Achieving Scalability and Load Balance across Blockchain Shards for State Sharding

Canlin Li*, Huawei Huang*†, Yetong Zhao*, Xiaowen Peng*, Ruijie Yang§, Zibin Zheng*†, Song Guo‡
†School of SSE, *School of CSE, Sun Yat-sen University.
‡Department of Computing, The Hong Kong Polytechnic University. Email: song.guo@polyu.edu.hk
§Huawei Blockchain Lab, Huawei Cloud Tech Co., Ltd.
Corresponding author: Huawei Huang. Email: huanghw28@mail.sysu.edu.cn

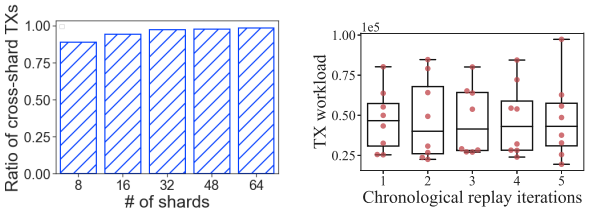
2022 41st International Symposium on Reliable Distributed Systems (SRDS) | 978-1-6654-9753-4/22/$31.00 ©2022 IEEE | DOI: 10.1109/SRDS55811.2022.00034

***Abstract*—Sharding technique is viewed as the most promising solution to improving blockchain scalability. However, to implement a sharded blockchain, developers have to address two major challenges. The first challenge is that the ratio of cross-shard transactions (TXs) across blockchain shards is very high. This issue significantly degrades the throughput of a blockchain. The second challenge is that the workloads across blockchain shards are largely imbalanced. If workloads are imbalanced, some shards have to handle an overwhelming number of TXs and become congested very possibly. Facing these two challenges, a dilemma is that it is difficult to guarantee a low cross-shard TX ratio and maintain the workload balance across all shards, simultaneously. We believe that a fine-grained account-allocation strategy can address this dilemma. To this end, we first formulate the tradeoff between such two metrics as a network-partition problem. We then solve this problem using a community-aware account partition algorithm. Furthermore, we also propose a sharding protocol, named *Transformers*, to apply the proposed algorithm into the sharded blockchain system. Finally, trace-driven evaluation results demonstrate that the proposed protocol outperforms other baselines in terms of throughput, latency, cross-shard TX ratio, and the queue size of transaction pool.**

I. Introduction

Sharding is viewed as the most promising technique to improving blockchain scalability while not violating the decentralization principle of blockchain systems [1]–[7]. Using sharding technique, an original blockchain can be partitioned into multiple smaller committees, each is called a network shard. Blockchain shards verify and execute transactions (TXs) in parallel. Thus, the throughput of a sharded blockchain can be much boosted.

Currently, sharding technologies are classified into three categories, i.e., network sharding, transaction sharding and state sharding. Network sharding is the foundation of the other two. Basically, network sharding divides all blockchain nodes into different parallel groups in the network layer. The major difference between transaction sharding and state sharding associates with the storage of blockchain ledger's state. Transaction sharding requires every blockchain node to store the whole ledger state such as the *world state* in Ethereum's blockchain. In contrast, each blockchain node in state sharding only has to store a subset of the holistic ledger. However, many other technical issues occur when dividing a large state ledger into multiple smaller ones. The most representative issues include the atomicity of cross-shard TXs [7] and the security issues [8] in the circumstance of sharded blockchains. In summary, state sharding is still in its immature stage and faces many technique challenges when applying it into practice.



(a) Cross-shard TX ratio is high (b) Highly imbalanced workloads

Fig. 1. Results of motivation simulation. Following the account assignment policy of Monoxide [7], we observe a high ratio of cross-shard TXs shown in (a), and imbalanced shard workloads across blockchain shards shown in (b).

Motivation. Although several state-of-the-art blockchain sharding solutions [1]–[7] have been proposed, a practical state sharding solution is still absent from the literature. Thus, we are curious about why the account-based state sharding is so difficult to implement. To find clues, we conduct the following motivation simulations to emulate the TX assignment of a blockchain sharding system. We first collect one million continuous historical TXs from the real-world Ethereum [9] dataset. To emulate blockchain shards, we then divide all TXs into five continuous groups following their chronological sequences. We finally replay them in 5 sequential consensus iterations. In each iteration, 200,000 (200K) TXs are assigned to totally 8 blockchain shards referring to the assignment policy of Monoxide [7], which is a classic account-based state sharding protocol. Through the simulations depicted above, we have the following two findings.

Finding 1: The ratio of cross-shard TXs is high. As demonstrated in Fig. 1(a), when the number of shards increases, the cross-shard TX ratio grows and approaches to 100%. The overwhelming number of cross-shard TXs could degrade the system throughput and increase the inter-shard communication cost. In Monoxide [7], the authors also mentioned this issue but they did not address it. Monoxide protocol partitions all accounts according to the prefix/suffix of their addresses into different state shards on average. The problem

2575-8462/22/$31.00 ©2022 IEEE
DOI 10.1109/SRDS55811.2022.00034

is that such account-assignment policy does not consider the correlations between accounts. When the accounts having strong correlations are assigned to different shards, a large number of cross-shard TXs will be induced.

Finding 2: The TX workloads across blockchain shards are imbalanced. Fig. 1(b) shows the imbalanced workloads across 8 blockchain shards. Some shards are congested across different replay iterations and some are almost empty. In this paper, we call such congested and empty shards the *hot shards* and *cold shards*, respectively. Those imbalanced TX workloads significantly limit the scalability of a blockchain sharding system. The reasons are two-folded. Firstly, the imbalanced workloads enforce hot shards to store and process more TXs comparing with cold shards. Secondly, the cross-shard TXs congested in hot shards suffer from large confirmation latency.

Motivated by the two findings mentioned above, we emphasize the following two goals in this paper. i) To avoid a large ratio of cross-shard TXs, we intend to schedule all accounts located in different shards elastically taking their correlations into consideration. ii) We also aim to alleviate the workload imbalance across blockchain shards. To achieve our goals, we devise an elastic sharding protocol for the blockchain state sharding. Compared with existing sharding solutions, the proposed protocol can not only reallocate accounts across all blockchain shards in each consensus iteration, but also improve the scalability of sharded blockchain while promising the security of blockchain shards.

Our study includes the following **contributions**.

- To describe the tradeoff between the workload balance of shards and the cross-shard TX ratio, we first formulate an optimization problem. To solve this problem, we then devise a community-aware account-partition algorithm.
- We then propose an elastic sharding protocol, named *Transformers*, to apply the devised account-partition algorithm into state-sharding blockchains.
- Finally, we conduct trace-driven simulations using the real-world Ethereum TX dataset. The evaluation results show that the proposed protocol outperforms other baselines in terms of throughput, TX confirmation latency, cross-shard TX ratio, and the queue size of TX pool.

The rest of this paper is organized as follows. Section II reviews the related work. Section III presents problem formulation. Section IV elaborates the proposed account-partition algorithm. We then present and analyze the proposed sharding protocol in Sections V and VI. Section VII shows the evaluation results. Finally, Section VIII concludes this paper.

II. Related Work

A. Blockchain Sharding

A blockchain sharding system is operated in *epochs*. Each epoch includes two critical phases: consensus phase and reconfiguration phase. In the consensus phase, each shard committee processes TXs through intra-shard consensus locally. In the reconfiguration phase, blockchain nodes may join or leave the system, thus shard committees need to be shuffled under a given policy. In consequence, shard committees are reconfigured to avoid Sybil attacks [10] and Bribery attacks [11]. Elastico [1] is viewed as the first sharding system designed for public blockchain. In Elastico, member committees verify disjoint groups of TXs in parallel. The final committee then include the verified TX blocks in the main chain. Elastico only achieves *transaction sharding*, which requires every blockchain node to store the holistic state ledger, and thus costs a lot of storage and communication resources. Since then, researchers have paid great attention to *state sharding*, and have proposed several state-of-the-art protocols such as OmniLedger [3], RapidChain [2] and Monoxide [7]. Recently, some emerging studies [12], [13] are investigating the state sharding of smart contracts, which is a promising research direction.

B. Transaction Models

Blockchain systems currently execute through exploiting two major types of transaction models, i.e., the Unspent Transaction Output (UTXO) model and the account/balance model. Firstly, in a UTXO-based blockchain, users can have multiple UTXOs in their digital wallets. When a transaction spends a certain number of UTXOs, new UTXOs will be generated and can be spent by other future TXs. Such UTXO model is widely adopted by Bitcoin [14] and several sharded blockchains [1]–[3]. Secondly, in an account-based blockchain, each user has his own accounts. Tokens are deposited in these accounts just as the way we experience in bank system. The account/balance model is believed more promising than UTXO model since it supports smart contracts. Furthermore, the TX structure of account/balance model is conciser because a TX is only associated with one sender account and one receiver account. The account/balance model is also generally adopted by sharding blockchains such as Ethereum 2.0 [9] and Monoxide [7].

Based on UTXO, OmniLedger [3] proposes a novel *Byzantine Shard Atomic Commit* protocol to ensure the atomicity of cross-shard TXs. However, the bottleneck of OmniLedger is in the client end. RapidChain [2] proposes a *transfer mechanism* to relieve the responsibility of client users. Using the account-based model, Monoxide [7] exploits the *relay transaction* mechanism, which is in fact a message-relay TX processing approach. Although relay TXs guarantee the eventual atomicity of cross-shard TXs, those TXs may experience long confirmation latency.

C. Cross-shard TX Processing in State Sharding

Cross-shard TXs are inevitable in a sharded blockchain [7]. State sharding only requires blockchain nodes in each shard to store a part of the whole state ledger. However, the verification and the execution of cross-shard TXs under state sharding are challenging, because the inputs and outputs of a cross-shard TX locate in different shards. Thus, how to process the cross-shard TXs efficiently while guaranteeing their eventual atomicity is a challenge in state sharding.

We notice that a similar work is BrokerChain [8], in which the authors also consider that the throughput of a sharded

blockchain is significantly affected by the high cross-shard TX ratio. Through comparison, we summarize the difference between our work and BrokerChain [8] as follows. BrokerChain partitions state ledger into multiple groups in the *protocol layer* by exploiting broker accounts. Their mechanism mainly aims to accelerate the confirmation of cross-shard TXs. What's more, BrokerChain did not provide a formal formulation for account's partition problem. In this paper, through considering the tradeoff between the workload balance across shards and the cross-shard TX ratio, we formulate an optimization problem and design an algorithm to partition user accounts in the *account layer* of a sharded blockchain system. In addition, we also analyze both the overhead and the security guarantee of the proposed sharding protocol.

Comparing with the state-of-the-art blockchain sharding solutions [2]–[4], [7], [8], [15], the unique contribution of our proposed sharding protocol relies on that our approach ties to find a tradeoff between the workload balance across all shards and the ratio of cross-shard TXs.

III. System Model And Problem Statement

A. Preliminaries

The proposed sharding protocol is based on the account/balance transaction model. Similar to existing sharding blockchains [1]–[4], we utilize Byzantine Fault Tolerance (BFT)-based consensus as the intra-shard consensus mechanism. Throughput is a significant metric of blockchain system [1], [2], [7]. Furthermore, referring to BrokerChain [8], the cross-shard TX Ratio (shorten as *CTR*) and the workload distribution across blockchain shards significantly affect the throughput and TX confirmation latency of a sharded blockchain. Thus in this paper, we focus on improving the throughput and reducing the TX confirmation latency of a sharded blockchain.

B. The Impact of Account Partition

As mentioned in Section I, the excessive number of cross-shard TXs and workload imbalance issue are caused partially by the unreasonable partition of accounts. To have a better understanding, we illustrate the effect of account partition using the example shown in Fig. 2. The account network includes 8 account addresses and 7 edges. Each edge represents the TX correlation between a pair of accounts. The edge weight denotes the number of TXs occurred between those two associated accounts. For simplicity, we measure the weight of each edge as 1, and suppose the blockchain system has 2 shards. The workload of each shard is calculated by the summed number of TXs contributed by this shard's accounts. We then use the absolute value of the workload difference between the two shards to measure the index of workload imbalance. As shown in Fig. 2(b), such account partition leads to only one cross-shard TX. However, shard #1 needs to process 7 TXs while shard #2 only needs to process one TX. Thus, the current index of workload imbalance is as high as 6. In the next partition shown in Fig. 2(c), the TX workloads of both shards are perfectly balanced. However, this account

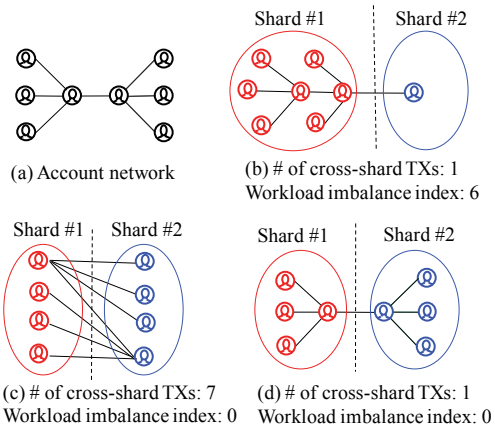


Fig. 2. The effects of different account-partition solutions.

partition strategy induces 7 cross-shard TXs, which are large TX workloads to both shards. The ideal account partition is shown in Fig. 2(d), where only one cross-shard TX is observed and the index of workload imbalance is 0.

From the example shown above, we can see the close correlation between CTR and workload distribution. A good account-partition solution can not only reduce the number of cross-shard TXs, but also help all shards reach workload balance. To better illustrate the impact of account partition, we formulate it as an optimization problem as shown in the following subsection.

C. Problem Formulation of Account Partition

Consider an account network denoted by $G(V, E)$, where V is a set of N account addresses written as $[N] = \{1, 2, ..., N\}$, and E is a set of edges associated with these addresses. We use $[K] = \{1, 2, ..., K\}$ to denote the set of K shards. For each account address $i (\in [N])$, we define a binary variable $x_{i,k} (k \in [K])$ to denote the assignment of an account:

$$x_{i,k} = \begin{cases} 1, & \text{address } i \in [N] \text{ is assigned to shard } k \in [K]; \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Furthermore, each address i should be assigned to exactly one shard. That is, $\sum_{k=1}^{K} x_{i,k} = 1,\ i \in [N]$. We also define $\boldsymbol{x} = \{x_{i,k},\ i \in [N],\ k \in [K]\}$ to denote the account-partition solution of all addresses.

We then use $e_{i,j} \geq 0\ (i, j \in [N])$ to denote the edge weight between addresses i and j, i.e., the number of TXs associated with addresses i and j. Recall that assigning addresses i and j into two different shards will cause an extra number $e_{i,j}$ of relay TXs, which increase TX workloads at the associated shards. If addresses i and j are assigned into the same shard, we have $\sum_{k=1}^{K} x_{i,k} \cdot x_{j,k} = 1$; otherwise, the result is 0. Let $C(\boldsymbol{x})$ denote the induced cross-shard TX workloads under a

account-partition solution $\boldsymbol{x}$, we have

$$C(\boldsymbol{x}) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}[e_{i,j}\cdot(1-\sum_{k=1}^{K}x_{i,k}\cdot x_{j,k})]. \quad (2)$$

TX workload of each shard includes both intra-shard and cross-shard TXs. Let $L_{k,l}(\boldsymbol{x})$ denote the cross-shard TX workload between shards k and l ($k,l \in [K], k \neq l$), i.e., the total weight of edges associated with these two shards. Then $L_{k,l}(\boldsymbol{x})$ can be calculates as:

$$L_{k,l}(\boldsymbol{x}) = \sum_{i=1}^{N}\sum_{j=1}^{N} e_{i,j}\cdot x_{i,k}\cdot x_{j,l},\ \forall k,l \in [K],\ k \neq l. \quad (3)$$

For the case $l = k$, $L_{k,k}(\boldsymbol{x})$ represents the intra-shard TX workload of shard k, which is expressed as:

$$L_{k,k}(\boldsymbol{x}) = \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} e_{i,j}\cdot x_{i,k}\cdot x_{j,k},\ \forall k \in [K]. \quad (4)$$

Thus, the total TX workload of shard k, denoted by $L_k(\boldsymbol{x})$, is written as:

$$\begin{aligned} L_k(\boldsymbol{x}) &= \sum_{l=1}^{K} L_{k,l}(\boldsymbol{x}) = \sum_{l=1,l\neq k}^{K}\sum_{i=1}^{N}\sum_{j=1}^{N} e_{i,j}\cdot x_{i,k}\cdot x_{j,l} \\ &+ \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N} e_{i,j}\cdot x_{i,k}\cdot x_{j,k},\ \forall k \in [K]. \end{aligned} \quad (5)$$

We then define $\hat{L}(\boldsymbol{x}) = \frac{1}{K}\sum_{k=1}^{K} L_k(\boldsymbol{x})$ as the average workload of the sharding system and $D(\boldsymbol{x})$ as the system workload imbalance index. A larger $D(\boldsymbol{x})$ represents a more unbalanced system workload. $D(\boldsymbol{x})$ is computed as:

$$D(\boldsymbol{x}) = \max_{k\in[K]} |L_k(\boldsymbol{x}) - \hat{L}(\boldsymbol{x})|. \quad (6)$$

When given an account network, our objective is to find a well-devised account partition solution that can yield the fewest number of cross-shard TXs and the lowest workload imbalance index. We now formulate the account partition problem as the following optimization.

$$\begin{aligned} &\min\ F(\boldsymbol{x}) = \alpha\cdot C(\boldsymbol{x}) + (1-\alpha)\cdot D(\boldsymbol{x}) \\ &s.t.\ \sum_{k=1}^{K} x_{i,k} = 1,\ i \in [N]. \\ &\text{Variables}: x_{i,k} \in \{0,1\},\ \forall i \in [N],\ k \in [K], \end{aligned} \quad (7)$$

where $\alpha \in [0,1]$ is a predefined coefficient measuring the weight between the two objective terms $C(\boldsymbol{x})$ and $D(\boldsymbol{x})$.

Referring to the conventional graph-partition problems [16]–[18], to find the optimal solution of the above problem is NP-hard [19]. Fortunately, we find the account network is a social network. Thus, we are motivated to utilize the *community detection* mechanism to devise an account-partition algorithm, which is presented in the next section.

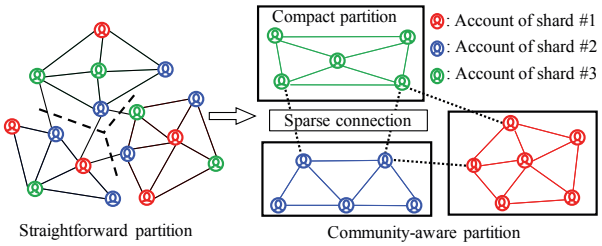


Fig. 3. Straightforward partition vs. community-aware partition.

IV. Community-aware Partition Algorithm

A. Community Detection over Blockchain's Account Network

In a complex network, communities refer to node groups in which connections are dense. Nodes within each community have strong mutual connections. In contrast, the inter-community connections of nodes are sparse. Community reflects the local properties of individual node behavior in a network. Community detection aims to find the communities in a given complex network.

All accounts and transactions in a blockchain construct a social network. Transactions reflect connections between accounts. As shown in Fig. 3, a straightforward partition policy leads to a large number of cross-shard TXs. In contrast, taking the advantage of community properties, the number of cross-shard TXs can be reduced. In a blockchain sharding system, each shard is in fact a community consisting of multiple accounts. In other words, if we can detect a number of communities across the system and map each community to a designated shard, the number cross-shard TXs are possibly reduced. Based on the principle described above, we can improve the performance of the sharding system by devising a community-detection algorithm.

The community detection in social networks has been studied over the past few years. Representative effective community-detection algorithms include label propagation [20], Louvian algorithm [21], and Infomap [22]. However, those state-of-the-art community detection algorithms cannot be directly exploited to solve the proposed account partition problem (7). This is because classical community-detection algorithms do not restrict the size of a community, thus they may lead to very large communities. Although the cross-shard TX workload can be maintained in a low level by using an effective community-detection algorithm, they may also lead to a high index of workload imbalance. Therefore, based on the conventional framework of community-detection methods, we propose a novel community-aware account-partition algorithm, which is then presented in the following subsection.

B. Constrained Label Propagation Algorithm (CLPA)

Label propagation algorithm (LPA) is a lightweight community detection algorithm. It has high efficiency on community detection. LPA has been adopted to solving the balanced k-way partition problem [18]. In original LPA, each vertex is initially assigned a label randomly. Then its label is iteratively

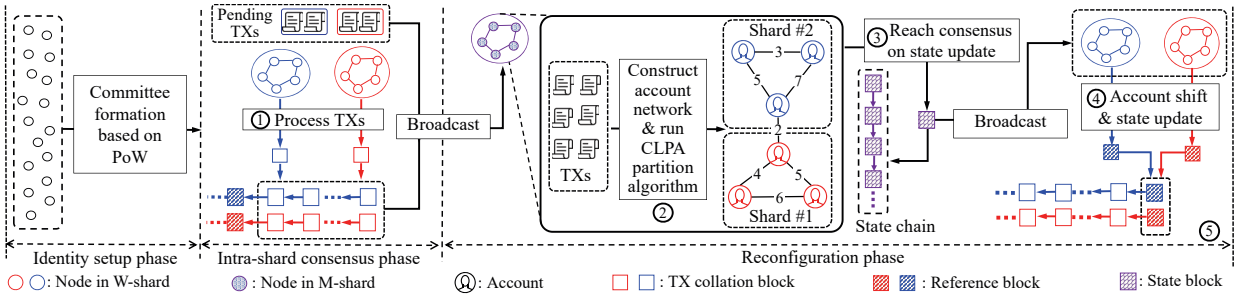


Fig. 4. The overview of the proposed *Transformers* sharding protocol illustrated in an epoch. Note that each epoch includes multiple rounds of intra-shard consensus. Thus, a certain number of collocation blocks will be generated in each epoch.

updated according to the weight of its neighbors' labels until convergence. Based on original LPA, we propose the constrained label propagation algorithm (CLPA) to solve the formulated account-partition problem.

The CLPA algorithm takes the account network $G(V, E)$, the maximum number of iterations τ and the threshold of updating times ρ as inputs. In the end, CLPA outputs the account-partition results $V_1, V_2, ..., V_K$ where $V_i (i \in [K])$ denotes the set of account addresses assigned to shard i. Now we elaborate the algorithm. In initialization phase, $l(i)$ is firstly utilized to denote the label of vertex $i \in [N]$, which is assigned an initial value of $l(i)$ according to its current shard ID. Then, each vertex traverses its neighbors, and computes a score by invoking the score function $s(i, k)$ to update its new label. We use $\mathrm{NB}(i)$ to represent the neighbor vertexes of vertex i and $\mathrm{K}(i) = \{l(j),\ j \in \mathrm{NB}(i)\}$ to denote the label set of all its neighbor vertexes. For each iteration, the new label of vertex i is represented by $\arg\max_{k \in \mathrm{K}(i)} s(i, k)$. The score function $s(i, k)$ is then defined as follows.

$$s(i,k) = \sum_{j \in \mathrm{NB}(i)} \frac{e_{i,j} \cdot \delta(l(j),k)}{\sum_{h \in \mathrm{NB}(i)} e_{i,h}} (1 - \beta \frac{W_k}{\min_{h \in [K]} W_h}), \quad i \in [N], k \in \mathrm{K}(i), \tag{8}$$

where $\delta(l(j), k)$ is an indicator function. Only when $l(j)$=k (i.e., the label of vertex j is equal to k), $\delta(l(j), k)$=1; otherwise, $\delta(l(j), k)$=0. In addition, $W_k (k \in \mathrm{K}(i))$ denotes the total weight of edges associated with label k. The edge weight can be computed and updated incrementally according to Eq. (5). Parameter $\beta \in [0, 1]$ is a tunable coefficient that measures the penalty of label k's edge weight. A lower value of β leads to lower cross-shard TX workload but a higher workload imbalance index. Through Eq. (8), the proposed algorithm assigns a score of each shard corresponding to a specified account address $i \in [N]$. If a community's label (i.e., shard ID) is with a large size, the algorithm can assign a low score such that the size of this community is restricted.

The convergence properties of LPA have been discussed in [20]. In contrast, during the updating phase of the proposed CLPA, each vertex (i.e., account address) may frequently shift its community between a pair of shards. This behavior would prevent the algorithm from terminating. Therefore, we set a maximum number of iteration τ for CLPA. CLPA algorithm keeps executing in an iterative manner no matter whether it reaches a convergence or not within the given τ iterations. Each vertex is allowed to update its label at most ρ times. Thus, the computing complexity of CLPA is $O(N \cdot \tau)$.

V. Sharding Protocol Design

To support the proposed account-partition mechanism, we design an elastic sharding protocol, named *Transformers*, for the account-based blockchain state sharding.

A. Protocol Overview of Transformers

The overview of the proposed elastic sharding protocol is illustrated in Fig. 4. The proposed sharding protocol runs in *epochs*. Each epoch is defined as a fixed rounds of consensus. There are two types of shards existing in the proposed protocol, i.e., the *worker shard* and the *main shard*, which are defined as follows.

- **Worker shard** (shorten as *W-shard*). A worker shard is in charge of processing transactions and generating a certain number of *collation blocks* in each epoch.
- **Main shard** (shorten as *M-shard*). The main shard is responsible for running the proposed account-partition algorithm, and maintaining the latest state of whole blockchain ledger through generating a *state block* in every epoch.

Similar to existing sharding systems [1]–[4], our protocol adopts the classic PBFT [23] as intra-shard consensus protocol. We also adopt the *relay transaction* mechanism [7] to process the cross-shard TXs.

Fundamental steps of protocol. We present the major steps of the proposed protocol using Fig. 4. Those phases are briefly presented as follows.

- Step ①: W-shards collect newly arrived transactions and generate *collation blocks* in parallel. Then, these collation blocks and the pending TXs are synchronized to M-shard.
- Step ②: M-shard constructs the account network and conducts account partition. M-shard first establishes the account network by reading the TXs from its local TX

pool. Then, M-shard runs CLPA algorithm to perform the community-aware partition over the account network.

- Step ③ : M-shard reaches consensus on the state of sharding blockchain according to the partition result and generates a *state block*.
- Step ④ : Each W-shard updates its ledger state according to the state block, and generates a *reference block* as the genesis block of next epoch.
- Step ⑤ : Blockchain system refreshes the formation of both M-shard and W-shards. Sharded blockchain system then enters into the next epoch.

B. Identity Setup and Shard Formation

To defend Sybil attacks in each shard, our protocol entails a key phase, named the *identity setup* phase. We describe its functionality as follows. Before Step ① , all blockchain nodes need to solve a hash-based puzzle to establish their identities for committee formation. Referring to related studies [2], [3], at the beginning of each epoch, an unpredictable and unbiased random string *Epoch Randomness* (ER) is generated via the verifiable random function (VRF) [24]. For blockchain nodes who intend to join the shard committees, they need to find a difficulty-specified Proof-of-Work (PoW) solution which is used to form committee shards. Specifically, each node needs to find a valid *nonce* satisfying the following constraints:

$$\textit{Output} = H(\textit{ER}||\textit{IP}||\textit{PK}||\textit{Type}||\textit{Nonce}) \leq 2^{\gamma - D_{\text{Type}}},$$

where $||$ represents the conjunction operation; and *IP*, *PK* denotes the node's IP address and public key, respectively. Parameter γ is the bit-length of the random output of oracle $H(\cdot)$, while D_{Type} is the predefined mining difficulty. The field *Type* denotes which type of shards the node intends to join:

$$\textit{Type} = \begin{cases} \text{'M'}, \text{ if the node intends to join M-shard;} \\ \text{'W'}, \text{ if the node intends to join W-shard.} \end{cases}$$

Since M-shard needs fewer nodes than W-shards, the mining difficulty to become a valid member of M-shard D_{M} is bigger than that of W-shard D_{W}. We use m and n to denote the number of nodes in a M-shard and a W-shard, respectively. Without loss of generality, we also assume all W-shards have the same number of blockchain nodes. Thus, we set $D_{\text{M}} = D_{\text{W}} + \log_2 \frac{K \cdot n}{m}$. Besides, D_{M} and D_{W} can be changed according to real-time mining hashrate to keep the number of nodes stable within a fixed protocol window. To prevent malicious nodes from only establishing identities in M-shard and W-shards, the proposed protocol allows nodes to mine for two identities simultaneously. That is, a node i can fill the field *Type* with both 'M' and 'W'. If the lottery-drawing ticket $\textit{Output}_i \in (2^{\gamma - D_{\text{M}}}, 2^{\gamma - D_{\text{W}}}]$, node i joins the W-shard; if $\textit{Output}_i \in [0, 2^{\gamma - D_{\text{M}}}]$, node i joins the M-shard. Our design is unbiased for all nodes since they can choose the type of shards they intend to join. The M-shard and W-shards can be formed following a specified sequence, such as the sequence of solving the PoW hash-based puzzle.

C. Consensus on Ledger State Update

In Step ② , M-shard leverages the collected TXs to construct the account network and then performs account partition. In Step ③ , the partition results yielded by CLPA algorithm and the updated ledger state need to be recorded in the state chain for verification. It is insecure to delegate these two most fundamental operations of Step ② and Step ③ to a third party, since the third party could be corrupted by malicious adversary. This is the reason why our protocol introduces M-shard. On the other hand, the coordination between the W-shards and the M-shard is described as follows.

Consensus on state update in M-shard. The M-shard needs to collect all the collation blocks and the pending TXs sent from all W-shards in every epoch. Accordingly, M-shard continuously updates the account state through these TXs and collation blocks. At the reconfiguration phase, M-shard runs the account-partition algorithm locally. Later on, through running the PBFT consensus protocol, M-shard reaches consensus on the updated state. Finally, M-shard broadcasts the consensus result, i.e., the updated ledger state, to all W-shards.

We call the block generated by M-shard the *state block*, which is devised to record the state-update results that can be verified by other shards. Fig. 5 illustrates the data structure design of the state block. Let $\{B_1, B_2, .., B_K\}$ denote the latest collation blocks stored on the longest legal chain of the K W-shards, respectively. Then, $H(B_1||B_2||..||B_K)$ will be recorded in the header of a state block, aiming to reach a consistent view of these K W-shards. For the account-partition result, instead of recording its hash directly, we propose a practical design, i.e., the *Mappings* shown in Fig. 5. M-shard constructs a modified Merkel Patricia Tree (MPT) [9] to store the mappings between account addresses and all W-shards. Such mapping item consists of an account address, and the ID of the account's new destination shard. Through recording such the root hash of the partition results in the header of a state block, a Merkle path that can trace an account's shifting is formed. Such the Merkle path and the root hash of account-partition results can be served as a proof of the account's shifting in a specified epoch. Other blockchain nodes can trace the shifting history of any specific account using the merkle-path proof easily. Furthermore, the ledger state of all W-shards after their reconfiguration is also required to record in the header of a state block, aiming to help W-shards synchronize and verify the whole ledger's state. A leaf node of the *modified MPT of ledger state*, i.e., the *account state*, includes shard ID (i.e., the ID of the shard where this account locates at), balance, nonce and other attributes.

State update of W-shards. After receiving the partition results from the M-shard, W-shards need to verify the results and update their local ledger states. W-shards bootstrap their state reconfiguration through reading from state block. Following the partition results, each W-shard $k \in [K]$ can identify the accounts that are planed to be moved to other W-shards. Those accounts are presented using a *stale account set* V_{out}^k. In the same way, W-shard k can construct a *fresh account set*

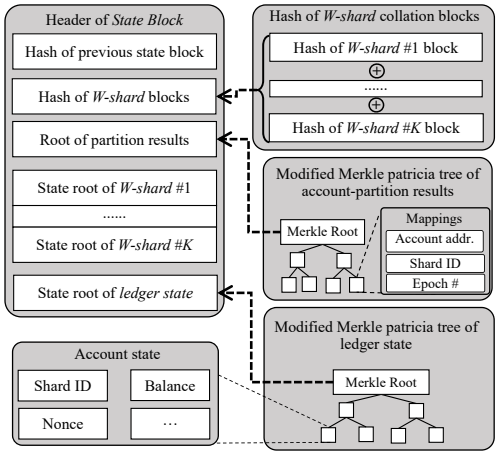


Fig. 5. Data structure design of the *state block* generated by M-shard.

V_{in}^k in which the elements denote the newly added accounts for W-shard k. Then, W-shard k notifies the states of those stale accounts recorded in V_{out}^k together with their Merkle path proofs to their associated destination W-shards. W-shard can also actively request the states of new accounts recorded in V_{in}^k from other W-shards. The correctness of an account's states can be verified by using the associated Merkle path proof and the state root hash recorded in the latest W-shard's collation blocks. This verification is with a time complexity $O(1)$ according to the property of MPT.

After collecting all account information it needs, each W-shard creates a particular *reference block*, which is used to store the latest state of accounts associated with a specific W-shard. The states of all accounts are required to store in a MPT, and the root hash of such account states will be recorded in the reference block. After reaching consensus, the new reference block will be added to the W-shard's local chain. Reference blocks can help new W-shard nodes synchronize the entire ledger state quickly as the initial world state for the next epoch. At the beginning of next epoch, the new shard nodes only need to download the latest ledger states and verify them by referring to the reference block generated in previous epoch.

D. Overhead Analysis and Incentive Mechanism

Overhead of W-shard. Let $|B|$ denote the size of blockchain ledger. If the ledger is divided into K disjoint parts, each W-shard stores a size $O(\frac{|B|}{K})$ of state data. Because the proposed protocol refreshes the account's location across all shards periodically, the shard ID of an account can not be obtained as the conventional way. Thus each W-shard is required to store the latest mappings from accounts to all W-shards. W-shards can update the mappings incrementally according to the partition results of every epoch. The cost of such mapping storage is $O(N)$, where N is the number of total accounts existing in the system. Using mappings, W-shard can search the shard ID of any account with a time complexity $O(1)$. Given that the storage of each account address costs 20 bytes [9], the storage cost of the mappings is negligible for a sharding blockchain system.

Overhead of M-shard. M-shard is required to store the whole state ledger, thus its ledger storage overhead is $O(|B|)$. For state block, its header contains a number K+4 of hash fields, each of which is 32 bytes in size. For the account-partition result, the M-shard nodes need to store the mappings. We can use $\lceil \log_2 K \rceil$ bits to denote the shard ID of an account. Hence, the overhead of each state block is $(K+4)\cdot 32 + S\cdot(20+\frac{\lceil \log_2 K \rceil}{8})$ bytes, where S is the number of accounts which need to be shifted in the current epoch.

Incentives. Since the M-shard maintains the global state of sharded system and is responsible for state update of all W-shards, M-shard consumes more blockchain resource. In *identity setup* phase, blockchain nodes can choose the type of shards they intend to join according to their own network resources and computation resources. The reward for M-shard nodes should be set higher than the W-shard nodes. Because the system throughput is closely related with the account-partition results, the reward of M-shard should be positively correlated to system throughput in order to motivate the M-shard nodes to act honestly. The design of such an incentive mechanism is left as our future work.

VI. Security Analysis of Proposed Protocol

A. Security of M-shard

Since the identity setup for M-shard is based on PoW, the security degree of M-shard correlates with the hashpower of malicious nodes. Let h_p and h_q denote the hashpower of honest nodes and malicious nodes, respectively. We assume that a proportion α_1 of the honest hashpower is mining for M-shard identities, a proportion α_2 of the honest hashpower for W-shard identities, and a proportion α_3 for the both identities. Malicious nodes only mine for the M-shard or W-shard to establish more Sybil identities. Thus we can calculate the probability that a node in M-shard is malicious, denoted by f_{M}, as follows.

$$f_{\mathrm{M}} = h_q \cdot [h_q + (\alpha_1 + \alpha_3)\cdot h_p]^{-1}. \tag{9}$$

For example, we assume the malicious nodes control 20% of the total hashpower. When $\alpha_1 = 0.1, \alpha_2 = 0.1, \alpha_3 = 0.8$, we have $f_{\mathrm{M}} = 21\%$. The hashpower of malicious nodes is not amplified so much, when α_3 is large. Therefore, our *Transformers* protocol encourages honest nodes to participate in the PoW-based identity setup for both types of shards simultaneously, aiming to enhance the security of M-shard.

PBFT consensus protocol can tolerant $1/3$ fraction of byzantine nodes. Thus, the failure probability of M-shard, denoted by P_{MF}, can be formulated as:

$$P_{MF} = \sum_{i=\lfloor m/3 \rfloor+1}^{m} \binom{m}{i} (f_{\mathrm{M}})^i \cdot (1-f_{\mathrm{M}})^{m-i}, \tag{10}$$

where m is the number of nodes in the M-shard. We can evaluate the security degree of M-shard by combining Eq. (9) and Eq. (10), and choose an appropriate M-shard size m

to ensure its high security. What's more, the M-shard may suffer from DDoS attacks. However, such DDoS attacks can be defended by using sophisticated state-of-the-art solutions working in the network layer and transportation layer.

B. Correctness and Security of W-shard

Theorem 1. *(Correctness) Suppose that the M-shard has no more than $1/3$ fraction of malicious nodes, all the honest nodes in one W-shard have the identical and correct system view after updating shard ledger.*

Proof. When M-shard nodes update the ledger state of all W-shards according to the account-partition results, those M-shard nodes will record the latest state of whole system ledger in the state block. Then, M-shard can reach unique consensus on the state block when the fraction of malicious nodes in M-shard is lower than $1/3$. Next, W-shards update their local ledger states according to the account-partition results and construct the modified MPT of ledger state. The root of the constructed ledger-state tree is identical as it is recorded in the state block. It is impossible to construct two state trees with the same root hash value. Thus, the consistency of the updated ledger state in W-shards can be guaranteed. □

Theorem 1 guarantees that W-shards can get the correct ledger state in reconfiguration phase. However, W-shards may still suffer from byzantine attacks. Following the same way to derive Eq. (9), the probability that a node in a W-shard is malicious, denoted by f_W, can be expressed as $f_{\mathrm{W}} = h_q \cdot [h_q + \alpha_2 \cdot h_p + \alpha_3 \cdot (1 - 2^{D_{\mathrm{W}} - D_{\mathrm{M}}}) \cdot h_p]^{-1}$. Let random variable N_c denote the number of malicious nodes among the total $K \cdot n$ W-shard nodes. Recall that n is the number of nodes in a W-shard. Thus, we get the probability distribution of N_c as follows:

$$P(N_c = n_c) = \binom{K \cdot n}{n_c} \cdot (f_{\mathrm{W}})^{n_c} \cdot (1 - f_{\mathrm{W}})^{K \cdot n - n_c}. \quad (11)$$

Given $K \cdot n$ nodes including n_c malicious ones. If we first select n nodes randomly from all nodes. Then, the probability that extracts a number $i > 0$ of malicious nodes from those n nodes can be expressed as:

$$h(K \cdot n, n_c, n, i) = \frac{\binom{n_c}{i} \cdot \binom{K \cdot n - n_c}{n - i}}{\binom{K \cdot n}{n}}. \quad (12)$$

We then depict W-shard's security through Theorem 2.

Theorem 2. *Let the random variable X_k ($k \in [K]$) denote the number of malicious nodes in W-shard #k. Given N_c, the joint distribution of random variable $\{X_1, X_2, ..., X_K\}$ can be written as:*

$$P(X_1 = x_1, ..., X_K = x_K | N_c = n_c) = \frac{\prod_{j=1}^{K} \binom{n}{x_j}}{\binom{K \cdot n}{n_c}}. \quad (13)$$

Proof. The equation (13) can be directly derived from the theory of hypergeometric distribution [25], [26]. □

Using Theorem 2, we can formulate the failure probability of a W-shard as follows.

$$P_{WF} = \sum_{n_c=0}^{n} P(N_c = n_c) \cdot P(\textit{Failure} | N_c = n_c)$$
$$= 1 - \sum_{n_c=0}^{n} \sum_{x_1=0}^{\lfloor n/3 \rfloor} ... \sum_{x_K=0}^{\lfloor n/3 \rfloor} \prod_{j=1}^{K} \binom{n}{x_j} (f_{\mathrm{W}})^{n_c} \cdot (1 - f_{\mathrm{W}})^{K \cdot n - n_c}. \quad (14)$$

The problem is that it is difficult to calculate the probability (14) directly since the computation complexity is $O(n^K)$. Thus, we evaluate such probability by calculating its upper bound as follows:

$$P_{WF} \leq \sum_{k=1}^{K} P(X_k > \lfloor \frac{n}{3} \rfloor) = K \cdot \sum_{i=\lfloor \frac{n}{3} \rfloor + 1}^{n} \binom{n}{i} (f_{\mathrm{W}})^{i} \cdot (1 - f_{\mathrm{W}})^{n-i}. \quad (15)$$

Eq. (15) indicates the worst case of W-shard's security level, which varies following parameters n, K and f_{W}. We then describe the security of the blockchain system under the proposed protocol through the perspective of a so-called (ϵ_m, ϵ_w)-secure, where $\epsilon_m > 0$ and $\epsilon_w > 0$. Its definition is given as follows.

Definition 1. *We call the blockchain sharding system is (ϵ_m, ϵ_w)-secure if $P_{MF} \leq 2^{-\epsilon_m}$ and $P_{WF} \leq 2^{-\epsilon_w}$.*

When $m = n = 50$, $f_{\mathrm{M}} = f_{\mathrm{W}} = 0.1$ and $K = 8$, we get $P_{MF} = 3.8 \cdot 10^{-6} < 2^{-18}$ and $P_{WF} = 3 \cdot 10^{-5} < 2^{-15}$. Thus we say that the sharding system is $(18, 15)$-secure under the current settings. Using such (ϵ_m, ϵ_w)-secure description, blockchain developers can evaluate the security level of their blockchain sharding system via estimating the hashpower of malicious nodes and taking parameters m, n and K into account simultaneously.

C. Security of Ledger State in State Block

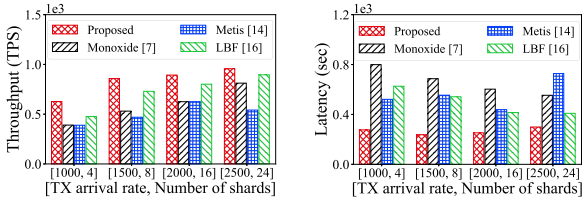
When refreshing the shard ID of an account, a malicious node may try to modify the balance of an account. However, the malicious behaviors tampering the account balance can be easily detected by honest nodes through the Merkle root of ledger state. On the other hand, a malicious M-shard node may collude with W-shards for delaying the transactions of a victim account. Since the formation of W-shards is unpredictable, the malicious M-shard node is not possible to collude with W-shard nodes beforehand to put the victim accounts into a specified target W-shard.

VII. Performance Evaluation

A. Simulation Settings

We firstly develop a transaction-driven blockchain-sharding simulation using Python. We then evaluate the proposed sharding protocol using the simulation.

Datasets. We extract 8 million TXs from the first 2 million historical Ethereum blocks [9]. *Transformers* is evaluated by replaying the first 3 million TXs.



(a) Throughput (b) Latency

Fig. 6. Throughput and transaction's confirmation latency under various combinations of TX arrival rate and the number of shards.

Transaction processing. Each collation block yielded by W-shards can include up to 2000 TXs. In our simulations, a client is exploited to continuously feed sequential TXs to the sharding system under a specific TX arrival rate. Each shard stores the arrived TXs that have not been confirmed yet in a queue (i.e., TX pool).

Baselines. We compare *Transformers* with the following three state-of-the-art methods. *Monoxide* [7] is a classic sharding protocol devised for the account/balance-based sharding blockchain. *Metis* [17] is a well-known K-way partition algorithm which can partition the network into K parts with almost equal vertex sizes while minimizing the weight of cutting edges. *LBF* (Load-Balance First) [19] emphasizes the goal of workload balance, but ignoring to handle cross-shard TXs.

Other parameter settings. Referring to existing studies [8], [23], the consensus latency of PBFT is set to 10 seconds approximately. The length of each epoch is set to 100 seconds around. Our simulation involves totally 10 epochs. By default, β, τ and ρ are set as 0.5, 100, and 50, respectively.

B. Throughput (measured in TPS)

We first evaluate throughput measured in TXs per second (TPS), while varying the number of shards and TX arrival rates. As shown in Fig. 7(a), we fix the TX arrival rate as 2500 TXs/Sec and vary the number of shards from 4 to 40. We observe that the system throughput increases following the number of shards, and converges when the number exceeds 32. The proposed *Transformers* always exhibits highest throughput. For example, *Transformers* achieves 822 TPS with only 6 shards, while Monoxide needs 24 shards to reach the same TPS. When the number of shards is 40, the TPS of *Transformers* and LBF are both around 950, while Monoxide and Metis only achieve 850 and 660, respectively. Taking the advantages of *Transformers*, the throughput is 34.8% higher than that of Monoxide on average. In Fig. 7(c), we set K as 12 and vary the TX arrival rate from 1000 TXs/Sec to 2500 TXs/Sec. With the increase of TX arrival rate, system throughput of baselines do not always increase. In contrast, *Transformers* always outperforms others and keeps high throughput. For example, with a 2000 TXs/Sec arrival rate, *Transformers* achieves 887 TPS while Monoxide, Metis and LBF hit 625 TPS, 511 TPS and 662 TPS, respectively.

For a more comprehensive comparison, we also present the throughput performance under different parameter combinations of TX arrival rate and the number of shards in Fig. 6(a). The result shows that *Transformers* still outperforms others in terms of throughput.

C. TX Confirmation Latency

We then measure the average TX confirmation latency of different methods under the parameter settings same with that of throughput evaluation. As demonstrated in Fig. 7(b), Fig. 7(d) and Fig. 6(b), under various configurations, the proposed sharding protocol always maintains low confirmation latency. In Fig. 7(b), we observe that latency shows a decreasing trend following the increasing number of shards. In particular, we also see that the proposed protocol shows 50% around lower latency than other baselines. In Fig. 7(d), when the TX arrival rate increases from 1000 TXs/Sec to 2500 TXs/Sec, the proposed *Transformers* protocol maintains almost half confirmation latency comparing with other three baselines.

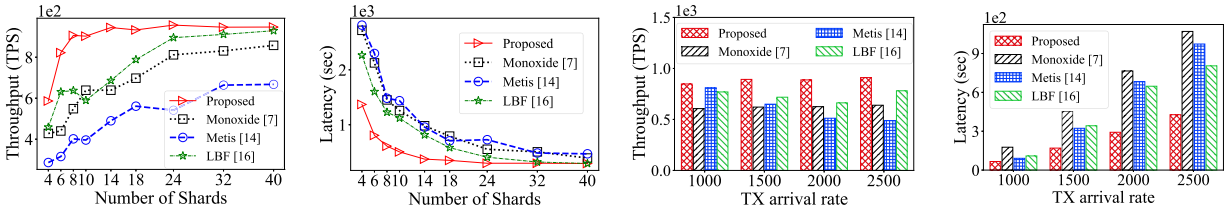
D. TX-Pool Size, Cross-shard TX Ratio and Workload Balance

In the following two groups of simulations, we fix the number of shards to 8, and set the TX arrival rates to 1000 TXs/Sec and 2000 TXs/Sec, respectively. As illustrated in Fig. 8(a), the queue size of TX pool under all methods keeps growing when new transactions keep injecting to the blockchain system. We also observe that *Transformers* maintains a more stable and shorter queue than Metis, Monoxide and LBF.

We are also curious about the performance of throughput, latency, cross-shard TX ratio and the workload balance among all shards when keep injecting a large number of TXs into the TX pool. Thus, we dive deep into more insights. From the results shown in Fig. 8(b), we see that proposed *Transformers*, Metis and LBF can produce fewer cross-shard TXs than Monoxide. Though Metis results in the fewest cross-shard TXs, its other performance metrics are almost the worst as shown in Fig. 7-Fig. 9. The reason is that Metis does not consider the workload balance among shards. Although LBF yields the second highest throughput, it generates the second highest cross-shard TX ratio. Thus, the workload balance and the cross-shard TX ratio are a pair of tradeoff in the context of blockchain state sharding. In contrast, the proposed *Transformers* protocol can maintain a low ratio of cross-shard TXs and have the highest throughput among all methods.

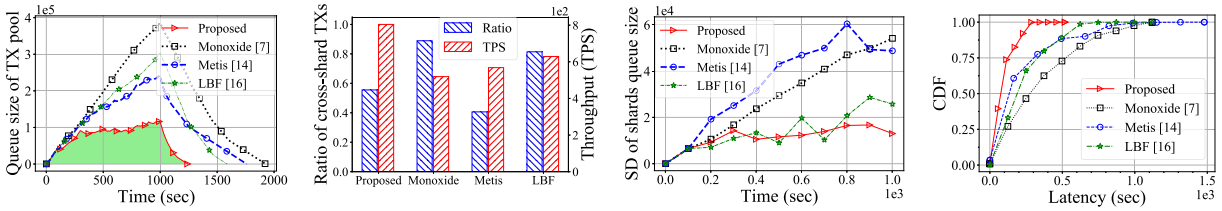
Next, to evaluate the workload-balance metric of different methods, we study the Standard Deviation (SD) of the shard's local queue size across different epochs. Fig. 8(c) shows that proposed *Transformers* maintains the lowest SD of W-shard's local queue size comparing with other baselines. This result proves that *Transformers* has the best workload balance performance among all methods. In contrast, Metis leads to large fluctuating SDs of shard's queue size. This observation indicates that the workload imbalance index of Metis is large. This is because Metis only aims to reduce the number of cross-shard TXs without considering the workload balance.

Recall that, hot shards degrade the scalability of blockchain sharding system, because the congested hot shards cause large TX confirmation latency. From Fig. 8(b), Fig. 8(c) and Fig. 8(d), we find that workload imbalance not only degrades the



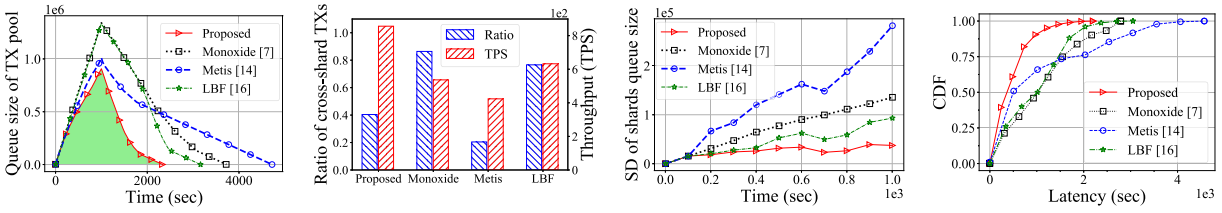
(a) TX arrival rate = 2500 TXs/Sec (b) TX arrival rate = 2500 TXs/Sec (c) Throughput, $K = 12$ (d) Latency, $K = 12$

Fig. 7. Throughput and latency performance while varying the TX arrival rate and the # of shards.



(a) Queue size of TX pool (b) Cross-shard TX ratio vs. TPS (c) Standard deviation of queue size (d) CDF of TX confirmation latency

Fig. 8. Multiple metrics while fixing $K = 8$ and TX arrival rate = 1000 TXs/Sec.



(a) Queue size of TX pool (b) Cross-shard TX ratio vs. TPS (c) Standard deviation of queue size (d) CDF of TX confirmation latency

Fig. 9. Multiple metrics while fixing $K = 8$ and TX arrival rate = 2000 TXs/Sec.

system throughput, but also might notably increase the TX confirmation latency. To prove our speculation, we study the cumulative distribution function (CDF) of TX confirmation latency under different methods. From Fig. 8(d), we observe that up to 70% of TXs are confirmed within 100 seconds under the proposed *Transformers* protocol. For other baselines, within the same latency window, only 24%, 46% and 26% of TXs are confirmed under Monoxide, Metis and LBF, respectively. Furthermore, *Transformers* protocol spends around 519 seconds to confirm all TXs while Monoxide, Metis, and LPA consume 1119, 1479 and 1121 seconds, respectively.

Next, as shown in Fig. 9, we investigate same metrics but with a larger TX arrival rate 2000 TXs/Sec. In Fig. 9, we observe similar results with that of the previous group of simulations shown in Fig. 8. The insights behind those methods are also similar with that shown in Fig. 8. Thus, we omit the associated explanation here due to space limitation.

Through all evaluations, we have the following summaries. i) *Transformers* can reduce the ratio of cross-shard TXs and maintain the workload balance across shards. ii) In addition to cross-shard TX, workload balance is another significant factor affecting the performance of a sharded blockchain.

VIII. Conclusions

This paper aims to achieve a high scalability and balance the workloads across all shards in account-based sharded blockchains. We first formulate the tradeoff between the cross-shard TX ratio and workload balance as a network-partition problem. To solve this problem, we then devise a community-aware account partition algorithm. To support the proposed account-partition mechanism, we carefully design a sharding protocol, i.e., *Transformers*. The trace-driven simulation results demonstrate that the proposed protocol outperforms other baselines in terms of throughput, TX confirmation latency, queue size of TX pool, cross-shard TX ratio, and the standard deviation of shard's queue size.

Acknowledgment

This work is partially supported by fundings from Key-Area Research and Development Program of Guangdong Province (No. 2021B0101400003), NSFC (No. 61902445), the Key-Area Research and Development Program of Shandong Province (No. 2021CXGC010108), Technology Program of Guangzhou, China (No. 202103050004), and CCF-Huawei Populus euphratica forest fund (CCF-HuaweiBC2021004).

REFERENCES

[1] L. Luu, V. Narayanan, C. Zheng, K. Baweja *et al.*, "A secure sharding protocol for open blockchains," in *Proc. of ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*, 2016, pp. 17–30.

[2] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *Proc. of ACM SIGSAC Conf. on Computer and Communications Security (CCS'18)*, 2018, pp. 931–948.

[3] 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. of IEEE Symposium on Security and Privacy (SP'18)*, 2018, pp. 583–598.

[4] Z. Hong, S. Guo, P. Li, and W. Chen, "Pyramid: A layered sharding blockchain system," in *Proc. of IEEE Conference on Computer Communications (INFOCOM'21)*, 2021.

[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 *Proc. of the International Conference on Management of Data (SIGMOD'19)*, 2019, pp. 123–140.

[6] L. N. Nguyen, T. D. Nguyen, T. N. Dinh, and M. T. Thai, "Optchain: optimal transactions placement for scalable blockchain sharding," in *Proc. of IEEE 39th International Conference on Distributed Computing Systems (ICDCS'19)*, 2019, pp. 525–535.

[7] J. Wang and H. Wang, "Monoxide: Scale out blockchains with asynchronous consensus zones," in *Proc. of 16th {USENIX} Symposium on Networked Systems Design and Implementation (NSDI'19)*, 2019, pp. 95–112.

[8] H. Huang, , X. Peng, J. Zhan, S. Zhang, Y. Lin, Z. Zheng, and S. Guo, "Brokerchain: A cross-shard blockchain protocol for account/balance-based state sharding," in *Proc. of IEEE Conference on Computer Communications (INFOCOM'22)*. IEEE, 2022, pp. 1–10.

[9] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[10] S. Zhang and J.-H. Lee, "Double-spending with a sybil attack in the bitcoin decentralized network," *IEEE transactions on Industrial Informatics (TII)*, vol. 15, no. 10, pp. 5715–5722, 2019.

[11] S. Gao, Z. Li, Z. Peng, and B. Xiao, "Power adjusting and bribery racing: Novel mining attacks in the bitcoin system," in *Proc. of the ACM SIGSAC Conference on Computer and Communications Security (CCS'19)*, 2019, pp. 833–850.

[12] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis, "Chainspace: A sharded smart contracts platform," *arXiv preprint arXiv:1708.03778*, 2017.

[13] E. Fynn, A. Bessani, and F. Pedone, "Smart contracts on the move," in *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2020, pp. 233–244.

[14] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *Decentralized Business Review*, p. 21260, 2008.

[15] M. Król, O. Ascigil, S. Rene, A. Sonnino, M. Al-Bassam, and E. Rivière, "Shard scheduler: object placement and migration in sharded account-based blockchains," in *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, pp. 43–56.

[16] W. Fan, R. Jin, M. Liu, P. Lu, X. Luo, R. Xu, Q. Yin, W. Yu, and J. Zhou, "Application driven graph partitioning," in *Proc. of the International Conference on Management of Data (SIGMOD'20)*, 2020, pp. 1765–1779.

[17] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM Journal on scientific Computing*, vol. 20, no. 1, pp. 359–392, 1998.

[18] J. Ugander and L. Backstrom, "Balanced label propagation for partitioning massive graphs," in *Proc. of the sixth ACM international conference on Web search and data mining (WSDM'13)*, 2013, pp. 507–516.

[19] D. P. Williamson and D. B. Shmoys, *The design of approximation algorithms*. Cambridge university press, 2011.

[20] U. N. Raghavan, R. Albert, and S. Kumara, "Near linear time algorithm to detect community structures in large-scale networks," *Physical review E*, vol. 76, no. 3, p. 036106, 2007.

[21] V. D. Blondel, J.-L. Guillaume, R. Lambiotte, and E. Lefebvre, "Fast unfolding of communities in large networks," *Journal of statistical mechanics: theory and experiment*, vol. 2008, no. 10, p. P10008, 2008.

[22] M. Rosvall and C. T. Bergstrom, "Maps of random walks on complex networks reveal community structure," *Proc. of the national academy of sciences*, vol. 105, no. 4, pp. 1118–1123, 2008.

[23] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *Proc. of USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, vol. 99, no. 1999, 1999, pp. 173–186.

[24] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*. IEEE, 1999, pp. 120–130.

[25] R. A. Johnson, I. Miller, and J. E. Freund, *Probability and statistics for engineers*. Pearson Education London, 2000, vol. 2000.

[26] R. L. Scheaffer, M. S. Mulekar, and J. T. McClave, *Probability and statistics for engineers*. Cengage Learning, 2010.