" operation, which will be introduced in the next subsection.

We claim that it is easy for the developer to set a correct *Shard Flag* while sending a transaction. In Section V-A, an internal shard mapping function will be described. The developer can use the function to generate a *Shard Flag* corresponding to the parameters of the calling contract.

B. JUMP Operation

The *JUMP* operation is given for the transaction to be executed in different shard servers. As shown in Fig. 3, there is a transaction that is divided into three stages. First, the transaction is delivered to *Shard1* to execute the *a* stage. The execution result of *a* will generate a message for the master to forward to *Shard2* to execute *b*. And so do *b* and *c*. Note that Fig. 3 shows the sequential manner of stages for only one transaction. Suppose that there are more transactions; the stages in different transactions (e.g., the *a* in transaction X and the other *a* in transaction Y) can be executed in parallel. Hence, the execution can be accelerated.

Algorithm 1: Example of Cross-Shard Contract Function.

```
1 function Transfer (from, to, value) {
2     Stage 0:
3         if (balanceOf[from]<value) return;
4         balanceOf[from] -= value;
5         JUMP(SHARD_OF(to), 1);
6     Stage 1:
7         balanceOf[to] += value;
8 }
```

The JUMP message can be represented as a tuple of <*OriginTx, Shard, Stage*>, where *OriginTx* is the origin transaction. During the execution of a block, the JUMP messages of different transactions can be packed together. In other words, the execution of a block is divided into different rounds. Each round consists of the stages of different transactions. The cross-shard message delivery is done between the rounds. A cross-shard message includes many JUMP messages so that the internal communication of the cluster is reduced to a minimum. Sincerely, JUMP operation increases the complexity of smart contract development. However, we claim that this increase is acceptable and worthy, as the developer just needs to divide the function into stages and combine them through the JUMP operation.

In Aeolus, the JUMP operation is a built-in function preset in the contract execution environment. It requires two input parameters: the shard index and the stage. Algorithm 1 shows a typical example of a cross-shard smart contract function. This function is used to transfer assets (or the so-called tokens) between accounts (*from* and *to*). Before executing this function, the transaction sender should set the *Shard Flag* as the index of the shard server that handles the state of *from*. This function has two stages, and it defaults to begin at *Stage0*. In this stage, the balance of *from* is checked and reduced by *value*. Then, it "jumps" to the shard server, which increases the balance of *to* on it. *SHARD_OF(to)* is another function that returns the corresponding index of the shard server, which will be described in Section V. Thus, the cross-shard function of smart contracts can be executed in different shard servers.

C. Atomicity

In traditional blockchain systems (e.g., Ethereum and Fabric), the transaction only has two statuses: success or failure. Once the transaction is failed for any exceptions of the smart contract, all the operations would be given up that the state will not be changed. This characteristic is called *Atomicity* [22] in database systems. Since the transaction is executed in different shards, it is hard to ensure the atomicity across different shards. In previous blockchain sharding research, Elastico, OmniLedger, RapidChain, and Monoxide use different strategies to ensure the atomicity of blockchain transactions. The limitation is that their strategies have to match their consensus protocol. Moreover, their consensus protocols will cause new security problems (discussed in Section VII). In Aeolus, the goal is not to change the block structure and consensus protocol to keep the security, so that the atomicity should be ensured by the blockchain transaction itself. To achieve the atomicity of transactions, the key is that the change of the state in different shards should be reverted when the execution fails. In the distributed blockchain transaction structure, one of the possible methods is to set the master to record the change in every shard so that it can revert the change. However, in this way, the master will afford a much larger workload than the shard servers, which results in the performance bottleneck. In this article, Aeolus provides two optional methods to revert the state in failed transactions to ensure the transaction atomicity: one is Rollback Stage and the other is *Block Atomicity*. The *Rollback Stage* is similar to the exception handling function in the traditional programs. In those stages that might cause the exception, which results in the execution failure, Aeolus requires the contract developer to define the *Rollback Stage*, which consists of the operation to revert the state.

Algorithm 2: Pseudocode of Distributed State Update Sharding Algorithm.

```
Input: Blockchain transactions $txs$, Shards $shards$
Output: World State $stateRoot$
1  while $txs \neq []$ do
2      /* Round begins */
3      for $s$ in $shards$ do
4          s $\xleftarrow{send}$ txs[s]
5      end
6      stateRoot = init()
7      /* Asynchronous*/
8      for $s$ in $shards$ do
9          packet $\xleftarrow{get}$ s
10         JUMPS[s] $\leftarrow packet.JUMPS$
11         $stateRoot = stateRoot \oplus packet.stateRoot$
12     end
13     txs $\xleftarrow{update}$ JUMPS
14 end
15 return stateRoot
```

Ideally, most blockchain transactions should be executed successfully since most exceptions could be avoided by checking the contract state before execution, i.e., using code like "if-else." However, in practical applications, there could be few exceptions that could not be checked before execution. Thus, the rollback stage is necessary to keep the atomicity of the blockchain transaction. In practical permissioned blockchain systems, the exceptions are rare. The reason is that, the peers and contracts in the permissioned blockchain systems are stable, and all the contract code has been carefully checked and tested. Under this premise, *Block Atomicity* can be considered. The method is that, once the *Execution Master* finds any exception of transactions thrown, the master will revert all the state on the shard server. In other words, any exception in the transactions of a block will make all the transaction execution of the block fail. This is a one-size-fits-all approach. However, there will be few failed blocks if there are few failed transactions.

V. Distributed State Update Sharding

This section will propose how Aeolus distributes the state onto different shard servers and keeps the consistency of the blockchain in different clusters. The pseudocode of distributed state update sharding algorithm is shown in Algorithm 2, explained in the following subsections. It accepts the blockchain transactions and shards as an input and then comes out with the world state, which is represented as the "stateRoot."

A. Sharding Strategy

In Aeolus, the state is distributed into shards, which are held on different shard servers. The aim of the sharding strategy is to determine which shard should be operated when create or update a state record (e.g., a key–value pair). In Aeolus, we argue that the sharding strategy of the state of blockchain should meet the following requirement.

1) The sharding strategy needs to make the state evenly distributed across all the shard servers. This is related to the load balancing of shard servers. Consider an extreme case, and if the sharding strategy makes the state spread across only a few shard servers, then the transaction will be all executed on those servers, resulting in very low resource utilization.
2) The sharding strategy should be dynamic for different data types of input parameters. For those blockchain systems with a single function that is only conducted for transferring cryptocurrencies (e.g., Bitcoin), the sharding strategy could be static. For example, the strategy could be calculating the first *k* bits of the sender's address as the index of shard [11]. However, this kind of strategy does not work for the state of the smart contract because the input parameters could be all kinds of data types (e.g., char and string). Therefore, in blockchain-based smart contracts, the sharding strategy of the state should be dynamic for different data types.
3) The sharding strategy should be able to be flexibly controlled by the smart contract developer. In practical blockchain application scenarios, developers need to perform batch state queries based on actual factors, such as business type.

If the developer can determine which shard server that data are located in, the query efficiency will be improved since only a few shard servers are needed. In other words, the sharding strategy is not entirely transparent to the smart contract developer. However, it can also be autonomous through simple programming such as using a hash function. With the above concerns, Aeolus leaves the *Shard Flag* in the structure of the transaction and the JUMP operation for the developer to set, so that requirement 3 could be satisfied. Aeolus provides an internal shard mapping function as (1). In this equation, *arg* is the given parameter, which needs to be mapped to one shard, such as the account. *HASH(arg)* is a hash function (e.g., MD5, SHA256) that could be chosen by the blockchain manager to give the hash value of an argument (arg). *n* is the number of shard servers in the cluster. In this way, Aeolus meets requirements 1 and 2. Note that even though the state (e.g., accounts) is distributed evenly, the transactions might not always even to each shard since some accounts would send more transactions than others. This is called a *Hot-spot Account* problem in blockchain sharding. The impact of such unevenness will be evaluated in Section VI-F

$$SHARD\_OF(\text{arg}) = HASH(\text{arg})\%n. \tag{1}$$

This sharding strategy is used in lines 3–5 in Algorithm 2.

B. Round-Based Distributed State Update

From Section IV-B, we can learn that the execution of a block is divided into different rounds. Each round consists of the stages of different transactions. The "JUMP" messages are packed and delivered between rounds. In the first round, the transactions are distributed to the corresponding shard servers. After that, the remaining stages of transactions will be delivered back to the master. The delivered messages can be represented as the following equation:

$$JUMPS_r(i,:) = Round_r(i).result. \tag{2}$$

As shown in (2) and (3), $JUMPS_r(i,j)$ represents the cross-shard message in round *r* from shard *i* to shard *j*. $Round_r(i)$ represents the set of transactions that are executed on Shard *i* in round *r*. In this round, $JUMPS_r(i,j)$ can be considered as the execution result of the transactions in this round. After that, the master needs to determine what to execute in which shard server in the next round. The messages that the master sends to the shard server can be represented as

$$Round_{r+1}(s) = \bigcup_{i=1,2,\ldots n, j=s} JUMPS_r(i,j). \tag{3}$$

In this equation, $Round_{r+1}(s)$ represents the set of transactions that are executed on Shard s in round $r+1$. It can be calculated as the union of the JUMP messages that "jump" to Shard s. In this way, the overhead of the network of each round can be reduced to a minimum. After all the execution, the traditional blockchain system uses a string called state root to represent the status of the whole state. The state root is recorded in the block in order to check whether the execution result is the same as other blockchain peers. In Aeolus, the state root is reserved as *multistate root*, which can be calculated as follows:

$$\begin{aligned} mulStateRoot = {} & stateRoot_1 \oplus stateRoot_2 \oplus \cdots \\ & \oplus stateRoot_n \end{aligned} \tag{4}$$

where *mulStateRoot* represents the multistate root of the state that could be calculated as the result of a continuous XOR calculation of all the stateRoot of shard servers. The reason for using XOR instead of continuous hashing or other methods is that XOR is not related to the order of operations. In multithread communication within the cluster, the order of messages from the shards to the master can be different. Therefore, by using XOR, the master can process the returned messages as quickly as possible, without waiting for all shards to return. This round-based distributed state strategy is shown in lines 8–12 in Algorithm 2.

In summary, with the same sharding strategy and round-based distributed state update approach, different clusters can do the same execution of the block and check the result with other clusters. In this way, the consistency of blockchain can be kept in Aeolus.

VI. Evaluation

For practical usage and evaluation, we implement Aeolus to answer the following research questions:

1) *RQ1:* Is transaction execution the bottleneck of current blockchain systems in permissioned peers?
2) *RQ2:* How is the scalability of Aeolus in different numbers of shards?
3) *RQ3:* How is the performance of Aeolus compared to other existing blockchain systems?
4) *RQ4:* How is the performance of Aeolus according to the different configurations of transaction workload?

These research questions are answered in the corresponding subsections as follows.

A. Implementation

We develop Aeolus to check the feasibility, called Aeolus-Geth, as it is based on Go-Ethereum [16], one of the popular Ethereum clients. Based on Go-Ethereum, Aeolus-Geth mainly modifies the modules of Ethereum virtual machine (EVM), StateDB, and Worker, with about 3500 lines of codes (LoC) above that of Geth's codebase. Although Ethereum is mainly designed as a permissionless blockchain, its client provides both the permissioned and permissionless protocols. As for permissioned scenarios, Geth has embedded a PoA consensus protocol called "Clique" for permissioned blockchain, which is also used in Aeolus-Geth.

B. Experimental Setup and Workloads

We run Aeolus-Geth on AliCloud. Each server is equipped with an Intel Xeon 8269CY CPU, four CPU cores, 32 GB of RAM, 1.8-TB solid-state drive. The number of servers is case by case in the experiment. In maximum, we use four permissioned peers, and each of them consists of 32 shards. Thus, the testbed includes $(32+1)\times 4 = 132$ servers in maximum. The other blockchain systems are also run on the same configuration machine. Unless otherwise specified, each experiment is conducted in a permissioned blockchain consisting of four peers.

Workloads: We use the public Ethereum dataset [23] to extract real-world transactions. It consists of more than 190 000 000 ERC20 transactions among 37 397 908 addresses, extracted from Ethereum #0 to #10 000 000 blocks. The same transactions are organized into two workloads: ERC20-Balanced and ERC20-Unbalanced. The ERC20-Balanced is balanced for each shard that the address distribution is even. The ERC20-Unbalanced is unbalanced that the address distribution is uneven. Monoxide [11] uses the same ERC20 workload to evaluate real-world asset transferring. Aeolus processes the transfer functions in these workloads as fully cross-shard, with the pseudocode shown in previous sections.

C. RQ1: Bottleneck of Current Blockchain

To verify the motivation of Aeolus, we first evaluate the performance of current blockchain systems. As well known, the bottleneck of the blockchains lies in the consensus protocol in the permissionless blockchain. However, as for permissioned

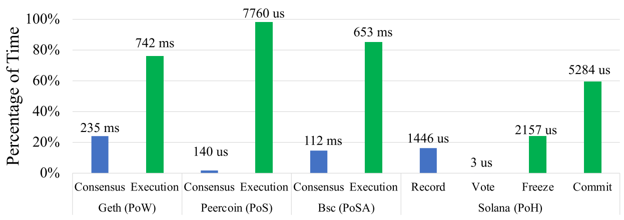

Fig. 4. Performance bottleneck of current blockchain with different consensus protocols (Geth, Peercoin, Bsc, and Solana).

blockchain, an empirical experiment is needed to show the performance of the transaction execution in existing platforms. We run four famous blockchain systems (Geth [16], Peercoin [15], Binance Smart Chain (Bsc) [24], and Solana [25]) with different consensus protocols [PoW, PoS, proof-of-staked-authority (PoSA), and proof-of-history (PoH)] as a private chain to see their performance in the permissioned blockchain. We instrument the source code of clients to measure the cost distributions of transaction executions and consensus protocol. The cost is measured by the average time of consensus and execution in each block. Note that the threat to the validity of this experiment is that different systems could show different performances under different configurations (e.g., difficulty in PoW and authorities in PoSA) and workloads.

The results are shown in Fig. 4. As for Geth and Bsc, they both run the EVM smart contract and spend more time in execution than consensus. The difference between them is that PoW consumes more computing resources than PoSA, thus taking more time in consensus. Peercoin (PoS) processes the Bitcoin-like transactions, which are faster than EVM contracts, but its PoS still shows faster than execution in the permissioned peers. As for Solana, its consensus protocol [26] consists of PoH (Record) and tower byzantine fault tolerance (TBFT) (Vote) and its execution consists of Freeze and Commit. Fig. 4 shows that the green part is greater than the blue part, indicating that the percentage of time in execution is larger than that in consensus.

In summary, in a permissioned blockchain with good network conditions for consensus protocol, the transaction execution is the bottleneck of current systems, as it occupies most of the block time on average.

D. RQ2: Scalability of Aeolus

The idea of Aeolus is promising to improve the scalability of the blockchain system. Thus, we need to find out whether Aeolus is scalable with the increasing number of servers.

Based on the ERC20-balanced workload, we use a smart contract to test the performance in different configurations. The benchmark contract consists of two functions. The function "`NewAccount(account)`" (hereinafter referred to as "`NewAccount`") is used to push a new element into the array of accounts, reflecting the performance of the underlying key–value database and the noncross-shard performance. The function "`Transfer(from, to, value)`" (hereinafter referred to as "`Transfer`") is the most called function in real-world blockchain systems as Ethereum [23]. It reduces

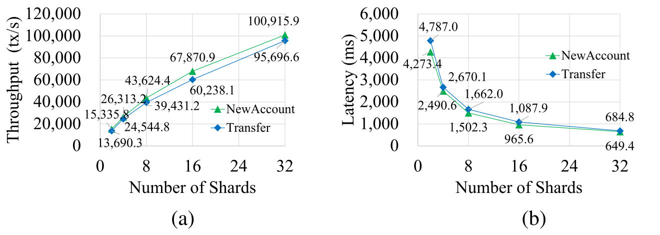

Fig. 5. Scalability of Aeolus-Geth in different number of shards (running ERC20-Balanced workload, each peer is with 2, 4, 8, 16, and 32 shards, measuring the throughput and latency). (a) Throughput. (b) Latency.

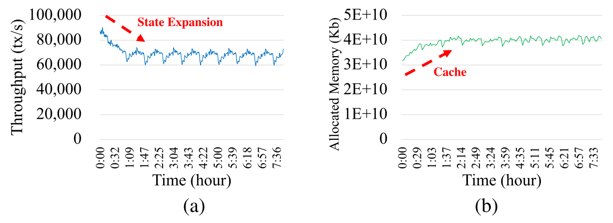

Fig. 6. Robustness of Aeolus-Geth in big scales (running 190 000 000 ERC20 transactions for ten times, spending 8 h in total, each peer is with 16 shards, measuring the throughput and memory consumption). (a) Throughput. (b) Memory of Shard.

the balance of an account and transfers the money to another account. In a sharded environment, it can reflect the performance of cross-shard transactions and is also tested by other sharding research [11].

Fig. 5(a) shows that the measured throughput (TPS) scales out as the number of shards increases. The upward trend is almost linear. At the experiment of 32 shards, Aeolus-Geth achieves 100 915.9 TPS for key–value insertion in "NewAccount" and 95 696.6 TPS for cross-shard transactions in "Transfer." More specifically, in this case, we use $(32+1)\times 4=132$ servers to conduct a permissioned blockchain with four peers, each with 32 shards. The transactions are confirmed in 1600 blocks, with 65 536 transactions per block, to make the system performance tend to a stable value. We also focus on the latency, observed by the client, from the time the transaction is submitted to the time the block is notified. It includes the time of transaction submission, execution, persistence, and response. The measured latency is shown in Fig. 5(b). At the experiment of 32 shards, the average latency is 684.8 ms, which is acceptable in the applications of permissioned blockchain systems. As the number of shards grows, the latency decreases. Fig. 5(b) shows that the latency reaches the limit at a certain value since the block persistence and transaction submission also take a certain amount of time. As shown in Fig. 5, the performance of the workload of "Transfer" is lower than the "NewAccount." Because "Transfer" is a cross-shard transaction, which needs one more round.

To evaluate the robustness of our system on a large scale, we run the ERC20-Balanced workloads for ten times in 16 shards. It takes 8 h for our system to deal with the transactions, as shown in Fig. 6. At the first run of ERC20-Balanced, since the state and cache expand with the raising accounts, the TPS decreases,

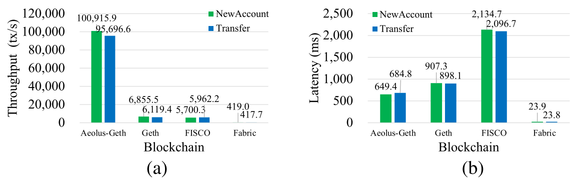

Fig. 7. Comparison of Aeolus-Geth, Geth, FISCO-BCOS, and Fabric (running ERC20-Balanced workload, using four peers, measuring the throughput and latency). (a) Throughput. (b) Latency.

and the memory increases. After that, since the number of accounts is constant, the fluctuation is caused by the distribution of transaction accounts. The TPS tends to a stable value, which reflects the robustness of Aeolus-Geth.

E. RQ3: Comparison With Other Systems

Since Aeolus is mainly designed for the permissioned blockchain, we compare the performance of Aeolus with other blockchain systems in the permissioned scenario of four peers in the consortium. We run Aeolus-Geth, original Go-Ethereum 1.9.12 (Geth), FISCO-BCOS v2.0 (FISCO), and Hyperledger Fabric v1.4 (Fabric) with the same workload. In a specific implementation, Aeolus-Geth, Geth, and FISCO run the EVM contract, while Fabric runs the contract in docker container [7]. Built-in consensus protocols in these systems are used, as Aeolus-Geth and Geth use the Clique protocol, Fabric uses the Raft protocol, and FISCO uses the PBFT protocol. Since there are four peers in the permissioned blockchain, each system consists of four blockchain peers with the same configuration. In addition, Fabric has one more orderer and Aeolus equips each peer with 32 shard servers. We test these blockchain systems with the workload of invoking two benchmark functions. The result of the comparison is shown in Fig. 7. As for transferring assets (cross-shard transaction), which is one of the most important scenarios, Aeolus-Geth can reach 95 696.6 TPS, which is 15.6 times that of the original Geth, 16.2 times that of FISCO, and 229.4 times that of Fabric. As shown in Fig. 7(b), Aeolus-Geth shows lower latency than Geth and FISCO, but higher than Hyperledger Fabric.

Then, we will discuss the detailed comparison of the experiment. As for Aeolus-Geth and the original Geth, Aeolus-Geth executes the transactions in a distributed way. As for FISCO-BCOS, FISCO also uses EVM contracts. However, in the real-world token transactions, a great number of tokens are transferred to the same account; then, FISCO needs to execute them in serial to avoid conflicts. Hence, FISCO shows lower throughput. As for Hyperledger Fabric, we use the default configuration, and it generates blocks at a fast speed. Each block is with about 10–20 transactions, generated in the duration of 20–30 ms, leading to lower throughput but also lower latency for Fabric.

Note: Fabric configuration cannot be changed to involve more transactions in a block to improve the throughput. Once we improve this configuration, more conflicts would be in the write set of Fabric, leading to transaction failure.

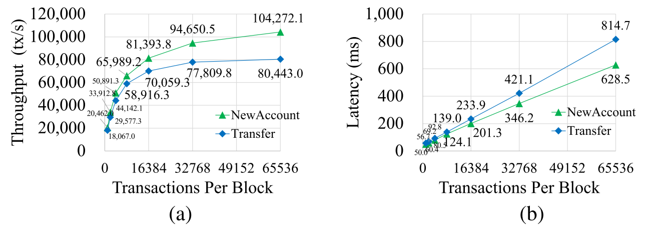

Fig. 8. Impact of the number of transactions in a block (running 6 553 600 transactions of ERC20-balanced workload, packing 2048, 4096, 8192, 16384, 32768, 65536 transactions per block, measuring the throughput and latency, each peer is with 16 shards). (a) Throughput. (b) Latency.

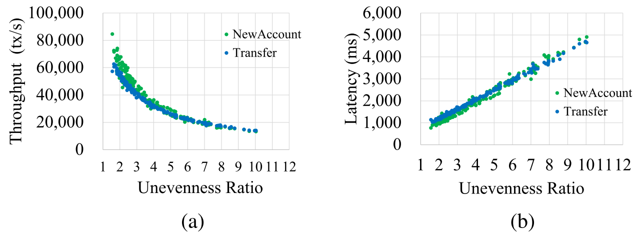

Fig. 9. Impact of the evenness of shards (running 6 553 600 transactions of ERC20-Unbalanced workload, measuring the throughput and latency; each peer is with 16 shards, and unevenness is the ratio of the max number of transactions in one shard to the average). (a) Throughput. (b) Latency.

In short, under the similar ERC20 workload to Monoxide [11], measured by the clients, Aeolus-Geth delivers 15.6× throughput of the original Geth. As for the other blockchain sharding studies, unfortunately, we can only compare with their claimed TPS in Section VII, although the consensus protocols and testbeds could be very different. The reasons and detailed comparison is also given in Section VII.

F. RQ4: Impact of Workload

From Section VI-D, we have learned that the performance of Aeolus can be affected by the number of shards and the state scales. Moreover, the transaction workloads can also have significant impacts on the performance. There are several critical factors of workload, which affect its performance: the size of block and the unevenness of transactions in each shard (also known as shard distribution). In this subsection, we will test different workloads to find the impacts.

In blockchain systems, a block needs to propagate in the peer-to-peer network; thus, the block size affects the performance. The size of the block depends on the number of transactions per block. In the previous experiment of Aeolus-Geth, the workloads are set with 65 536 transactions per block (TPB). Therefore, we use the workloads in different TPB. The results are shown in Fig. 8. If TPB is too small, the block costs little time for execution, wasting much time in communication, resulting in lower throughput. However, even using a workload of 1024 TPB, Aeolus-Geth achieves more than 18 000 TPS, which is also higher than the compared blockchain systems in RQ2. Smaller blocks result in lower latency, as shown in Fig. 8(b). If the block consists of 16 384 transactions, the latency is about 200 ms. Therefore, if the blockchain application requires a latency of about 200 ms, Aeolus-Geth can also meet the requirement, with the throughput of 70 000 TPS to 80 000 TPS, equipped with 16 shards.

In the Internet environment, the bandwidth is not unlimited. Thus, we need to measure the network consumption in Aeolus-Geth. The size of the block including 65 536 "`NewAccount`" transactions is about 8987.2 kB. As for "`Transfer`," the size of the block is 15 334.3 kB on average, as the transaction data include more parameters for the contract function. Take the "`NewAccount`" block with 65 536 TPB as an example, the latency is 628.5 ms and the size is 8987.2 kB. This indicates that Aeolus-Geth requires the network speed to be higher than 8987.2 kB/628.5 ms = 14.3 MB/s, to achieve the throughput of 104 272.1 TPS. Nowadays, this network requirement is acceptable in most Internet service providers. As for internal communication, the major data packet is the packet of JUMP messages. The size of compressed JUMP messages of each block is only 1750.3 kB on average, which can be propagated very fast through the internal network of the cluster.

Fig. 9 shows the scatter diagram of performance under different uneven workloads. One dot represents one block, the horizontal axis is the unevenness of its transactions in each shard, and the vertical axis is throughput and latency. In this article, we calculate the unevenness by the ratio of the max number of transactions in one shard to the average. For example, in a block with 65 536 transactions in 16 shards, if the maximum number of one shard is 8192, then the unevenness is 8192/(65 536/16) = 2. Fig. 9 shows that the throughput goes lower in the more uneven block. The reason is that in each round of Aeolus, the execution master has to wait for all the shards to finish the round before the next round. Therefore, the execution will be blocked, waiting for the last one who has the most transactions. And this waiting in the master is worthy. It is very likely to provide a global clock for each shard, making it easier and safer for the developers to write smart contracts.

VII. Related Work

Since this article proposes the Aeolus as a sharded permissioned blockchain, the related studies will be categorized into sharded permissionless blockchain and permissioned blockchain. After that, we will provide a detailed comparison.

Permissionless blockchain: Sok [27] provides a systematic review on blockchain sharding techniques. Luu *et al.* [8] propose a secure sharding protocol for the permissionless blockchain. After that, more sharding strategies of blockchain are proposed. Zilliqa [9] blockchain assigns validating peers to each shard in groups of 600. OmniLedger [10] distributes generated verifier identities to different shards through blockchain. RapidChain [12] is proposed to be resilient to Byzantine faults from up to 1/3 malicious peers. RepChain [28] integrates reputation scores with sharding-based blockchain with an incentive mechanism, characterizing the heterogeneity among the validators.

TABLE I
COMPARISON BETWEEN AEOLUS AND RELATED BLOCKCHAIN SHARDING STUDIES (CS' REPRESENTS CROSS-SHARD)

Permission	Blockchain	CS' TPS	# Shards	# Machines
Permissionless	Elastico [8]	40	100	800
	Ominiledger [10]	5,860	16	60
	RapidChain [12]	7,380	250	32
	Monoxide [11]	11,694	2,048	1,200
	RepChain [28]	6,852	225	900
Permissioned	AHL+ [31]	3,000	36	972
	Aeolus-Geth	**95,696**	32	132

Monoxide [11] uses Chu-ko-nu mining to defend 51% malicious peers in the sharded blockchain. As for dynamic sharding, Tao *et al.* [29] propose an intershard merging algorithm for shard merging and an intrashard transaction selection mechanism that select transactions for validation. Huang *et al.* [30] propose a cross-shard blockchain protocol that exploits fine-grained state partition. In the cryptocurrency community, Ethereum 2.0 [6] uses shading with a beacon chain to maintain the security and consistency for shard chains.

Permissioned blockchain: AHL+ [31] enhances Hyperledger Fabric with a two-phase commit method to process the distributed transactions in the sharded permissioned blockchain. Huang *et al.* [32] propose a drift-plus-penalty (DPP-based) algorithm that can alleviate the imbalanced transaction assignment in the PBFT-based sharded permissioned blockchain. ByShard [33] is proposed as a unifying framework for the study of sharded blockchain in the Byzantine environment, with 18 cross-shard transaction processing protocols. Blockchain also has wide applications in industrial IoT security [34], [35]. Jia *et al.* [36] designed a model of blockchain-enabled federated learning [37] to enhance security in the industrial IoT.

Comparison: We compare Aeolus with the related sharding studies that claim to have been actually tested on machines, comparing the parameters of throughput, shards, and machines. Sincerely, the testbeds could be very different, since we cannot obtain the source code from others for a more fair evaluation. Even though AHL+ is open source, it is based on Fabric where the smart contract model is completely different from Aeolus-Geth. As shown in Table I, Aeolus-Geth shows better cross-shard throughput than the others, as the others are mainly designed for the permissionless blockchain. The advantages of Aeolus are as follows. From above, except for Aeolus, the blockchain peer cannot maintain the full ledger. As mentioned before, maintaining a partial ledger will take a longer time for cross-shard communication because the peer that holds a partial ledger needs extra cryptographic algorithms or network communication to trust another peer. However, in Aeolus, the ledger is fully maintained by the peer, thus improving the cross-shard efficiency. Moreover, this partial ledger is not suitable for some kinds of blockchain applications. Especially, in a permissioned blockchain, the peer in the network usually needs to interact with all the data in the ledger. Once the peer does not own the complete ledger by itself, the credibility and availability of the data will decline. In this article, the blockchain peer in Aeolus still maintains the ledger completely to enhance the trust-worthy.

VIII. DISCUSSION

Generality: As mentioned before, Aeolus only changes the transaction structure rather than the block structure. In blockchain, most consensus protocols (e.g., PoW [5], PoS [15], PoA [16], PBFT [17], etc.) reach the consensus of blocks among peers. Hence, Aeolus can be adapted to these consensus protocols without tuning. However, PoH [26] needs a further extension to be adapted since PoH sequences the transactions rather than the blocks, where the sequences should be verified.

Elastic scaling: In most blockchain applications, the performance requirements are not always high. To save the fees and reduce operating costs, it is necessary to study the method of automatically expanding and reducing the number of shards.

IX. CONCLUSION

In this article, we proposed Aeolus blockchain to achieve the distributed execution of blockchain transactions. It is motivated by the performance evaluation of current blockchain systems that finds one of the bottlenecks in the transaction execution. To accelerate the transaction execution, Aeolus proposes a distributed blockchain transaction structure that enables the transaction to be executed on multiple servers. Aeolus also proposes distributed state update sharding as a "Master-Shards" architecture to keep the consistency of different clusters. Aeolus is implemented on Geth. On a testbed of 132 servers, measured by the clients, Aeolus-Geth can achieve more than 100 000 TPS for ERC20 transactions that are extracted from the real-world blockchain. Its performance is 15.6 times that of the original Geth. It is also stable to continuously process ERC20 transactions for more than 8 h.

In the future, besides the mentioned elastic scaling, the possible improvements are as follows:

1) *Efficient communication inside the cluster:* The communication in Aeolus causes a significant overhead on the execution master. An optional method is to make the shard server communicate with each other in the internal execution.
2) *Combined research on network sharding:* As the network among the peers can be the bottleneck, network sharding can be considered to reduce the overhead of peers to achieve further improvement.
3) *Customizable dependence handling algorithm:* Traditional concurrency control solutions are with potential further improvement for blockchain transactions. Once the dependence can be handled by the developer, it could be more efficient, leading to higher throughput.

REFERENCES

[1] H.-N. Dai, Z. Zheng, and Y. Zhang, "Blockchain for Internet of Things: A survey," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8076–8094, Oct. 2019.

[2] W. Chen *et al.*, "Cooperative and distributed computation offloading for blockchain-empowered industrial Internet of Things," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8433–8446, Oct. 2019.

[3] AliCloud IoT Platform, 2021. [Online]. Available: https://www.alibabacloud.com/product/iot/

[4] P. Zheng, Z. Zheng, X. Luo, X. Chen, and X. Liu, "A detailed and real-time performance monitoring framework for blockchain systems," in *Proc. 40th Int. Conf. Softw. Eng.: Softw. Eng. Pract.*, 2018, pp. 134–143.

[5] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. Accessed: Apr. 15, 2022. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[6] V. Buterin, "Ethereum 2.0 mauve paper," 2016. Accessed: Apr. 15, 2022. [Online]. Available: https://cdn.hackaday.io/files/10879465447136/Mauve%20Paper%20Vitalik.pdf

[7] E. Androulaki, C. Cachin, K. Christidis, C. Murthy, B. Nguyen, and M. Vukolic, "Hyperledger fabric proposals: Next consensus architecture proposal," 2016. [Online]. Available: https://github.com/hyperledger-archives/fabric/wiki/

[8] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 17–30.

[9] *The ZILLIQA Whitepaper*, ZILLIQA Team, Singapore, 2017. [Online]. Available: https://zilliqa.com

[10] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "OmniLedger: A secure, scale-out, decentralized ledger via sharding," in *Proc. IEEE Symp. Secur. Privacy*, 2018, pp. 583–598.

[11] J. Wang and H. Wang, "Monoxide: Scale out blockchains with asynchronous consensus zones," in *Proc. 16th USENIX Symp. Netw. Syst. Des. Implementation*, 2019, pp. 95–112.

[12] M. Zamani, M. Movahedi, and M. Raykova, "RapidChain: Scaling blockchain via full sharding," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 931–948.

[13] M. J. Amiri, D. Agrawal, and A. El Abbadi, "ParBlockchain: Leveraging transaction parallelism in permissioned blockchain systems," in *Proc. 39th IEEE Int. Conf. Distrib. Comput. Syst.*, 2019, pp. 1337–1347.

[14] FISCO BCOS, 2019. [Online]. Available: https://fisco-bcos-documentation.readthedocs.io/

[15] Peercoin, 2012. [Online]. Available: https://github.com/peercoin/peercoin/

[16] Go Ethereum, 2017. [Online]. Available: https://github.com/ethereum/go-ethereum/

[17] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proc. 3rd Symp. Oper. Syst. Des. Implementation*, 1999, pp. 173–186.

[18] N. Szabo, "The idea of smart contracts," 1997. Accessed: Apr. 15, 2022. [Online]. Available: https://nakamotoinstitute.org/the-idea-of-smart-contracts/

[19] T. Chen *et al.*, "An adaptive gas cost mechanism for Ethereum to defend against under-priced DoS attacks," in *Proc. Int. Conf. Inf. Secur. Pract. Experience*, 2017, pp. 3–24.

[20] J. M. Faleiro, D. J. Abadi, and J. M. Hellerstein, "High performance transactions via early write visibility," *Proc. VLDB Endowment*, vol. 10, pp. 613–624, 2017.

[21] J. M. Faleiro and D. J. Abadi, "Rethinking serializable multiversion concurrency control," *Proc. VLDB Endowment*, vol. 8, pp. 1190–1201, 2015.

[22] S. Mehrotra, R. Rastogi, Y. Breitbart, H. F. Korth, and A. Silberschatz, "Ensuring transaction atomicity in multidatabase systems," in *Proc. 11th ACM SIGACT-SIGMOD-SIGART Symp. Princ. Database Syst.*, 1992, pp. 164–175.

[23] P. Zheng, Z. Zheng, J. Wu, and H.-N. Dai, "Xblock-ETH: Extracting and exploring blockchain data from Ethereum," *IEEE Open J. Comput. Soc.*, vol. 1, pp. 95–106, 2020.

[24] Binance Smart Chain, 2021. Accessed: Apr. 15, 2022. [Online]. Available: https://github.com/bnb-chain/bsc/

[25] Solana Github, 2018. Accessed: Apr. 15, 2022. [Online]. Available: https://github.com/solana-labs/solana

[26] A. Yakovenko, "Solana: A new architecture for a high performance blockchain," *Whitepaper*, 2018.

[27] G. Wang, Z. J. Shi, M. Nixon, and S. Han, "SoK: Sharding on blockchain," in *Proc. 1st ACM Conf. Adv. Financial Technol.*, 2019, pp. 41–61.

[28] C. Huang *et al.*, "RepChain: A reputation-based secure, fast, and high incentive blockchain system via sharding," *IEEE Internet Things J.*, vol. 8, no. 6, pp. 4291–4304, Mar. 2021.

[29] Y. Tao, B. Li, J. Jiang, H. C. Ng, C. Wang, and B. Li, "On sharding open blockchains with smart contracts," in *Proc. IEEE 36th Int. Conf. Data Eng.*, 2020, pp. 1357–1368.

[30] H. Huang *et al.*, "BrokerChain: A cross-shard blockchain protocol for account/balance-based state sharding," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2022. [Online]. Available: https://drive.google.com/file/d/1PSWsk9aSCVZ2BmqE2aSfH4IRG4pUzRqr/view

[31] H. Dang *et al.*, "Towards scaling blockchain systems via sharding," in *Proc. Int. Conf. Manage. Data*, 2019, pp. 123–140.

[32] H. Huang *et al.*, "Elastic resource allocation against imbalanced transaction assignments in sharding-based permissioned blockchains," *IEEE Trans. Parallel Distrib. Syst.*, vol. 33, no. 10, pp. 2372–2385, Oct. 2022.

[33] J. Hellings and M. Sadoghi, "ByShard: Sharding in a byzantine environment," *Proc. VLDB Endowment*, vol. 14, pp. 2230–2243, 2021.

[34] D. V. Medhane, A. K. Sangaiah, M. S. Hossain, G. Muhammad, and J. Wang, "Blockchain-enabled distributed security framework for next-generation IoT: An edge cloud and software-defined network-integrated approach," *IEEE Internet Things J.*, vol. 7, no. 7, pp. 6143–6149, Jul. 2020.

[35] A. K. Sangaiah, D. V. Medhane, G.-B. Bian, A. Ghoneim, M. Alrashoud, and M. S. Hossain, "Energy-aware green adversary model for cyberphysical security in industrial system," *IEEE Trans. Ind. Informat.*, vol. 16, no. 5, pp. 3322–3329, May 2020.

[36] B. Jia, X. Zhang, J. Liu, Y. Zhang, K. Huang, and Y. Liang, "Blockchain-enabled federated learning data protection aggregation scheme with differential privacy and homomorphic encryption in IIoT," *IEEE Trans. Ind. Informat.*, vol. 18, no. 6, pp. 4049–4058, Jun. 2022.

[37] Y. Li, C. Chen, N. Liu, H. Huang, Z. Zheng, and Q. Yan, "A blockchain-based decentralized federated learning framework with committee consensus," *IEEE Netw.*, vol. 35, no. 1, pp. 234–241, Jan./Feb. 2021.

Peilin Zheng (Member, IEEE) is currently working toward the Ph.D. degree in software engineering with Sun Yat-sen University, Guangzhou, China.

His research interests include blockchain optimization, smart contracts, and blockchain-based decentralized applications.

Quanqing Xu (Senior Member, IEEE) received the Ph.D. degree in computer science from Peking University, Beijing, China, in 2009.

He is currently with the Research Institute of Ant Group, Hangzhou, China. He was a Research Scientist with the Agency for Science, Technology and Research, Singapore, and an Adjunct Faculty with the Singapore Institute of Technology, Singapore. His research interests mainly include blockchain, database systems, and distributed systems.

Dr. Xu is the recipient of the Award of Excellent Ph.D. Graduate because of his excellent performance at Peking University. He is a fellow of the Institution of Engineering and Technology and a Senior Member of the ACM and the China Computer Federation.

Xiapu Luo (Member, IEEE) received the Ph.D. degree in computer science from the Hong Kong Polytechnic University, Hong Kong, in 2007.

He is an Associate Professor with the Department of Computing, Hong Kong Polytechnic University, Hong Kong. His research interests include blockchain/smart contracts, mobile/Internet of Things security and privacy, network/web security and privacy, software engineering, and Internet measurement with papers published in top security/software engineering/networking conferences and journals.

Prof. Luo is the recipient of the eight best paper awards, including the ACM SIGSOFT Distinguished Paper Award at 2021 International Conference on Software Engineering, the Best Paper Award at 2018 IEEE International Conference on Computer Communications, the Best Research Paper Award at 2016 IEEE International Symposium on Software Reliability Engineering, and several awards from the industry. He regularly serves in the Program Committee of major security conferences and is an Editor for IEEE/ACM TRANSACTIONS ON NETWORKING.

Zibin Zheng (Senior Member, IEEE) received the Ph.D. degree in computer science from the Chinese University of Hong Kong, Hong Kong, in 2011.

He is currently a Professor with Sun Yat-Sen University, Guangzhou, China. His research interests include blockchain, services computing, and software engineering.

Dr. Zheng is the recipient of the ACM SIGSOFT Distinguished Paper Award at 2010 International Conference on Software Engineering and the Best Student Paper Award at 2010 IEEE International Conference on Web Services.

Weilin Zheng is currently working toward the M.Eng. degree with the School of Computer Science and Engineering, Sun Yat-Sen University, Guangzhou, China.

His research interests include performance monitoring and optimization, blockchain computing power utilization, blockchain data analysis, and blockchain-based decentralized applications.

Xu Chen received the B.S. degree in software engineering in 2019 from Sun Yat-Sen University, Guangzhou, China, where he is currently working toward the M.S. degree with the School of Computer Science and Engineering.

His current research interests include blockchain, Internet of Things, and distributed systems.

Zhiyuan Zhou received the master's degree in computer science from the Chinese Academy of Sciences, Beijing, China, in 2007.

He is currently a Staff Engineer with Blockchain Platform Division, Ant Group, Hangzhou, China. He is currently leading a team to deliver the core engine of Antgroup's consortium blockchain system, specifically designed for Internet scale asset exchange. Besides engineering, he is doing research on blockchain scalability. He also had profound experience in cloud computing.

Ying Yan received the Ph.D. degree in computer science from Fudan University, Shanghai, China, in 2008.

She is currently a Senior Staff Engineer with Blockchain Platform Division, Ant Group, Hangzhou, China. She was the Lead Researcher and the Head of the Blockchain Department, Microsoft Research Asia. Her current research interests include blockchain confidentiality and scalability.

Dr. Yan was on the list of Forbes China 50 Top Women In Tech in 2019. She was the Editor-in-Chief of *Ethereum: Detailed Explanations and Practices*.

Hui Zhang (Senior Member, IEEE) received the Ph.D. degree in computer science from the University of Southern California, Los Angeles, CA, USA, in 2005.

He is currently the Senior Director of the Blockchain Platform Division, Ant Group, Hangzhou, China, and also the Head of the Blockchain Laboratory, Alibaba Damo Academy, Hangzhou. He is responsible for the R&D and commercialization of Ant Group's blockchain technology. Prior to joining Ant Group, he was the Head of the Department of Systems Research, NEC Laboratories America, Princeton, NJ, USA, focusing on R&D for high-performance distributed systems and networks, especially peer-to-peer network algorithms and big data analytics.