ABChain: A Dynamic Sharding Protocol with Balanced Account Partitioning and Adaptive Brokers for IoT Blockchain

Journal:	*IEEE Internet of Things Journal*
Manuscript ID	IoT-47889-2025
Manuscript Type:	Regular Article
Date Submitted by the Author:	27-Feb-2025
Complete List of Authors:	Li, Tong; Jinan University Wu, Yongdong; Jinan University Weng, Jian; Jinan University Lu, Jiao; Jinan University Huang, Shishi; Jinan University Deng, Weichu; Jinan University
Keywords:	Efficient Communications and Networking < Sub-Area 2: Communications and Networking for IoT, Network Architecture < Sub-Area 2: Communications and Networking for IoT

ABChain: A Dynamic Sharding Protocol with Balanced Account Partitioning and Adaptive Brokers for IoT Blockchain

Tong Li, Yongdong Wu, Jian Weng, Jiao Lu, Shishi Huang, and Weichu Deng

***Abstract*—Sharding enhances Internet of Things (IoT) blockchain scalability by parallel transaction (TX) processing, yet cross-shard TXs and dynamic workload imbalance persist as key bottlenecks. Existing solutions employ brokers that have segmented accounts across multiple shards to efficiently handle cross-shard TXs. Nevertheless, they rely on pre-set optimal accounts as brokers, which may not be feasible in dynamic IoT blockchains where TX workloads are unpredictable. To this end, this paper proposes ABChain, a blockchain sharding protocol for efficient state synchronization with two key components. One is a balanced account partitioning method which allocates accounts to the shards with the highest degree of correlation, and another is an adaptive broker framework which includes dynamical broker selection from historically active accounts, seamless broker update via account splitting/aggregation, and dual broker queues to minimize crossshard TX overhead. The theoretical analysis shows that the former can reduce the number of cross-shard TXs and mitigate workload imbalance, while the latter can efficiently handle cross-shard TXs and adapt to the dynamic blockchain environment. Experimental evaluations demonstrate that ABChain improves throughput by 33.4%, reduces cross-shard TX rate by 52.3%, and lowers confirmation latency by 67.7% compared to state-of-the-art protocols.**

***Index Terms*—Blockchain scalability, sharding, adaptive broker, low cross-shard.**

I. Introduction

With its core features such as decentralization, immutability, transparency, and distributed consensus, blockchain technology provides a secure and trustworthy data-sharing platform for IoT devices. Against this backdrop, a new paradigm that integrates blockchain with IoT, known as IoT blockchain [1], is being investigated. Specifically, IoT blockchain is widely applied in multiple fields, such as agri-food supply chain management [2], intelligent transportation systems in smart cities [3] and energy trading systems for microgrids [4]. However, when faced with the large-scale access of IoT devices and the processing of massive amounts of data, traditional blockchain technology has performance bottlenecks and insufficient scalability, making it difficult to meet the actual needs of IoT.

Sharding technology is regarded as a promising approach to scaling blockchain. By dividing the blockchain network into multiple smaller shards, each shard validates and stores relevant TX data via a specific group of nodes. Sharding technology enables distributed processing of data, significantly enhancing overall processing speed and scalability [5], [6]. However, despite the throughput improvement achieved by sharding, a notable gap remains between the throughput achieved by existing protocols and the potential maximum throughput. A high volume of cross-shard TXs and imbalanced TX loads across shards are key factors contributing to this performance degradation [7]–[9]. To make it even worse, more than 95% of the TXs are cross-shard [10], [11].

One approach to improve the sharding system is to reduce the number of cross-shard TXs by allocating users/accounts with high degrees of association (i.e., those that conduct a large number of TXs) into the same shard. Based on historical TX patterns, BrokerChain [12] and X-Shard [13] partition the account relationship graph with a classical balanced graph edge partitioning algorithm [14]. Community detection algorithms are also employed for partitioning blockchain accounts. A study [15] proposed the Constrained Label Propagation Algorithm (CLPA) for account partitioning. TxAllo [16] leverages the Louvain [17] method to obtain an initial account-shard mapping. In Estuary [18], Community Overlap PRopagation Algorithm (COPRA) [19] is used to partition blockchain users in Unspent TX Output (UTXO)-based system [20]. Some methods focus on the process of account migration. LB-Chain [21] migrates active accounts (i.e., accounts involved in numerous TXs) from heavily loaded shards to lightly loaded shards, thereby achieving workload balancing across each shard. Furthermore, a study [22] proposes a more fine-grained approach that enables the continuous processing of some normal TXs for accounts pending migration during the migration process.

Another approach to improve sharding systems is to handle cross-shard TXs with an "overlapping" structure that facilitates communication across multiple shards. Some approaches, such as Pyramid [23], Saguaro [24] and HieraChain [25], adopt hierarchical architectures. For instance, in Pyramid, high-capability nodes participate in multiple shards, storing their data and enabling cross-shard TX verification without splitting. Some proposals address the issue at the account level by designing cross-shard account entities. BrokerChain executes account partitioning at the protocol layer so that well-partitioned account states can be distributed across multiple shards. These well-partitioned accounts, referred to as brokers, can serve as intermediaries for cross-shard TXs. Following this, BlockEmulator [26] further advances this by open-sourcing a framework that combines CLPA's partitioning

Tong Li, Yongdong Wu, Jian Weng, Jiao Lu, Shishi Huang, and Weichu Deng are with the College of Cyber Security, Jinan University, Guangzhou 510632, China (e-mail: li_tong@1998qq.com; wuyd007@qq.com; cryptjweng@gmail.com; lujiao2023@163.com; sissi_hss@qq.com; weichudeng@stu2024.jnu.edu.cn)

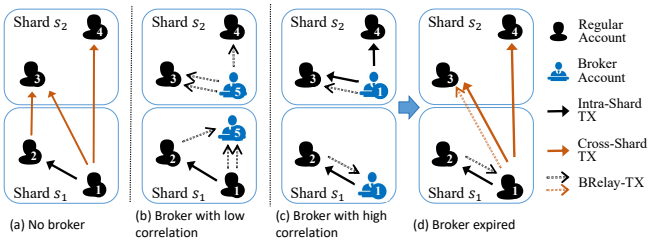


Fig. 1: The effects of different broker accounts on cross-shard TXs. In (b), broker b_5 is split across shard s_1 and s_2, yet it does not engage in TXs with accounts within either s_1 or s_2. Consequently, it primarily serves as an intermediary to facilitate cross-shard TXs by being split. In contrast, in (c), selecting active account v_1 as broker, which has a high degree of association with both s_1 and s_2, can effectively reduce the generation of additional BRelay-TXs. If broker b_1 in (c) expired and BRelay-TX TX_{v_2,v_1} has already been processed, then in (d) BRelay-TX TX_{v_1,v_3} will become a new cross-shard TX.

efficiency with broker-based cross-shard coordination.

As depicted in Fig. 1 (c), since the broker has segment entities (referred to as split-accounts) in multiple shards, the TXs initiated or received by the broker can be processed in the shard where the counter-party account is located. Moreover, the broker can act as an intermediary for cross-shard TXs by splitting a cross-shard TX into two sub-TXs (referred to as BRelay-TXs). This mechanism can reduce the number of cross-shard TXs in IoT blockchains, boost TX throughput, and can be applied to a variety of IoT scenarios, such as edge-computing-based IoT and fog-computing-based IoT scenarios. For instance, in the edge-computing-driven distributed service model, edge servers can serve as host nodes for broker accounts. By leveraging their local computing capabilities, these edge servers can break down cross-shard TXs into intra-shard TXs specific to certain shards.

Motivation: As depicted in Fig. 1, we observe that selecting active accounts as brokers can significantly reduce the generation of additional TXs (compared to Fig. 1 (b) and (c)). Furthermore, in unforeseen IoT blockchain scenarios, IoT devices may join/leave frequently, and broker accounts may become inactive or even invalid. Therefore, it is necessary to update brokers dynamically. Additionally, when brokers update, some inactive broker accounts are converted to regular accounts (marked as expired), and due to TX processing delays, there may still be some BRelay-TXs in the TX pool that potentially become new cross-shard TXs (compared to Fig. 1 (c) and (d)). This calls for a smoother broker update mechanism. Based on the above observations, the pre-set or static brokers [12], [26] cannot meet the complex and dynamic environment in IoT blockchains. Unfortunately, to the best of our knowledge, there are few schemes to select and timely update the broker accounts yet.

This paper presents adaptive brokers to manage broker dynamics and then employs them to build an effective sharding protocol for the IoT blockchain, named ABChain. In ABChain, active accounts are designated as brokers to facilitate TXs across multiple shards. This paper aims to enhance system performance through periodic reconfiguration of accounts and brokers. Firstly, ABChain migrates accounts to their optimal shards based on historical TXs and taking shard workload into account. In addition, active accounts are selected as brokers on a regular basis according to the distribution of historical TXs, in order to replace the existing expired brokers, which are then downgraded to regular accounts.

The contributions of this paper are summarized as follows:

1) We formulate the problem of how account partitioning in sharding systems impacts cross-shard TXs and workload optimization. To tackle this problem, we devise a community-aware shard-balanced account partitioning algorithm.
2) We investigate the impact of broker selection on cross-shard TXs and workload optimization in sharded systems. To address this challenge, we propose an adaptive broker framework comprising three key components: (i) a broker selection algorithm for periodically selecting brokers from the most active accounts; (ii) an account splitting/aggregation mechanism combined with a "dual broker queue" mechanism to enable seamless broker updates; and (iii) a cross-shard consensus protocol to achieve efficient broker state synchronization.
3) Building upon existing open-source projects, we develop ABChain and conduct extensive experiments. Experimental results based on tracking real Ethereum TXs demonstrate that, in comparison to existing blockchain sharding solutions, ABChain outperforms the state-of-the-art baselines in terms of throughput, cross-shard TX ratio, and TX confirmation latency.

The remainder of the paper is organized as follows. In Section II, we present related work and preliminaries of this study. Section III presents an overview of ABChain. Section IV describes the proposed partitioning algorithm. Section V introduces the details of the adaptive brokers. Section VI includes the security analysis. Section VII demonstrates the performance evaluation results. Finally, Section VIII summarizes the paper.

II. Related Work and Preliminaries

A. Blockchain Sharding in Internet of Things

Sharding technology originated from traditional large-scale databases. It is now generally acknowledged that Elastico [5] was the first to introduce sharding technology into blockchain. However, it only sharded the nodes in the blockchain (network sharding), while each node still needed to store the complete ledger (not state sharding). Subsequent sharding technologies [6], [10], [11] built upon this foundation to realize state sharding.

Currently, an increasing number of researchers are interested in enhancing the scalability and performance of IoT blockchains. HMMDShard [27] achieves the adaptive dynamic incremental update of blockchain shards by integrating the Hidden Markov Model (HMM) [28], effectively reducing the cross-shard TXs of all shards. DYNASHARD [29] combines adaptive sharding management, a hybrid consensus method, and efficient state synchronization and dispute resolution protocols to handle dynamic workloads and ensure secure cross-shard TXs. By formulating the long-term tradeoff of throughput and security as a Markov decision process, SmartChain

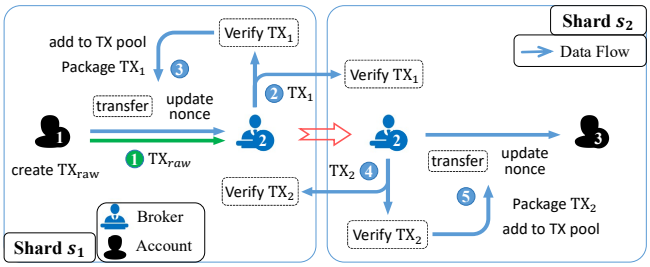


Fig. 2: Account v_1 in shard s_1 transfers funds to v_3 in s_2 through broker b_2, where b_2's state is split and stored across s_1 and s_2.

[30] proposes a dynamic and adaptive sharding framework that aims to make sharding decisions for IoT blockchains with dynamism and heterogeneity. Additionally, blockchain sharding addresses performance bottlenecks in IoT data processing, enabling flexible and efficient device management. Paramart [31] studies the fundamental problem of resource allocation in edge-cloud services by minimizing the system cost of parallel processing mobile services and cost of TXs in sharding blockchain. These solutions enhance the scalability of blockchains in IoT systems, thus ensuring the efficient processing of IoT data and TXs.

B. Cross-Shard Transaction Processing with Broker

BrokerChain achieves atomic execution of cross-shard TXs by introducing the concept of "partial state timelock" within the TX's data structure. As shown in Fig .2, assume account v_1 in shard s_1 sends a TX TX_{raw} to account v_3 in shard s_2 (step ①), with account b_2, whose account is split across shards s_1 and s_2, serving as a broker. Broker b_2 will encapsulate the original TX_{raw} into the first half BRelay-TX TX_1 and broadcast it to shard s_1 (as shown in step ②). Shard s_1 then executes the operation of "transferring funds from v_1 to b_2 with a timelock" for TX_1 and records it on the blockchain (as illustrated in step ③). Upon observing the successful on-chain recording of TX_1 in shard s_1 within a specified timeframe, broker b_2 will broadcast the second half BRelay-TX TX_2 to shard s_2 (as shown in step ④). Finally, in step ⑤, shard s_2 executes the operation of "transferring funds from b_2 to v_3" for TX_2 and records it on the blockchain, thereby completing the execution of a cross-shard TX. If, during the TX process, there is malicious behavior or a timeout from either party, the execution is interrupted: shard s_2 will record TX_1 on the blockchain and send a proof to shard s_1, thereby terminating the execution of the cross-shard TX.

III. Overview of ABChain

In this section, we provide an overview of ABChain, including system model, security model, definition of symbols, and workflow.

A. System Model

The blockchain encompasses multiple shards, with all nodes distributed among them, and each shard maintains its own independent ledger. A myriad of accounts exist within the system, enabling TXs among them. TXs initiated by accounts are dispatched to one or more nodes in the network. Subsequently, nodes follow the gossip protocol to route these TXs to the corresponding shards. Nodes are interconnected through partially synchronous peer-to-peer networks, where messages can reach any other node with optimistic assumptions and exponentially growing timeout periods. Similar to existing blockchain sharding systems, we define an epoch as a fixed length of the system's running time.

Based on the account/balance TX model, ABChain contains two types of accounts: regular accounts and intermediary accounts known as brokers. Regular accounts are assigned to a unique shard, whereas brokers are permitted to participate in TXs across multiple shards, with their account states being jointly maintained by these shards. ABChain consists of two types of shards: Allocation-Shard (A-Shard) and Consensus-Shard (C-Shard). C-Shards are responsible for packaging TXs into TX-blocks and achieving intra-shard consensus at the beginning of each epoch. A-Shard is designed to select new active accounts and partition account states adaptively during each epoch to achieve account state reconfiguration. ABChain incorporates k C-Shards and one A-Shard, employing the Practical Byzantine Fault Tolerance (PBFT) [32] protocol for intra-shard consensus.

B. Security Model

ABChain is a sharding-based blockchain system designed to operate under the Byzantine Fault Tolerance (BFT) model, akin to existing frameworks such as [12], [15], [21], [23]. It comprises multiple nodes that collaboratively maintain network integrity and consensus, even when a subset of nodes exhibits malicious behavior. In contrast to honest nodes that adhere strictly to all protocols, malicious nodes may violate protocols in arbitrary manners, such as denying service, tampering, forging, and intercepting messages. These malicious nodes are controlled by a slowly adapting Byzantine adversary, which may initially control or influence a small number of nodes in the network before gradually expanding its reach. The sets of malicious and honest nodes remain fixed within each epoch and can only change between epochs.

C. Formula Definition

Considering an account-TX graph $\mathbf{G}(\mathbf{V}, \mathbf{E})$, where $\mathbf{V} = \{v_1, v_2, ..., v_n\}$ is a set vertices (account address); and $\mathbf{E}$ is a set of edges (TXs). An edge $E_{v_i,v_j} \in \mathbf{E}$ connects v_i and v_j with weight $\left|E_{v_i,v_j}\right| = w$ means w TXs between those two accounts. We define v_i and v_j to be neighbors of each other if $\left|E_{v_i,v_j}\right| > 0$. Neighboring accounts of v_i are denoted as $NB(v_i)$. The set of k Consensus-Shards is $\mathbf{S} = \{s_1, s_2, ..., s_k\}$, and the Allocation-Shard is s_A. Each account v_i has a state $state_{v_i}$. If v_i is assigned to shard s_i, we denoted it as $P_{v_i} = s_i$, and $P_{v_i \to s_i}$=1, otherwise $P_{v_i} \neq s_i$, and $P_{v_i \to s_i}$=0. A cross-shard edge occurs if $P_{v_i} \neq P_{v_j}$. Newly arrived account v_n are assigned to $P_{v_i} = Hv_i \bmod k$. where $H(\cdot)$ maps addresses to integers. The partition solution for all accounts is denoted as $\mathbf{P} = \{(v_i, P_{v_i}) | v_i \in \{v_1, ..., v_n\}\}$, indicating that each regular account is assigned to only one shard.

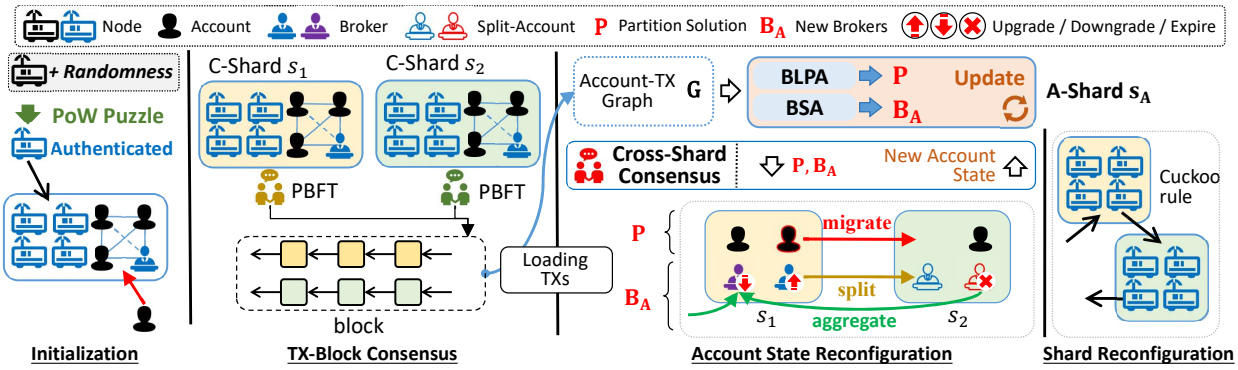


Fig. 3: Workflow of ABChain.

A broker b_i (after account v_i becomes a broker, it is denoted as b_i) has split-states $\{state_{b_i}^{s_1}, ..., state_{b_i}^{s_k}\}$ across multiple associated shards $GAS(b_i) = \{s_1, ..., s_k\}$, where $GAS(\cdot)$ is a function for obtaining an account's associated shards. For a regular account, e.g., v_j, $GAS(v_j) = P_{v_j}$. Each broker has a main shard, which represents the shard where all split-states should be aggregated after the broker expires. The main shard of a broker b_i is P_{v_i}.

The system maintains an active broker queue (denoted as ABQ) and an idle broker queue (denoted as IBQ). Meanwhile, new active accounts are periodically selected to serve as new brokers, denoted as B_A, replacing existing idle brokers.

D. Workflow

The workflow of ABChain, as shown in Fig. 3, includes initialization, TX-Block consensus, account state reconfiguration, and shard reconfiguration.

Initialization: At the beginning of each epoch, consensus nodes (i.e., devices participating in the IoT blockchain network) generate unbiased and unpredictable randomness based on Verifiable Random Functions (VRFs) [10], [11]. To defend against the Sybil attack [33], each node needs to solve a Proof-of-Work (PoW) [20] puzzle. By combining the PoW solution with the VRF-generated randomness and other parameters (such as the node's public key and IP address), the node can obtain a valid identity to participate in the consensus of the current epoch. Before the TX-Block Consensus phase begins, ABChain assigns consensus nodes with valid identities to a certain Consensus shard committee based on the last few bits of their PoW solutions.

TX-Block Consensus: C-Shards continuously receive user TXs and add valid TXs into the TX pool. During each epoch, C-Shards engage in an ongoing sharding consensus process. In each consensus round, the C-Shard leader selects TXs from the pool, packages them into a shard block, and achieves intra-shard consensus through the PBFT algorithm. Simultaneously, A-Shards construct an account-TX graph based on historical TXs and pending TXs.

Account State Reconfiguration: This phase includes two components:

1) Account Partition. To avoid shard workload imbalance and update more active brokers, A-Shard periodically executes the Balanced Label Propagation Algorithm (BLPA) based on the account-TX graph. This algorithm calculates the correlation between each account and each shard and assigns accounts to corresponding shards according to the principle of maximizing correlation, generating an account partition solution. Refer to Section IV.

- Regular accounts are migrated to the destination shard as indicated by the account partition solution. Each C-Shard migrates designated accounts and their pending TXs to target shards.

2) Adaptive Broker. A-Shard executes the Broker Selection Algorithm (BSA) according to the account-TX graph to select accounts with a high number of TXs and involvement in multiple shards, which are designated as the new brokers. A consensus on the account partition solution and new active accounts is reached among shards through a "cross-shard consensus protocol". Each C-Shard updates its account distribution and reloads pending unconfirmed TXs based on the account partition solution and new active accounts. Refer to Section V.

- For new active accounts selected by the BSA, if they agree to become brokers, they will broadcast a permission message. This message allows the current shard to split its state into multiple split-states and send it to different shards. Afterward, these accounts become brokers that can participate in TXs across multiple shards. This process is termed "upgrade".
- Based on their activity levels, brokers are classified as active brokers and idle brokers. Active brokers can serve as intermediaries for cross-shard TXs, while idle brokers cannot. As the system operates, new active accounts will transition to become active brokers, while inactive brokers will temporarily become idle brokers and ultimately expire, reverting to regular accounts.
- For expired idle brokers, all their split-states will be aggregated into a single shard to revert to a regular account. This process is termed "downgrade". During the account splitting and aggregating processes, the corresponding pending TXs will also be sent to the destination shard.

Shard Reconfiguration: At the end of each epoch, ABChain needs to invoke the Cuckoo protocol [34] to update the formation of both A-Shard and C-Shards in order to defend against join-leave attacks [35], [36].

IV. Account Partition

A. Formulation of the Account Partition Problem

For an account-TX graph, if $GAS(v_i) \cap GAS(v_j) = \varnothing$, then the edge E_{v_i,v_j} is considered to be cross-shard edge. We use $Load_w^{intra}$ and $Load_w^{cross}$ to represent the workloads of intra-shard edges and cross-shard edges with a weight of w, respectively. If, after account partition algorithm, v_i and v_j are assigned to the same shard, then the cross-shard edge is converted into an intra-shard edge, and the workload is reduced by $Load^{cross}_{|E_{v_i,v_j}|}$-$Load^{intra}_{|E_{v_i,v_j}|}$.

We use $Load_{s_i}(\mathbf{P})$ to denote the workload of shard s_i under partition solution $\mathbf{P}$, which can be calculated as:

$$\begin{aligned} Load_{s_i}(\mathbf{P}) = &\sum_{v_i \in \mathbf{V}} \sum_{v_j \in \mathbf{V}} P_{v_i \to s_i} \cdot P_{v_j \to s_i} \cdot Load^{intra}_{|E_{v_i,v_j}|} \\ &+ \frac{1}{2} \sum_{v_i \in \mathbf{V}} \sum_{v_j \in \mathbf{V}} P_{v_i \to s_i} \oplus P_{v_j \to s_i} \cdot Load^{cross}_{|E_{v_i,v_j}|} \end{aligned} \tag{1}$$

The total workload $Load_{Tot}(\mathbf{P})$ under partition solution $\mathbf{P}$ can be calculated by:

$$Load_{Tot}(\mathbf{P}) = \sum_{s_i \in \mathbf{S}} Load_{s_i} \tag{2}$$

To balance the workload among shards, it is necessary to avoid allocating a large number of accounts to the same shard. We define the system workload imbalance index, ImB. A larger ImB indicates a stronger imbalance in the system workload. The formula for calculating ImB is:

$$ImB(\mathbf{P}) = max_{s_i \in \mathbf{S}} \left| \frac{Load_{Tot}}{k} - Load_{s_i} \right| \tag{3}$$

When an account-TX graph is given, the objective of account partition is to find a well-designed partition solution $\mathbf{P}$ that can minimize the system workload and achieve the lowest workload imbalance index. We now formulate the account partitioning problem as the following optimization:

$$minF(\mathbf{P}) = \alpha Load_{Tot}(\mathbf{P}) + ImB(\mathbf{P}) \tag{4}$$

where α is a predefined coefficient measuring the weight between the two objective terms $Load_{Tot}$ and ImB.

B. Balanced Label Propagation Algorithm

Referring to the conventional graph-partition problems [14], [37], [38], finding the optimal solution to the formulated problem in Equation 4 is NP-hard [39]. Inspired by existing solutions [15], we found the aforementioned problem bears a resemblance to a community detection problem, which essentially involves allocating appropriate communities (shards) to accounts. The key to the account partitioning problem is to determine the shard to which each account should be assigned, to achieve lower system workload. Based on CLPA and FLPA [40], we propose Balanced Label Propagation Algorithm (BLPA) to solve the account partitioning problem. The pseudocode for the BLPA algorithm is illustrated in Algorithm 1.

Algorithm 1 Balanced Label Propagation Algorithm

Input: $\mathbf{V}$: Account set; $\mathbf{E}$: Edge set; $\mathbf{S}$: Shard set.
Output: $\mathbf{P}$: New account partition solution.

```
define AccQueue
for iter<τ do
    AccQueue enqueue all account from V
    for v_i ∈ AccQueue do
        AccQueue.dequeue(v_i)
        s_h = s_1 // highest scoring shard
        for s_i ∈ S do
            // Score function in Equation 5
            if Score(v_i, s_h)<Score(v_i, s_i) then
                s_h = s_i // update highest scoring shard
        if P_{v_i} ≠ s_h then
            P_{v_i} = s_h //update account partition map
            Recalculate the workload on each shard
            if v_j ∈ NB(v_i) and v_j ∉ AccQueue then
                AccQueue.enqueue(v_j)
return P
```

Firstly, we have designed a new scoring function to calculate the correlation between accounts and shards. It considers both the TX volume between accounts and shards and the workload balance across different shards. The correlation score of account v_i with shard s_i is defined as follows:

$$Score(v_i, s_i) = \frac{\sum_{v_j \in NB(v_i)} P_{v_i \to s_i} \cdot \left| E_{v_i,v_j} \right|}{\sum_{v_j \in NB(v_i)} \left| E_{v_i,v_j} \right|} * \left(\frac{\frac{Load_{Tot}}{k}}{Load_{s_i}}\right)^{\beta} \tag{5}$$

Where k is the number of shards, β is a hyper-parameter that weights the workload balancing of shards. To better balance the workload among shards, in addition to the workload of historical TX (denoted as $Load^{ht}_{s_i}$), $Load_{s_i}$ also includes the workload of pending-TXs in shard s_i (denoted as $Load^{pt}_{s_i}$), $Load_{s_i} = Load^{ht}_{s_i} + Load^{pt}_{s_i}$. Similarly, $Load_{Tot}$ includes both workload of historical TX and pending-TXs of all shards.

Through the scoring function, BLPA partitions accounts to the shard with the highest degree of correlation score. We maintain an explicit queue, $AccQueue$, of accounts that should be considered. If the account label (shard ID P_{v_i}) changes, we add some of its neighbors $v_j \in NB(v_i)$ with $P_{v_j} \neq P_{v_i}$ and $v_j \notin AccQueue$ to the queue. In each step, we dequeue an account from the front of the queue and continue processing all accounts until the queue is empty. This significantly reduces the number of nodes we need to consider, making the algorithm faster.

V. Adaptive Broker

In this section, we delve into the core aspects of ABChain, focusing on the critical processes of broker update and consensus, including the formulation of the broker selection problem (Section V-A), the design of adaptive broker (Section V-B), the broker selection algorithm (Section V-C), the account splitting/aggregation mechanism (Section V-D), the broker queue update under the dual-broker queue mechanism (Section V-E), and the cross-shard consensus protocol (Section V-F).

A. Formulation of the Broker Selection Problem

In an account-TX graph, the more active the account v_i is, the more likely it is to have a large number of neighboring accounts, that is, $NB(v_i)$ with a long length. Although v_i is assigned to a optimal shard through BLPA, it is inevitable that a portion of $v_j \in NB(v_i)$ with $P_{v_j} \neq P_{v_i}$ will still be assigned to other shards. If neither v_i nor v_j is a broker and $P_{v_j} \neq P_{v_i}$, then $GAS(v_i) \cap GAS(v_j) = \varnothing$, and thus the edge E_{v_i,v_j} is a cross-shard edge. Obviously, generally speaking, the more active v_i is, the more cross-shard TXs it sends/receives, i.e. $\sum_{v_j \in NB(v_i) \& P_{v_j} \neq P_{v_i}} |E_{v_i,v_j}|$ is more likely to be large. If, through a selection method, v_i is selected as the broker and it creates a split-account in shard P_{v_j}, then for the original cross-shard edge E_{v_i,v_j} where $P_{v_j} \neq P_{v_i}$, there is now $GAS(v_i) \cap GAS(v_j) = P_{v_j}$. This indicates that E_{v_i,v_j} is transformed into intra-shard TXs within shard P_{v_j}. At this point, the workload will be reduced by $Load_{rd}^{v_i}$:

$$Load_{rd}^{v_i} = \sum_{v_j \in NB(v_i) \& P_{v_j} \neq P_{v_i}} (Load^{cross}_{|E_{v_i,v_j}|} - Load^{intra}_{|E_{v_i,v_j}|}) \tag{6}$$

Based on the aforementioned, selecting the most active accounts as brokers can minimize the cross-shard TXs ratio and reduce the workload significantly.

We define the system load for splitting an account as $Load_{F_s}$, the workload for maintaining a broker as $Load_M$, and the system load for aggregating an expired broker as $Load_{F_a}$. In a sharded blockchain system with an adaptive broker, brokers are updated regularly. We define the system's broker solution as B, where B_{new} represents the newly added brokers during the broker update of the current epoch, B_{exp} represents the brokers that need to expire, and B_{cur} represents the brokers that remain unchanged from the previous epoch to the current epoch. We now formulate the broker selection problem S as the following optimization:

$$\begin{aligned} S(B) = max(&\sum_{v_i \in B_{new} \cup B_{cur}} Load_{rd}^{v_i} - \sum_{v_i \in B_{exp}} Load_{rd}^{v_i} \\ &- |B_{new}| * Load_{F_s} - |B_{cur}| * Load_M - |B_{exp}| * Load_{F_a}) \end{aligned} \tag{7}$$

where, $|B_{new}|$, $|B_{cur}|$, and $|B_{exp}|$ are the length (i.e. number of brokers) of B_{new}, B_{cur}, and B_{exp} respectively.

B. Design of Adaptive Broker

Obviously, operation workloads, $Load_{F_s}$, $Load_M$ and $Load_{F_a}$, are directly related to the selection of brokers. The smaller those workloads are, the more accounts the system would select as brokers and maintain. However, since the cost of those operations (account splitting/aggregation and broker maintenance) is difficult to estimate, it is hard to select the optimal B through a heuristic method using the above Equation 7. In addition, in IoT blockchains, the computing power of hardware devices is also an influencing factor. For example, shards composed of different IoT devices have different network stabilities, which may result in the response speed of broker accounts to TXs being slower in some shards compared to those in other shards. Therefore, we simplify the above problem: 1) we ignore the workload of $Load_{F_s}$, $Load_M$ and $Load_{F_a}$; 2) we set a fixed number n_b of brokers (ABQ with length n_b). Meanwhile, to avoid maintaining idle brokers for an extended period, we set the IBQ to have the same length as the ABQ (IBQ with length n_b). In each epoch, we select the top-n_u active accounts (B_A with length n_u) to update the existing brokers. It is worth noting that here $n_u \geq |B_{new}|$, because the top-ν active account in the previous epoch may include some brokers. Therefore, B_A is equal to B_{new} plus a part of B_{cur}. 3) we only select brokers based on historical TX data, while ignoring other factors such as response time and historical reputation (whether they have ever crashed or acted maliciously). Section V-C details the proposed broker selection algorithm.

Specifically, when an ordinary account is selected and is willing to become a broker, its state needs to be split into all of its associated shards to create split-accounts. Correspondingly, when a broker fails and is ready to become an ordinary account, the split-accounts distributed across various associated shards need to be aggregated to form an ordinary account. Inspired by Estuary, we have designed an account splitting/aggregation mechanism to achieve broker updates, details in Section V-D.

During the process of broker update, some BRelay-TXs that were split by expired brokers (now regular accounts) may still exist in the TX pool and become new cross-shard TXs. They require new brokers for further splitting, as shown in Fig.1 (d), which undoubtedly increases system overhead. To address this issue, we maintain an additional idle broker queue (IBQ) that contains brokers dequeueing from the active broker queue (ABQ). Specifically, the idle broker from IBQ still maintains sub-states in its associated shard, As a result, its BRelay-TXs can be processed directly, thereby avoiding further splitting. It's worth noting that only active brokers can split cross-shard TXs. Therefore, the number of BRelay-TXs in the TX pool that were originally split by an idle broker will not increase but will gradually decrease as TXs are packaged and processed. The broker update steps under the dual broker queue are detailed in Section V-E.

In each epoch, a consensus needs to be reached among all shards regarding the new account states, including the states of the brokers each shard maintains. For security considerations, authorization from the accounts is required before splitting their states. Hence, the cross-shard consensus protocol elaborated in Section V-F is essential to maintain the integrity and security of the sharding system.

C. Broker Selection Algorithm

We propose Broker Selection algorithm (BSA), specifically designed to select most active accounts and their associated shards. The pseudocode for BSA is illustrated in Algorithm 2.

First, based on the account-TX graph $\mathbf{G}$, for each account $v_i \in \mathbf{V}$, the top-n_c neighboring accounts in $NB(v_i)$ with the most TXs are selected as candidate accounts (line 4). v_i then casts a vote for these n_c candidate accounts (line 6). After

Algorithm 2 Broker Selection Algorithm

Input: $\mathbf{V}$: Account set; $\mathbf{E}$: Edge set; $\mathbf{S}$: Shard set; n_c: Number of candidate account; n_u: Number of new active account.
Output: B_A: New active accounts.

1: define $CandidateMap$, Top_{N_u}
2: **for** $v_i \in \mathbf{V}$ **do**
3: define temporary list $Top_{N_c}^{v_i}$.
4: Select the n_c accounts with the largest edge weights from NB(v_i), then add it to $Top_{N_c}^{v_i}$
5: **for** $v_j \in NB(v_i)$ and $v_j \in Top_{N_c}^{v_i}$ **do**
6: $CandidateMap[v_j]$+=1 // vote for candidate
7: **for** $s_i \in \mathbf{S}$ **do**
8: $TotScore$+=$Score(v_i, s_i)$
9: Select the n_u accounts with the highest number of votes from the $CandidateMap$, then add to a temporary list Top_{N_u}
10: Calculate $AvgScore$ through $TotScore$
11: **for** $v_i \in Top_{N_u}$ **do**
12: s_h = s_1 // Init highest scoring shard
13: // B_A add active accounts and its associated shards
14: **for** $s_i \in \mathbf{S}$ **do**
15: **if Score**(v_i, s_i)>$AvgScore$ **then**
16: $B_A[v_i]$.append(s_i)
17: **return** B_A

traversing all accounts, we then accumulate the votes for all candidate accounts. The top-n_u candidate accounts with the most votes are designated as active accounts (line 9). Next, we calculate the score for each account with each shard, sum them to compute a total score, then derive the average account-shard score (line 10). Then, for all $v_i \in B_A$, we determine whether their scores with shard s_i, $s_i \in \mathbf{S}$, exceed the average score. If it does, then we consider s_i as an associated shard of v_i (line 11-16). This method prevents the situation where an active account with low correlation to a shard still needs to create a split-account there.

D. Account Splitting/Aggregation Mechanism

Regular accounts selected by the BSA algorithm are upgraded to brokers through the account splitting operation, while expired brokers are downgraded to regular accounts via the account aggregation operation. We illustrate the proposed account splitting/aggregation mechanism in Fig. 4.

Split $\mathbf{F_S}$: Consider account v_1 with main shard $P_{v_1} = s_1$. It has been selected as an active account with new associated shards $B_A[v_1] = [s_1, s_2, s_3]$, indicating that account v_1 needs to split its state and store the parts in its associated shards s_1 and s_3. Therefore, after getting permission from v_1, s_1 executes the operation of $state_{v_1}^{s_1}, state_{v_1}^{s_2}, state_{v_1}^{s_3} = \mathbf{F_S}(state_{v_1})$ to divide v_1's state $state_{v_1}$ into three parts: $state_{v_1}^{s_1}$ sending to shard s_1, $state_{v_1}^{s_2}$ retaining on shard s_2, and $state_{v_1}^{s_3}$ sending to shard s_3. Upon receiving the split-states, the target shards s_1 and s_3 verify the relevant information and create the split-account of v_1. The details of "get permission from an account" are provided in Section V-F.

Aggregate $\mathbf{F_A}$: Upon receiving the B_A, each shard first collects the permissions of the selected accounts and then constructs an expired broker map B_E, which includes the addresses of expired brokers and their associated shards, details are provided in Section V-F. The split-state of each expired broker is sent to their main shard, which then performs

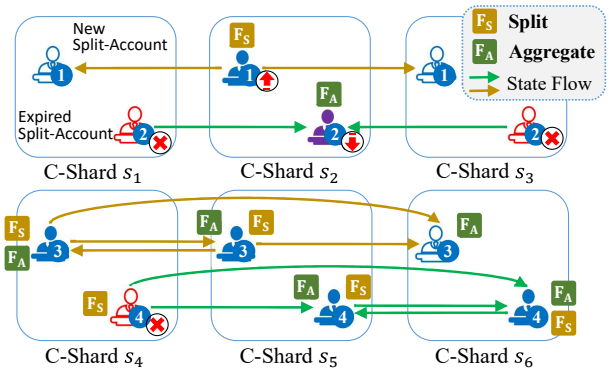


Fig. 4: Example of the account split/aggregate.

account aggregation based on the received split-states. As for b_2, split-account $b_2^{s_1}$ and $b_2^{s_3}$ sending their respective split-states $state_{v_2}^{s_1}$ and $state_{v_2}^{s_3}$ to main shard s_2. Then s_2 executes $state_{v_2} = \mathbf{F_A}(state_{v_2}^{s_1}, state_{v_2}^{s_2}, state_{v_2}^{s_3})$, ultimately downgrading expired broker b_2 to a complete regular account v_2.

In complex scenarios, for instance, broker b_3 is split and stored across shards s_4 and s_5, and needs to be split and stored into shards s_4, s_5, and s_6 according to B_A. To ensure an even distribution of each split-state (such as balance), we further split those split-states. Every split-state is required to split and send to other associated shards. Consequently, both $b_3^{s_4}$ and $b_3^{s_5}$ would need to execute $\mathbf{F_S}$ and $\mathbf{F_A}$, while in shard s_6, only by executing $\mathbf{F_A}$ can the split-account of b_3 be created as $b_3^{s_6}$.

Similarly, b_4, which is stored across shards s_4, s_5, and s_6, needs to be aggregated into shards s_4 and s_5. Both $b_4^{s_5}$ and $b_4^{s_6}$ would need to execute $\mathbf{F_S}$ and $\mathbf{F_A}$. In contrast,, $b_4^{s_4}$ would only need to execute $\mathbf{F_S}$ to send its split-state to shards s_5 and s_6.

E. Broker Update Under the Dual Broker Queue Mechanism

After acquiring new active accounts B_A through BSA algorithm, it is necessary to update the existing broker queues, active broker queue (ABQ) and idle broker queue (IBQ). As shown in Fig.5, the detailed steps are as follows:

Step ① : Firstly, compare B_A with ABQ and IBQ. If a new active account $v_i \in B_A$ already exist in these queues, $v_i \in$ ABQ or $v_i \in$ IBQ, then v_i needs to be removed from their original queue (not dequeued in the traditional sense).

Step ② : New active accounts are enqueued into the ABQ. Their states are then split across all associated shards through an account splitting operation to upgrade them into brokers (such as Fig.4 b_1). The broker removed in **Step** ① is re-enqueued at the end of the ABQ. Brokers located closer to the end of the queue have longer lifecycles, and therefore, re-enqueuing "refreshes" their lifecycles. If the original associated shards of the broker do not match the new associated shards in the B_A, the broker also needs to update its split-account through account splitting/aggregation mechanism (such as Fig.4 b_3 and b_4).

Step ③ : Subsequently, according to the "idle" mechanism settings, dequeue the broker at the head of ABQ and enqueue it into the IBQ queue.

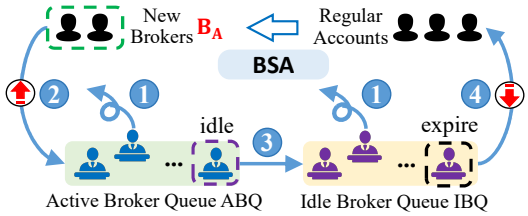


Fig. 5: Update brokers queue according to "dual broker queue" mechanism.

- We set a fixed size n_b for ABQ. When a new broker is enqueued, if the queue size ($|ABQ|$) exceeds the set value n_b, the broker(s) at the head of the queue (number = $|ABQ| - n_b$) will be dequeued and marked as "idle".

Step ④ : Finally, according to the "expired" mechanism settings, dequeue the broker at the head of IBQ and downgraded it to a regular account through the account aggregation operation (such as Fig.4 b_2).

- We set a fixed size n_b for IBQ. When a new ield broker is enqueued, if the queue size ($|IBQ|$) exceeds the set value n_b, the broker(s) at the head of the queue (number = $|IBQ| - n_b$) will be dequeued and marked as "expired". The information of these brokers will be temporarily stored and denoted as B_E for subsequent account aggregation operations.

F. Cross-Shard Consensus

We have designed a cross-shard consensus protocol with broker update permissions, aimed at achieving global consensus for broker updates. As illustrated in Fig. 6, the specific process is outlined below:

Step ① : Based on the account-TX graph, A-Shard obtains new active account B_A by executing BSA. Nodes within A-Shard will validate B_A, and achieve intra-shard consensus to ensure consistency. Upon reaching consensus, A-Shard generates a proof ($proof_A$), encapsulating key data hashes and node signatures from the algorithm execution process, serving as evidence of computation and result correctness.

Step ② : A-Shard combines B_A and $proof_A$ into a message $msg_1^A = [B_A, proof_A]$ and broadcasts it across the entire network, enabling other C-Shards $s_1, ..., s_k$ to access and verify this information.

Step ③ : After a period of network latency, upon receiving msg_1^A, C-Shard s_i verifies its authenticity and integrity and generates a proof ($proof_{s_i}$) through intra-shard consensus.

Step ④ : The selected account v_j provides a permission $pm_{v_j}^{F_S} = [msg_1^A, \sigma_{v_j}]$, where $P_{v_j} = s_1$ and $v_j \in B_A$, and σ_{v_j} is v_j's signature. This indicates that v_j agrees to become a broker and consents to split the state into its associated shards according to B_A.

Step ⑤ : Source C-Shard s_1 initiates account state migration $msg_2^{s_1 \to s_i}$, where $s_i = s_2, ..., s_k$, then

- Utilizing the B_A and permission messages from selected accounts, a group of brokers that have just expired B_E is constructed. B_E includes broker's split-account $b_m^{s_1} \in B_E$ that need to be aggregated to their main shard P_{b_m}, for example, $P_{b_m} = s_2$. $b_m^{s_1}$ provides a permission $pm_{b_m^{s_1}}^{F_A} = [msg_1^A, \sigma_{b_m^{s_1}}]$, and s_1 transmits the state of $b_m^{s_1}$ to shard s_2.

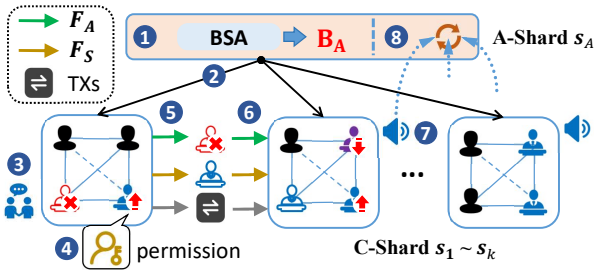


Fig. 6: Basic workflow of the proposed cross-shard consensus protocol.

Step ⑥ : Upon receiving the account state migration messages $msg_2^{s_i \to s_2}$, where $s_i = s_1, s_3, ..., s_k$, the destination C-Shard s_2 validates their authenticity based on the B_A and $\mathbf{P}$. Taking $msg_2^{s_1 \to s_2}$ as an example, Subsequently:

- Based on B_A and the received split-state $state_{v_j}^{s_2}$, s_2 creates a split-account instance for v_j.
- Leveraging B_E and the received expired split-state $state_{b_m}^{s_1}$, $b_m^{s_2}$ executes the aggregation function $\mathbf{F_A}$ to downgrade expired split-accounts back to a regular account v_m. The new state of this account is denoted as $state_{v_m}$.

Step ⑦ : s_2 broadcasts a message $msg_3^{s_2}$ to notify other shards of the new broker state in s_2.

Step ⑧ : A-Shard collects messages $msg_3^{s_i}$ from C-Shards and verifies the correctness of their signatures and contents. Based on the verification results, A-Shard updates the broker queue. At this point, all shards have reached a consensus on the new state of brokers.

VI. Discussion

A. Shard Security

We adopt the same settings as related work [10], [11], [12], and [18], which allows us to guarantee an equivalent level of security under identical security settings. Consequently, utilizing the cumulative hypergeometric distribution function, we determine the upper bound of the system failure probability per epoch, denoted as $Pr[Failure]$, as follows:

$$Pr[Failure] \leq (k+1) \sum_{n_m = \lfloor n_s/3 \rfloor}^{n_s} \frac{\binom{f \cdot n_t}{n_M} \binom{(1-f) \cdot n_t}{n_s - n_m}}{\binom{n_t}{n_s}} \quad (8)$$

where k+1 represents the number of shards (k C-Shards and 1 A-Shard); n_t represents the total number of consensus nodes with valid identities in each epoch; $n_s = \frac{n_t}{k+1}$ represents the number of consensus nodes per shard; $f \in [0, 1]$ represents the ratio of malicious nodes; n_m represents the number of malicious nodes in one shard.

From this, we can conclude that the sharding system is secure as long as the ratio of malicious nodes in each shard does not exceed $\frac{1}{3}$.

B. Account Migrate Security

First, ABChain utilizes the BFT-Ledger Partitioning Algorithm (BLPA) to compute the account partition solution for each epoch. BLPA extends the existing CLPA scheme [15], with convergence guarantees provided in [15] and a maximum iteration limit τ to ensure termination.

Next, based on the partition solution, ABChain migrates accounts along with their unconfirmed TXs to optimal shards. Following [22], we adopt two security properties:

- **Migration Liveness [22]**: Every account migration eventually completes via either success or failure.
- **Eventual Migration Consistency [22]**: All shards eventually agree on the target shard of migrated accounts.

Lemma 1. *The employed account migration scheme achieves migration liveness and eventual migration consistency as long as the source shard and destination shard are not compromised by malicious nodes.*

C. Property Analysis of Broker

If a broker is malicious, a potential attack scenario is as follows: When the first half of a BRelay-TX TX_1 is confirmed in shard s_1, a malicious broker b_2 initiates a double-spend attack by submitting a conflicting TX TX_a in shard s_2. TX_a carries the same signature as the second half of the BRelay-TX TX_2, attempting to get both TXs confirmed in s_2.

Our adaptive broker design is identical to that of [12], thus we employ the same cross-shard TX processing method. Each TX includes a globally monotonic *nonce* field tied to the account state. When a TX is confirmed, the account's nonce is incremented atomically. If TX_a is confirmed in s_2, b_2's nonce is updated, rendering TX_a invalid in any shard due to a nonce mismatch.

D. Broker Update Security

In ABChain, each shard maintains its individual split-account, ensuring that split and aggregate operations do not conflict with those of other shards. Sending the split-state to the destination shard and creating a split-account resembles an account migration. According to Lemma 1, we have:

Theorem 1. *As long as the source shard and the destination shard are not compromised by malicious nodes, the employed account splitting/aggregation achieve migration liveness and eventual migration consistency.*

It is noteworthy that whether an account is willing to split or aggregate its state is a matter of individual preference.

Lemma 2. *As long as the shards are not corrupted by malicious nodes and all accounts promptly and voluntarily provide permission, ABChain can reach consensus on broker updates.*

Proof: At each epoch, ABChain constructs an account-TX graph based on historical TX data and pending TXs, and selects active accounts, i.e. B_A, using the BSA algorithm. The BSA algorithm is executed by the A-Shard and validated through intra-shard consensus, generating a proof. Therefore, the B_A output by BSA is trustworthy, and cannot be forged by other parties. The A-Shard sends the B_A and $proof_A$ to each C-Shard. Within the C-Shards, accounts that are not selected cannot forge broker qualifications because they are not included in the B_A. Other nodes can identify forgers by checking the B_A. Subsequently, the selected accounts provide permission to execute the broker queue update. Finally, all C-Shards send the update result ($msg_3^{s_i}$, $s_i \in \mathbf{S}$) of the account status to the A-Shard, which updates its own account status based on $msg_3^{s_i}$. Consequently, the A-Shard and C-Shards maintain identical update results.

Lemma 3. *As long as the shards are not corrupted by malicious nodes, ABChain can still achieve consensus on broker updates even if the selected accounts refuse to provide permissions.*

Proof: If a selected account $v_i \in B_A$ refuses to provide permission during account split, its account state will not be split and will not be included in $msg_2^{s_i \to s_j}$. For other C-Shards, since they do not receive the split-state of account v_i, they will not create a split-account for v_i. During the broker queue update, v_i will not be promoted to a broker, which can be regarded as the existence of valid new active accounts B'_A =B_A.remove(v_i), where "remove" means remove an element from the map. and B'_A is used to replace B_A to update the broker queue. After the broker queue update, all C-Shards send the update result of the split-state (denoted as $msg_3^{s_i}$) to the A-Shard, which updates its account state based on $msg_3^{s_i}$. Therefore, even if $v_i \in B_A$ refuses to provide permission, the A-Shard and C-Shards can still reach consensus on all account states.

Lemma 4. *As long as the shards are not corrupted by malicious nodes, ABChain can still achieve consensus on broker updates even if the expired brokers refuse to provide permissions.*

Proof: Similarly, for an expired broker $b_j \in B_E$, during account aggregation, if b_j does not provide permission, it indicates that b_j does not want its split-account to be aggregated. For b_j's main C-Shard, since it does not receive the split-state of account b_j, it will not execute the aggregation operation to downgrade b_j to a regular account. During the broker queue update, b_j will not be downgraded to a regular account, which can be regarded as the existence of actual expired brokers B'_E=B_E.remove(b_j), and B'_E is used to replace B_E to update the broker queue. As mentioned earlier, the A-Shard updates its account state based on $msg_3^{s_i}$, $s_i = \{s_1, ..., s_k\}$ sent by all C-Shards. Therefore, even if $b_j \in B_E$ refuses to provide permission, the A-Shard and C-Shards can still reach consensus on all account states. If it is required to solve the problem that expired brokers are reluctant to be downgraded for a long time, a penalty mechanism can be set up to prompt those expired brokers to be downgraded.

In summary, according to Lemma 2, 3, and 4, we have:

Theorem 2. *As long as the shards are not corrupted by malicious nodes, ABChain can reach consensus on broker updates.*

Finally, based on Theorem 1 and Theorem 2, we can conclude that the security performance of the broker update proposed in this scheme is guaranteed.

E. Future Prospects

In cross-shard TX processing, brokers inevitably incur additional workloads. To encourage accounts to actively serve as brokers, it is critical to design an effective incentive mechanism. Current broker selection criteria lack comprehensive consideration of token holdings. Conversely, accounts with frequent TX histories but insufficient balances may introduce liquidity risks; thus, their likelihood of selection should be reduced. Furthermore, IoT Blockchains face unique challenges such as dynamic topology and hardware heterogeneity. Variations in node device capabilities (e.g., computational power, energy constraints) must be addressed in future work. Potential solutions include lightweight consensus protocols and adaptive sharding strategies tailored for resource-constrained environments.

VII. Experiment Evaluation

A. Experimental Setup

We have implemented a prototype of ABChain based on BlockEmulator [26]. The system is written in the Go language. The system runs on a hardware configuration of AMD R7 7840hs CPU@3.8Ghz, 32GB RAM. We evaluate the performance of the system by playing back 1 million historical Ethereum TXs sent by over 100,000 accounts. At the beginning of each epoch, TXs are prepared in chronological order and played back into the blockchain sharding system with a certain arrival rate. These TXs are then assigned to different C-Shards based on the state of their sender account. The code of ABChain is available at https://github.com/Gnotil/ABChain_Code.

Baseline and Metrics: The baseline includes Monoxide [6], CLPA [15], BrokerChain [12] and CLPA_Broker [26]. We investigated the impact of several key system parameters on the metrics of blockchain sharding systems under different modes. The key system parameters are shown Table I. We fixed the number of nodes per shard at n_s = 4, weight factor β = 4 and maximum iterations number τ = 50 in all experiments. The system metrics encompass the Average TX Per Second (AvgTPS), TX Confirmation Latency (TxCLatency), and Cross-Shard TX Ratio (CSTxRatio).

TABLE I: System Parameters.

Parameters	Descriptions
$Selc$	the selection method for broker
k	number of C-Shards
n_b	number of broker (length of ABQ)
T_r	account state reconfiguration interval, unit: second (s)
n_u	number of newly selected active account (length of B_A)
$rate_A$	TX arrival rate, The number of TXs arriving per second, writen as TX/s
$size_B$	maximum TX-block size

B. Baselines

We compared the proposed scheme with baselines. In addition, to verify the effectiveness of our proposed BLPA, we have set up two control schemes, which are as follows: 1) replace CLPA with our BLPA in scheme CLPA_Broker, denoted as BLPA_Broker; 2) replace BLPA with CLPA in our ABChain, denoted as CLPA_ABChain. The parameters are set as follows: k=8, n_b=90 (pertaining to BrokerChain, CLPA_Broker, and ABChain, specifically, n_u=n_b for ABChain), $Selc$=S500 (detailed in Sec. VII-C), $rate_A$=3000 TX/s, $size_B$=2000 TX, and T_r=50 s.

The experimental results are presented in Table II. By comparing row 4 with row 5, and row 6 with row 7, it can be seen that our BLPA can effectively improve the system performance. Moreover, it is evident that ABChain outperforms the baseline in terms of AvgTPS and CSTxRatio. Compared with CLPA_Broker, the AvgTPS of ABChain has increased by 33.4%, the CSTxRatio and TxCLatency have decreased by 52.3% and 67.7% respectively. The BrokerChain scheme, which does not employ periodic account migration, relies heavily on an appropriate intermediary account to reduce the CSTxRatio. When a poor intermediary account is selected (S500), it leads to significant degradation in scheme performance. Similarly, poor intermediary selection has a substantial impact on the CLPA_Broker scheme. In contrast, our scheme demonstrates superior performance due to its ability to periodically update the broker account.

TABLE II: Comparison between ABChain and baselines.

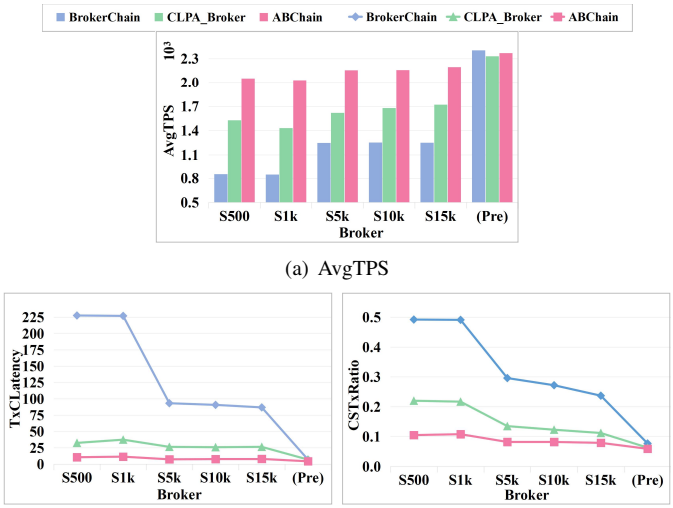
Method	AvgTPS ↑	CSTxRatio ↓	TxCLatency ↓
Monixide [6]	886.365	0.875	260.980
CLPA [15]	2033.970	0.315	47.123
BrokerChain [12]	850.573	0.492	227.585
CLPA_Broker [26]	1525.920	0.220	32.430
BLPA_*Broker*	1678.752	0.204	17.530
***CLPA*_ABChain**	1904.920	0.110	13.658
ABChain	**2051.611**	**0.105**	**10.488**

C. Broker Selection

We investigate the impact of different broker account selections on the existing BrokerChain, CLPA_Broker, and proposed ABChain. We establish 6 distinct sets of brokers, including selecting the most active account from the first 500 TX in the dataset as the broker, denoted as S500, and similarly, we have S1k (first 1,000 TX), S5k, S10k, and S15k. In addition, we also utilize the pre-set broker from the open-source BlockEmulator, denoted as "(Pre)". Note that since the existing schemes do not have an adaptive active account selection algorithm, the above 6 distinct sets of brokers will be pre-set for each scheme. Additionally, when the account state is reconfigured, our scheme will update the brokers adaptively. We set the other parameters as follows: k=8, n_b=n_u=90, $rate_A$=3000 TX/s, $size_B$=2000 TX, and T_r=50 s.

As shown in Fig. 7, as brokers become increasingly superior, the performance of BrokerChain and CLPA_Broker schemes gradually improves. In contrast, our scheme does not rely

on pre-set excellent brokers and outperforms the comparison schemes under various broker selection scenarios. Notably, our scheme demonstrates a significant advantage in cases involving inferior brokers. This indicates that in the absence of prior knowledge, specifically when the future TX situation is unknown and thus the globally optimal active accounts cannot be preselected, periodic updates of brokers are essential for enhancing sharded blockchain systems based on broker mechanisms. Our scheme dynamically selects active accounts to serve as brokers, efficiently reducing the number of cross-shard TXs and reducing system's TxCLatency.



(a) AvgTPS

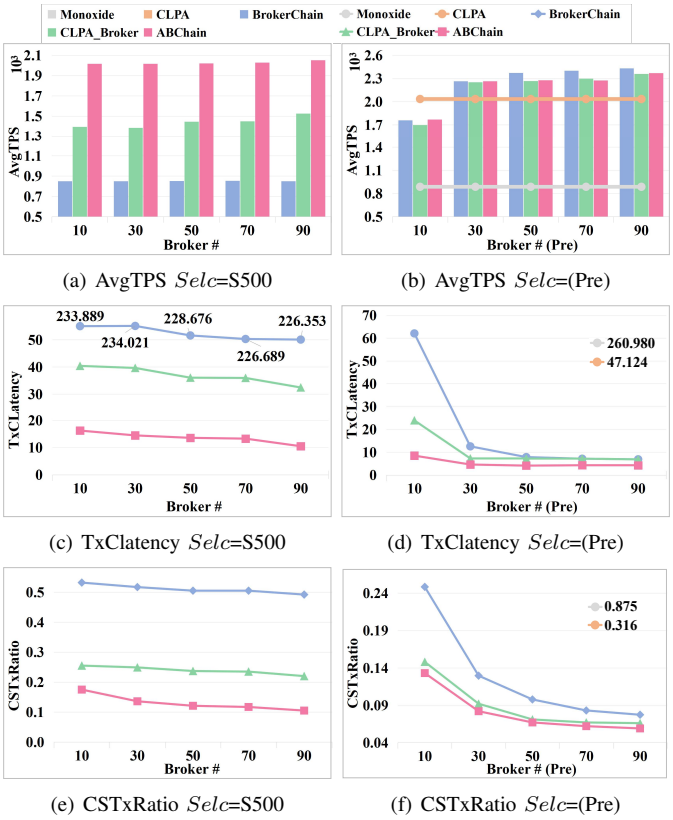
(b) TxCLatency (c) CSTxRatio

Fig. 7: Comparison of different broker selection methods.

D. Broker Number

In this subsection, we compared the impact of different numbers of brokers on system performance. We vary n_b with {10, 30, 50, 70, 90} and set $n_u = n_b$. Other system parameters were configured as follows: k=8, $rate_A$=3000 TX/s, $size_B$=2000 TX, and T_r=50 s. To provide a fairer assessment of the effects of different schemes, we compared two broker account selection methods, S500 and (Pre), representing poor and optimal broker selections, respectively. Meanwhile, we added Monoxide and CLPA on the right for a more comprehensive comparison.

From Fig. 8 (a), (c), and (e), we can observe that the increase in the number of poor brokers has an insignificant impact on the performance. In contrast, the increase in the number of excellent brokers has a relatively significant impact on the performance, especially for the BrokerChain scheme. The BrokerChain scheme does not periodically conduct account partitioning, so it relies on more excellent brokers to reduce the cross-shard TX rate. As the number of brokers increases to a certain extent, the main active accounts are all selected as brokers, and the performance improvement brought by further increasing the number of brokers will become smaller. In contrast, CLPA_Broker and ABChain are less affected by the increase in the number of brokers due to periodic account partitioning. Overall, our scheme outperforms the comparison schemes under various settings of the number of brokers.



(a) AvgTPS $Selc$=S500 (b) AvgTPS $Selc$=(Pre)

(c) TxClatency $Selc$=S500 (d) TxClatency $Selc$=(Pre)

(e) CSTxRatio $Selc$=S500 (f) CSTxRatio $Selc$=(Pre)

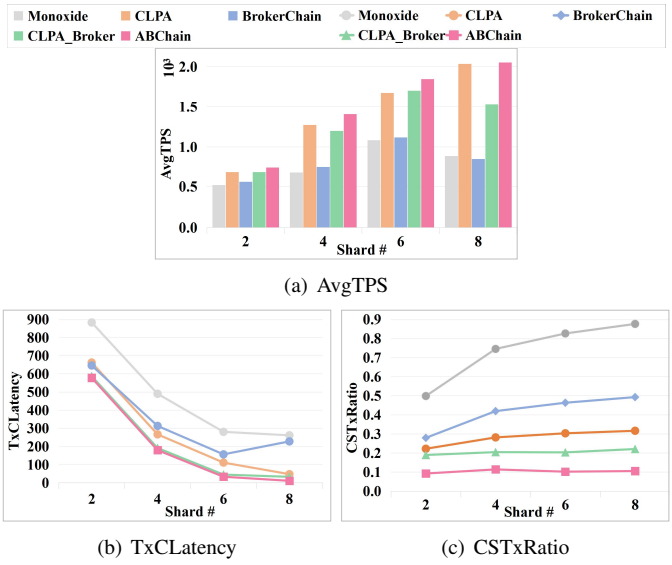
Fig. 8: Comparison of different number of brokers.

E. Shard Number

We evaluate how the number of shards (k) affects the system performance of both the baseline and our ABChain. We vary k within {2, 4, 6, 8}, and fix n_b=n_u=90, $Selc$=S500, $rate_A$=3000 TX/s, $size_B$=2000 TX, and T_r=50 s. The experimental results are shown in Fig. 9. It can be seen that as the value of k increases (from 2 to 6), most of the indicators of all the schemes have been improved. However, when k=8, Monoxide, BrokerChain and CLPA_Broker all experience a performance decline. We attribute this result to the following insights. Firstly, as k increases, the volume of cross-shard TXs also increases. Secondly, for BrokerChain and CLPA_Broker, poor brokers (S500) make the system unable to effectively optimize the processing of cross-shard TXs, resulting in the degradation of their performance. In contrast, under the same condition of poor brokers, ABChain significantly surpasses BrokerChain and CLPA_Broker in terms of the AvgTPS indicator and slightly surpasses CLPA, and also performs better than the comparison schemes in other indicators. Overall, our scheme demonstrates a better performance than the comparison schemes in the case of multiple shards, which indicates that our scheme ensures good shard scalability.

F. TX Arrival Rate

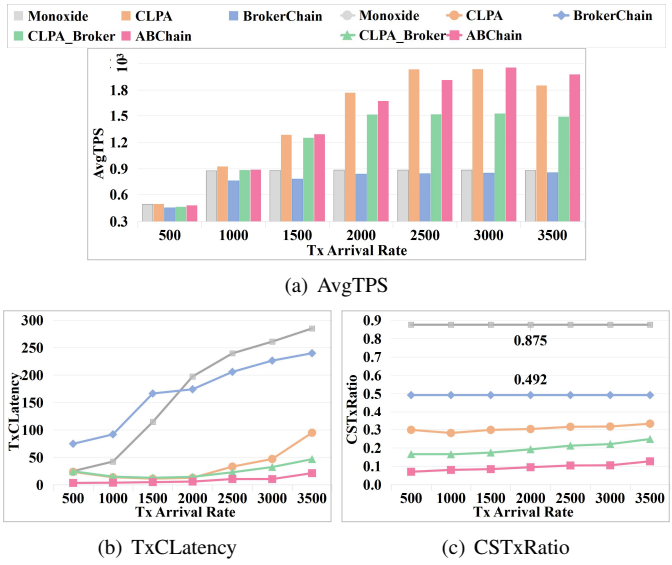
We investigated the impact of variations in TX arrival rates on the system performance of different schemes. $rate_A$ was



(a) AvgTPS

(b) TxCLatency (c) CSTxRatio

Fig. 9: Comparison of different number of shards.



(a) AvgTPS

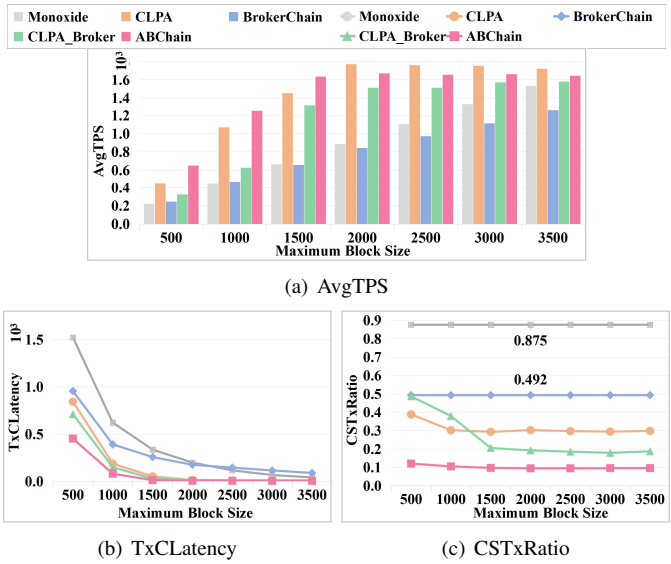
(b) TxCLatency (c) CSTxRatio

Fig. 10: Comparison of different TX arrival rate.

varied within {500, 1000, 1500, 2000, 2500, 3000, 3500} TX/s. Other parameters were set as follows: k=8, n_b=n_u=90, $Selc$=S500, $size_B$=2000 TX, and T_r=50 s. The results are shown in Fig. 10. When $rate_A$=500 TX/s, the performance of all schemes is poor, but the proposed scheme still outperforms the comparison schemes. As the $rate_A$ increases, all sharding schemes observe an improvement in the AvgTPS metric. However, beyond a certain threshold, this improvement becomes insignificant due to constraints from other factors, such as block size. Metrics like TxCLatency and CSTxRatio rise with the increase in $rate_A$, indicating a degradation in performance. Due to the static account allocation employed by Monoxide and BrokerChain, their CSTxRatio is a fixed value. Compared to the baseline, our ABChain exhibits a relatively more gradual and modest improvement in these metrics. Notably, when $rate_A$ is set to 3000 and 3500 TX/s, ABChain surpasses CLPA in the AvgTPS metric and demonstrates a significant advantage over CLPA_Broker, suggesting that the proposed method is capable of effectively handling large-scale TXs.

G. Maximum Block Sizes

We investigated the impact of variations in maximum block size on the system performance of different schemes. Larger block sizes allow for the inclusion of more TXs. We vary $size_B$ within {500, 1000, 1500, 2000, 2500, 3000, 3500} TXs, while other parameters were configured as follows: k=8, n_b=n_u=90, $Selc$=S500, $rate_A$=2000 TX/s, and T_r=50 s. The experimental results are presented in Fig. 11. When $size_B$=500 TX, $rate_A$=2000 TX/s is relatively a high value, resulting in a busy system state where a significant portion of time is consumed by block consensus, thereby leading to exceptionally high TxCLatency. It is evident that increasing the maximum block size does indeed improve the performance of the sharding schemes, as larger blocks allow for packing more TXs. However, after reaching a certain threshold, this improvement becomes less significant, because for sufficiently large blocks, the throughput is limited by factors such as TX arrival rate rather than block size. Overall, the proposed scheme outperforms the comparison schemes in terms of both AvgTPS, TxCLatency and CSTxRatio.



(a) AvgTPS

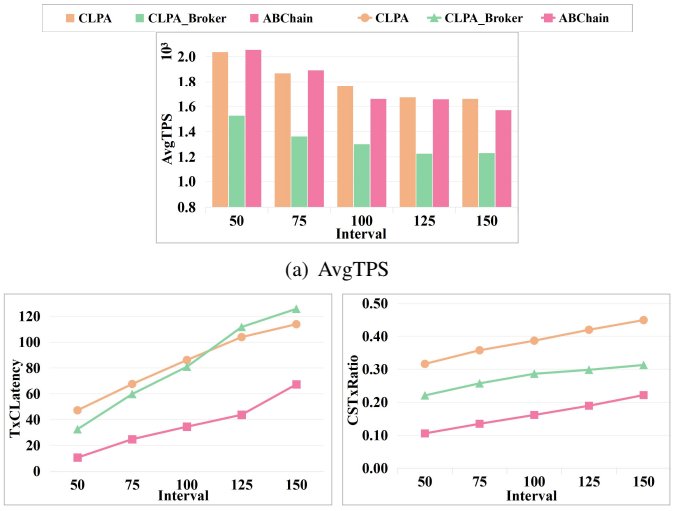
(b) TxCLatency (c) CSTxRatio

Fig. 11: Comparison of different maximum block size.

H. Account State Reconfiguration Interval

We compared the impact of different account state reconfiguration intervals on the system performance of different schemes, namely account partition interval in CLPA and CLPA_Broker, account partition and broker updates interval in ABChain. We vary T_r within {50, 75, 100, 125, 150}s, while fixing k=8, n_b=n_u=90, $Selc$=S500, $rate_A$=3000 TX/s, and $size_B$=2000 TX. As shown in Fig.12, with the increase of the T_r, the performance of all the schemes has deteriorated, indicating that account migration to reduce cross-shard TXs

is helpful for improving the performance of the blockchain system. Due to poor broker selection, CLPA_Broker has a very low AvgTPS, its TxCLatency is higher than that of CLPA at T_r = 125s and T_r = 150 s. Our scheme avoids the adverse effects brought by poor brokers through the adaptive update of brokers. CLPA has a similar AvgTPS as ABChain since there is no overhead for broker account reconfiguration, but a distinct divergence can be observed between CLPA and ABChain in terms of the TxCLatency and CSTxRatio. Overall, the performance of our scheme under different account state reconfiguration intervals is better than that of the comparison schemes.



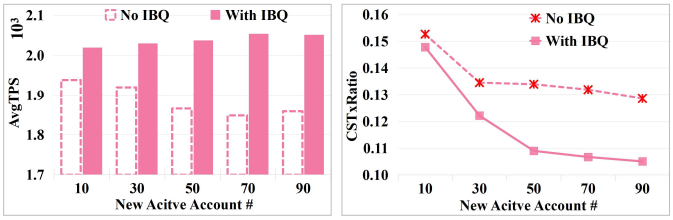
(a) AvgTPS

(b) TxCLatency (c) CSTxRatio

Fig. 12: Comparison of different state reconfiguration interval.

I. Number of Selected Active Accounts and The Presence of Idle Broker Queue

In this subsection, we investigate the impact of different n_u and tthe influence of configuring the IBQ or not on system performance. We set up two groups of controlled experiments, with IBQ (denoted as "With") and without IBQ (denoted as "No"). In each group of controlled experiments, we vary n_u within {10, 30, 50, 70, 90}, The other experimental parameters are set as follows: k=8, n_b=90, $Selc$=S500, $rate_A$=2000 TX/s. The experimental results are shown in Fig. 13. It can be observed that as n_u increases, the performance of the "No" scenario gradually degrades. This is because when brokers update, some BRelay-TXs split by expired broker inevitably remain unprocessed and become cross-shard TXs, necessitating further re-splitting. As n_u increases, the number of cross-shard BRelay-TXs also rises, hindering the efficient operation of the system. When an IBQ is introduced in the "With" scenario, an increase in n_u enhances system performance. This is because the IBQ functions as a buffer pool, where BRelay-TXs split by idle brokers do not need further re-split, and newly added active brokers can convert more cross-shard TXs into inner-shard TXs. In summary, for a scheme that supports the updating of mediator accounts, including an IBQ can effectively improve system performance.



(a) AvgTPS (b) CSTxRatio

Fig. 13: Comparison of different queue sizes for new brokers and investigating the impact of IBQ on performance.

VIII. Conclusion

This paper propose an adaptive broker-based sharding solution, termed ABChain. The core idea of ABChain is to select the most active accounts as brokers that are split and stored in multiple shards in each epoch, thereby minimizing cross-shard TXs. Additionally, through the brokers, cross-shard TXs can be effectively decomposed into several intra-shard TXs, significantly reducing communication and processing costs. To optimize the performance of ABChain and dynamically adjust the brokers, we further design a "dual broker queue" mechanism and devise a cross-shard consensus protocol. Compared to existing sharding schemes, ABChain does not rely on preset optimal brokers; instead, brokers adaptively update based on system operations. Experimental results fully demonstrate the effectiveness of selecting active accounts as brokers in ABChain, and the proposed method can significantly enhance the performance of sharded blockchain systems.

References

[1] Z. Xiong, Y. Zhang, D. Niyato, P. Wang, and Z. Han, "When mobile blockchain meets edge computing," *IEEE Commun. Mag.*, vol. 56, no. 8, pp. 33–39, 2018. [Online]. Available: https://doi.org/10.1109/MCOM.2018.1701095

[2] M. P. Caro, M. S. Ali, M. Vecchio, and R. Giaffreda, "Blockchain-based traceability in agri-food supply chain management: A practical implementation," in *2018 IoT Vertical and Topical Summit on Agriculture - Tuscany (IOT Tuscany)*, 2018, pp. 1–4.

[3] S. Liao, J. Wu, A. K. Bashir, W. Yang, J. Li, and U. Tariq, "Digital twin consensus for blockchain-enabled intelligent transportation systems in smart cities," *IEEE Trans. Intell. Transp. Syst.*, vol. 23, no. 11, pp. 22 619–22 629, 2022. [Online]. Available: https://doi.org/10.1109/TITS.2021.3134002

[4] J. Gao, K. O. Asamoah, Q. Xia, E. B. Sifah, O. I. Amankona, and H. Xia, "A blockchain peer-to-peer energy trading system for microgrids," *IEEE Transactions on Smart Grid*, vol. 14, no. 5, pp. 3944–3960, 2023.

[5] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *2016 ACM conference on computer and communications security*, 2016, pp. 17–30.

[6] J. Wang and H. Wang, "Monoxide: Scale out blockchains with asynchronous consensus zones," in *16th USENIX symposium on networked systems design and implementation (NSDI 19)*, 2019, pp. 95–112.

[7] L. N. Nguyen, T. D. Nguyen, T. N. Dinh, and M. T. Thai, "Optchain: optimal transactions placement for scalable blockchain sharding," in *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2019, pp. 525–535.

[8] N. Okanami, R. Nakamura, and T. Nishide, "Load balancing for sharded blockchains," in *Financial Cryptography and Data Security*, M. Bernhard, A. Bracciali, L. J. Camp, S. Matsuo, A. Maurushat, P. B. Rønne, and M. Sala, Eds. Cham: Springer International Publishing, 2020, pp. 512–524.

[9] S. Woo, J. Song, S. Kim, Y. Kim, and S. Park, "Garet: improving throughput using gas consumption-aware relocation in ethereum sharding environments," *Cluster Computing*, vol. 23, pp. 2235–2247, 2020.

[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 *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2018, pp. 583–598.

[11] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *2018 ACM conference on computer and communications security*, 2018, pp. 931–948.

[12] 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 *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022, pp. 1968–1977.

[13] J. Xu, Y. Ming, Z. Wu, C. Wang, and X. Jia, "X-shard: Optimistic cross-shard transaction processing for sharding-based blockchains," *IEEE Transactions on Parallel and Distributed Systems*, 2024.

[14] 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.

[15] C. Li, H. Huang, Y. Zhao, X. Peng, R. Yang, Z. Zheng, and S. Guo, "Achieving Scalability and Load Balance across Blockchain Shards for State Sharding," *IEEE International Symposium on Reliable Distributed Systems*, 2022.

[16] Y. Zhang, S. Pan, and J. Yu, "TxAllo: Dynamic Transaction Allocation in Sharded Blockchain Systems," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2023, pp. 721–733.

[17] 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.

[18] L. Jia, Y. Liu, K. Wang, and Y. Sun, "Estuary: A low cross-shard blockchain sharding protocol based on state splitting," *IEEE Transactions on Parallel and Distributed Systems*, 2024.

[19] S. Gregory, "Finding overlapping communities in networks by label propagation," *New journal of Physics*, vol. 12, no. 10, p. 103018, 2010.

[20] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," *consulted*, 2008.

[21] M. Li, W. Wang, and J. Zhang, "LB-Chain: Load-Balanced and Low-Latency Blockchain Sharding via Account Migration," *IEEE Transactions on Parallel and Distributed Systems*, pp. 1–14, 2023.

[22] H. Huang, Y. Lin, and Z. Zheng, "Account migration across blockchain shards using fine-tuned lock mechanism," in *IEEE INFOCOM 2024-IEEE Conference on Computer Communications*. IEEE, 2024, pp. 271–280.

[23] Z. Hong, S. Guo, P. Li, and W. Chen, "Pyramid: A layered sharding blockchain system," in *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, 2021, pp. 1–10.

[24] M. J. Amiri, Z. Lai, L. Patel, B. T. Loo, E. Lo, and W. Zhou, "Saguaro: An edge computing-enabled hierarchical permissioned blockchain," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2023, pp. 259–272.

[25] H. Tang, H. Zhang, Z. Zhang, Z. Zhang, C. Jin, and A. Zhou, "Towards high-performance transactions via hierarchical blockchain sharding," in *European Conference on Parallel Processing*. Springer, 2024, pp. 373–388.

[26] H. Huang, G. Ye, Q. Chen, Z. Yin, X. Luo, J. Lin, T. Li, Q. Yang, and Z. Zheng, "Blockemulator: An emulator enabling to test blockchain sharding protocols," *ArXiv*, vol. abs/2311.03612, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:265043188

[27] J. Xi, G. Xu, S. Zou, Y. Lu, G. Li, J. Xu, and R. Wang, "A blockchain dynamic sharding scheme based on hidden markov model in collaborative iot," *IEEE Internet of Things Journal*, vol. 10, no. 16, pp. 14 896–14 907, 2023.

[28] C. Lin, W. Liu, and D. Deng, "A genetic algorithm approach for detecting hierarchical and overlapping community structure in dynamic social networks," in *2013 IEEE Wireless Communications and Networking Conference (WCNC), Shanghai, Shanghai, China, April 7-10, 2013*. IEEE, 2013, pp. 4469–4474. [Online]. Available: https://doi.org/10.1109/WCNC.2013.6555298

[29] A. Liu, J. Chen, K. He, R. Du., J. Xu, C. Wu, Y. Feng, T. Li, and J. Ma, "Dynashard: Secure and adaptive blockchain sharding protocol with hybrid consensus and dynamic shard management," *IEEE Internet of Things Journal*, pp. 1–1, 2024.

[30] T. Cai, W. Chen, J. Zhang, and Z. Zheng, "Smartchain: A dynamic and self-adaptive sharding framework for iot blockchain," *IEEE Transactions on Services Computing*, vol. 17, no. 2, pp. 674–688, 2024.

[31] X. Ren, M. Xu, D. Niyato, J. Kang, C. Qiu, and X. Wang, "Paramart: Parallel resource allocation based on blockchain sharding for edge-cloud services," *IEEE Trans. Serv. Comput.*, vol. 17, no. 4, pp. 1655–1669, 2024. [Online]. Available: https://doi.org/10.1109/TSC.2024.3359608

[32] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, ser. OSDI '99. USA: USENIX Association, 1999, p. 173–186.

[33] J. R. Douceur, "The sybil attack," in *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*, ser. Lecture Notes in Computer Science, P. Druschel, M. F. Kaashoek, and A. I. T. Rowstron, Eds., vol. 2429. Springer, 2002, pp. 251–260. [Online]. Available: https://doi.org/10.1007/3-540-45748-8_24

[34] S. Sen and M. J. Freedman, "Commensal cuckoo: Secure group partitioning for large-scale services," *ACM SIGOPS Operating Systems Review*, vol. 46, no. 1, pp. 33–39, 2012.

[35] K. P. Puttaswamy, H. Zheng, and B. Y. Zhao, "Securing structured overlays against identity attacks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 20, no. 10, pp. 1487–1498, 2008.

[36] B. Awerbuch and C. Scheideler, "Towards a scalable and robust dht," in *Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures*, 2006, pp. 318–327.

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

[38] J. Ugander and L. Backstrom, "Balanced label propagation for partitioning massive graphs," in *Proceedings of the sixth ACM international conference on Web search and data mining*, 2013, pp. 507–516.

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

[40] R. Yan, W. Yuan, X. Su, and Z. Zhang, "Flpa: A fast label propagation algorithm for detecting overlapping community structure," *Expert Systems with Applications*, vol. 234, p. 120971, 2023.