A Secure and Efficient Blockchain Sharding Scheme via Hybrid Consensus and Dynamic Management

Meiqi Li, Xinyi Luo, *Graduate Student Member, IEEE*, Kaiping Xue, *Senior Member, IEEE*, Yingjie Xue, Wentuo Sun, *Graduate Student Member, IEEE*, and Jian Li, *Senior Member, IEEE*

***Abstract*— Sharding significantly enhances blockchain scalability by dividing the entire network into smaller shards that reach consensus and process transactions in parallel. Nevertheless, two new issues emerge with the adoption of sharding. One issue involves the shrinking size of consensus groups, which leads to vulnerability in consensus. Most existing works introduce periodic shuffle mechanisms to mitigate this problem. Nevertheless, these measures necessitate stronger security assumptions and can only offer a probabilistic assurance of consensus security. Another issue is the challenge in processing cross-shard transactions posed by the isolation of shards. Existing approaches utilize two-phase commit (2PC) or relay transaction mechanisms to handle cross-shard transactions. However, these approaches are vulnerable to double cross-shard attacks from malicious shards and are unable to achieve immediate atomicity. In this paper, to address the vulnerable consensus issue and achieve instant atomicity in cross-shard transactions, we design a hybrid consensus mechanism that embeds a lightweight global consensus into parallel intra-shard consensus processes. The global consensus allows all consensus nodes to jointly process cross-shard transactions, achieving cross-shard transaction instant atomicity. It also records shard snapshots to facilitate shard auditing to defend against malicious shards. Furthermore, we consider the performance of the proposed mechanism, and design a dynamic shard management mechanism. The dynamic shard management mechanism reduces transaction congestion and maintains an appropriate number of shards based on the system's state. We conduct analyses of potential attacks and prove that our approach ensures safety and liveness even in the presence of malicious shards. We also evaluate the performance of our system and compare it with both non-sharded and classic blockchain-sharding systems. The evaluation results demonstrate the efficacy of our approach in dealing with transaction congestion while astutely controlling the number of shards.**

***Index Terms*— Blockchain, sharding, distributed consensus.**

Manuscript received 18 December 2023; revised 21 April 2024; accepted 16 May 2024. Date of publication 27 May 2024; date of current version 31 May 2024. This work was supported in part by Anhui Province Key Technologies Research and Development Program under Grant 2022a05020050, in part by the National Natural Science Foundation of China under Grant 61972371 and Grant 62372425, in part by the Youth Innovation Promotion Association of the Chinese Academy of Sciences (CAS) under Grant Y202093, and in part by Guangzhou-The Hong Kong University of Science and Technology [HKUST (GZ)] Joint Funding Program under Grant 2024A03J0630. The associate editor coordinating the review of this manuscript and approving it for publication was Dr. Angelo Spognardi. *(Corresponding author: Kaiping Xue.)*

Meiqi Li, Xinyi Luo, Kaiping Xue, Wentuo Sun, and Jian Li are with the School of Cyber Science and Technology, University of Science and Technology of China, Hefei, Anhui 230027, China (e-mail: kpxue@ustc.edu.cn).

Yingjie Xue is with Guangzhou Municipal Key Laboratory of Financial Technology Cutting-Edge Research and the Thrust of Financial Technology, The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, Guangdong 511458, China.

Digital Object Identifier 10.1109/TIFS.2024.3406145

I. Introduction

BLOCKCHAIN technology has revolutionized various industries by providing decentralized ledgers with attributes such as data integrity, immutability, and reliable execution between untrusted parties [1]. The applications of blockchain span across diverse sectors, such as healthcare [2], [3], [4], [5], federated learning [6], [7], cloud computing [8], [9], mobile networks [10], [11], and the Internet of Things [12], [13], [14], [15]. However, traditional blockchain systems encounter a significant challenge in terms of scalability [16]. As the blockchain system scales up, the participation of each node in the consensus process imposes substantial communication [17], computation, and storage costs, hindering the application in large-scale scenarios [18]. To address the scalability limitation, sharding has emerged as a promising approach to improve consensus efficiency and overcome the scalability constraints of blockchain systems [19], [20], [21], [22], [23]. The basic idea of sharding is to divide the network into multiple disjoint shards. Each shard comprises a group of nodes forming a consensus group that processes transactions in parallel. The consensus achieved within a shard, known as *intra-shard consensus*, involves a smaller number of consensus nodes and a reduced volume of transactions to be processed [24]. Consequently, the communication, computation, and storage overhead are significantly reduced.

However, the sharding architecture introduces two main issues. On the one hand, the reduction in the number of nodes participating in a consensus process compared to non-sharding blockchain systems, referred to as ***consensus shrinkage***, leads to vulnerable consensus security [25]. Since the upper limit of the number of Byzantine nodes that a consensus can sustain decreases as the number of consensus nodes decreases [26], it is much easier for adversaries to control or attack a single shard than the whole system. On the other hand, the state of each shard is opaque to nodes outside the shard in blockchain sharding systems, since shards process disjoint sets of transactions and reach consensus independently, we refer to it as ***shards isolation***. Unfortunately, cross-shard transactions involve the state of multiple shards and require collaborative

verification and execution by all relevant shards [20]. The isolation of shards poses a challenge for a shard to ascertain the legitimacy of a related cross-shard transaction, as it lacks knowledge regarding the verification results of other relevant shards. Besides, due to the segregated nature of the execution process in relevant shards, achieving instant atomicity in cross-shard transaction processing becomes arduous. Furthermore, a malicious shard possesses the ability to generate illegal cross-shard transactions, such as those involving double-spending inputs, without detection from other shards, thereby destroying the security of the affected shards.

Facing the vulnerable intra-shard consensus brought by consensus shrinkage, existing approaches employ periodic shard reconfiguration mechanisms to enhance consensus safety and liveness. For instance, Elastico [19], Omniledger [27], and Rapidchain [20] periodically replace nodes of each shard to resist an adversary's concentrated corruption attacks on a certain shard. Facing the above cross-shard transaction processing issue brought by shard isolation, researchers propose mechanisms, such as two-phase commit (2PC) [22], [27], relay transactions [28], [29], etc. to handle cross-shard transactions. The underlying idea of these mechanisms is to enable the transaction receiver to execute the withdrawal operation within his/her shard after confirming that the transaction sender has successfully completed the deduction or withdrawal within his/her shard. Nevertheless, previous works still have certain limitations. First, even with periodical shards shuffle and node selection to prevent collusion, it is still possible for a shard to be controlled by adversaries. The shard corruption cannot be perceived by other shards due to a lack of auditing mechanisms. As a result, the corrupted shard can perform cross-shard double-spending attacks, destroying the system's security. In addition, as the withdrawal and deposit operations of cross-shard transactions are separately verified and executed across different shards, synchronizing their completion or abortion becomes a challenge, which poses a substantial obstacle to attaining cross-shard transactions' instant atomicity.

In this paper, we propose a secure and efficient dynamic blockchain sharding scheme to address the issues discussed earlier. Our scheme combines a carefully designed hybrid sharding consensus with a dynamic shards managing mechanism. Specifically, the hybrid consensus embeds a global consensus into multiple parallel intra-shard consensus processes. One challenge in designing the hybrid sharding consensus is the massive communication involved in a normal consensus process, such as PBFT [30], which limits the throughput due to the leader node bandwidth. Therefore, we only use the global block to record lightweight contents, including shard snapshots and cross-shard transactions. Besides, the communication cost is distributed to each shard. This innovative approach significantly enhances the efficiency of the standard global consensus process. To further improve the efficiency of transaction processing, and also avoid more serious consensus shrinkage caused by excessive sharding, we introduce a managing committee to dynamically adjust shard components based on the shard managing strategy. The dynamic shard management mechanism alleviates transaction congestion by splitting congested shards and improves shards utilization rate by merging underloaded shards. Besides, through adjusting user assignments, the dynamic shard management mechanism can also reduce the ratio of cross-shard transactions, thereby enhancing the system's performance. In summary, our main contribution can be summarized as follows.

- We propose a hybrid sharding consensus protocol that supports shard auditing and enables instant atomicity in processing cross-shard transactions. This protocol is capable of addressing vulnerable consensus issues and effectively defending against attacks from malicious shards.
- We design a dynamic shard management mechanism that enhances the efficient processing of potential transaction congestion while maintaining a balance between shard security and efficiency.
- The security analysis demonstrates that our system achieves consensus safety and liveness, as well as cross-shard transaction instant atomicity, even in the presence of corrupted shards. Additionally, the performance analysis supports the effectiveness of our approach in efficiently accelerating the processing of transaction congestion and achieving higher shard utilization rates.

Paper Organization: The rest of this paper is organized as follows. Section II provides preliminaries and related works for the paper. Section III describes the system model and the security assumptions. Section IV presents the details of our proposed blockchain sharding scheme. Section V gives several theorems on system security and discusses possible attacks. Section VI analyzes the system performance and evaluates the proposed system. Finally, we conclude our work in section VII.

II. Preliminaries And Related Work

A. Transaction Models of Blockchain Systems

There are mainly two transaction models widely used in the blockchain systems, namely the *unspent transaction output* (UTXO) model [31] and the *account/balance* model [32]. The UTXO model was introduced in Bitcoin and used in many blockchain systems. In this model, each transaction is composed of inputs and outputs. An input of a transaction references a previous transaction's output and the UTXO is "spent" by the transaction. Validators keep a database to record valid UTXOs and use it to verify whether a block can be accepted. The account/balance model is employed by Ethereum. In the account/balance model, each user has a pair of account and balance. Users can use their balance to generate a transaction that contains a sender account, a receiver account, and a value.

B. Representative Blockchain Sharding Schemes

In terms of blockchain's poor scalability issue, researchers proposed a great number of blockchain sharding schemes. Elastico [19] is the first sharding-based blockchain system, which divides nodes into multiple smaller consensus groups to validate transactions in parallel. However, each node needs to spread, receive, and store all the blocks of the system and therefore there still exists heavy communication

and storage burden. Omniledger [27] achieves complete sharding, which includes communication, computation, and storage sharding. It adopts a client-driven two-phase commit mechanism to handle cross-shard transactions. Zamani et al. proposed RapidChain [20] to reduce the overhead during shards reconfiguration and further speed up the consensus process. Wang et al. proposed Monoxide, which allows a miner to create multiple blocks in different shards by solving one proof-of-work puzzle, thereby amplifying the effective mining capacity of each shard. OptChain [33] minimizes the number of cross-shard transactions by using an optimal transactions placement strategy. Recently, Zheng et al. [29] proposed a sharding solution for consortium blockchain called Meepo, which enhances cross-shard efficiency via the cross-epoch and cross-call. Hong et al. proposed Pyramid [34], a layered sharding solution that uses bridging shards to validate and process cross-shard transactions internally. Huang et al. presented BrokerChain [35] that achieves workload balance among mining shards by adaptively partitioning account states using a partition shard.

C. Sharding Security and Cross-Shard Transaction Processing

Blockchain sharding brings two main challenges: consensus group size reduction and cross-shard transaction processing. Some sharding protocols [19], [20], [27] use a reconfiguration phase to resist adversary adaptive attacks by shuffling all shards or randomly adjusting some nodes. However, the sharding security is guaranteed by probability and it is still possible for a certain committee to be controlled by an adversary [25]. For cross-shard transaction processing, Omniledger [27] uses a two-phase commit mechanism, that allows users to complete cross-shard transactions by performing lock/unlock operations on each relevant shard. Liu et al. [36] utilized multiple parallel cross-shard Byzantine fault tolerance protocols to process cross-shard transactions more efficiently. RapidChain [20] splits a cross-shard transaction into multiple sub-transactions that can be processed in a single shard. Monoxide [28] adopts a relay transaction based solution, where the deduction operations and deposit operations are separatly executed in each related shard. Pyramid [34] introduces bridging shards that store multiple shards' states so that can process cross-shard transactions internally. However, those cross-shard transaction processing methods are all based on the premise that each shard remains secure. One shard crash can cause errors in validating all related cross-shard transactions, and therefore affect the correctness of other shards. Besides, the processing of each cross-shard transaction requires multiple rounds of intra-shard consensus, often resulting in system performance bottlenecks.

III. System Model, Threat Model, and Design Goals

A. System Model

This work adopts the UTXO model. All the nodes are connected by a partially synchronous peer-to-peer network in which messages can be sent to each other with optimistic, exponentially increasing time-outs. To prevent Sybil attacks, all the nodes should establish their identities (i.e., public/private keys) by solving a computationally hard puzzle [37] before joining the network. We assume all messages sent in the network are authenticated with the sender's private key.

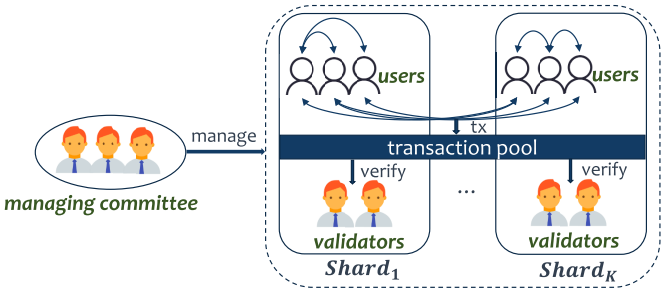


Fig. 1. System model.

There are two kinds of entities in the system, namely **users** and **validators**. As shown in Fig. 1, users and validators are divided into multiple shards. The number of shards in the system is represented by the variable K. Users trade with each other and generate transactions both intra-shard and cross-shard. Validators verify the transactions and keep a ledger to record them. A group of validators is randomly selected to form a **managing committee** ($\mathcal{MC}$), which manages the shards through a publicly determined shard management strategy.

B. Threat Model

Most sharding systems [19], [20], [27], [34] limit the malicious nodes can only be changed between epochs. We weaken that assumption and adopt an adaptive corruption model, where the adversaries know the allocation of nodes to the shards, and are able to adaptively select target shards to attack. We assume the number of malicious nodes grows linearly, and the fraction of malicious validators, denoted by f, remains below $1/3$ throughout the entire process. However, in certain shards, the proportion of malicious validators may exceed 1/2 due to concentrated attacks from adversaries. Here gives the threat model in our work as follows:

- *Users.* Malicious users may generate invalid transactions by including previously used inputs and launch double-spending attacks. Our system does not limit the proportion of malicious users.
- *Validators.* There are honest and malicious validators in our system. The honest validators perform all operations correctly according to the protocol, while the malicious validators try to interfere with the execution of the protocol by colluding with each other.

C. Design Goals

Main objective of our work aim to enhance the security and performance of blockchain sharding systems. In terms of security, except for consensus safety and liveness under normal circumstances, we have two further security goals:

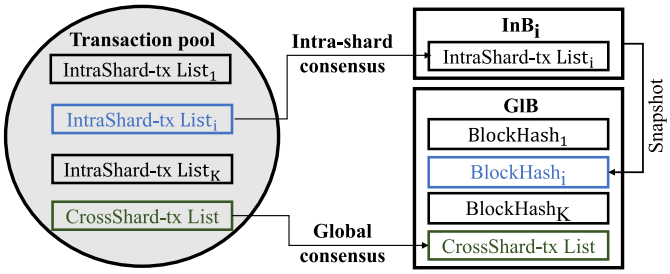


Fig. 2. Transactions in the transaction pool, the content of intra-shard blocks, and the global block.

malicious shard resistance and *instant atomicity for cross-shard transactions processing*, which are defined as follows.

- *Malicious shard resistance* refers to detect and process double-spending cross-shard attacks in the presence of corrupted shards.
- *Instant atomicity for cross-shard transactions processing* refers to the withdraw and deposit operations for a cross-shard transaction either all occur or none of them occur at the same time.

In terms of performance, our aim is to dynamically adapt to the varying transaction rates of the system, fulfilling the throughput requirements.

IV. Protocol Design

A. Overview

The proposed sharding protocol runs in fixed time periods called *epochs* as other blockchain sharding schemes [20], [34], [35]. At the beginning of each epoch, validators and new users establish their identities and join the network by solving a hash-based puzzle. Validators generate unpredictable randomness via public-verifiable and unbiased verifiable random functions (VRF) [38]. A group of validators is randomly selected based on epoch randomness to form a managing committee ($\mathcal{MC}$), which dynamically adjusts users and validators assignment to shards based on the system status.

There are two phases that cyclically alternate in the system: namely the *transaction processing phase* and the *shards managing phase*. The transaction processing phase involves the processing of user transactions using the hybrid sharding consensus mechanism. The shard managing phase includes auditing the shards and dynamically adjusting user and validator assignments. In the following, we provide a detailed description of the two phases.

1) Transaction Processing Phase: In the transaction processing phase, users generate transactions and put them into the transaction pool. As shown in Fig. 2, the transactions in the transaction pool are divided into intra-shard transactions and cross-shard transactions. Intra-shard transactions are verified by the corresponding shard validators, processed during the intra-shard consensus process, and recorded in intra-shard blocks. In contrast, cross-shard transactions are processed by all validators in the system and recorded in the global block along with all the snapshots $BlockHash$ of intra-shard blocks, which enables shards auditing. During each consensus round, validators verify intra-shard transactions and their related cross-shard transactions and broadcast a global message that contains the $BlockHash$ and the related cross-shard transaction verification result. Then, they use fragmented global messages from each shard to calculate the legal cross-shard transaction list and perform a hybrid sharding consensus. For brevity, the detailed processing method for cross-shard transactions and the hybrid sharding consensus process are described in section IV-B and section IV-C respectively.

2) Shard Managing Phase: The shards managing phase comprises two key functions: adjustment and auditing. The shards adjustment is performed periodically. Specifically, the managing committee ($\mathcal{MC}$), which is randomly selected at the beginning of each epoch, monitors the status of each shard and implements a dynamic shard management mechanism to make necessary adjustments to validators and users. To address transaction congestion issues, congested shards are split into smaller sizes to accelerate transaction processing. Besides, a graph partition algorithm is employed to reduce the proportion of cross-shard transactions by transferring users among shards strategically. Conversely, underloaded shards, which possess transaction processing capabilities that surpass the actual transaction volume, are merged together to improve the shards' utilization ratio. In the meantime, when the system remains idle, $\mathcal{MC}$ can propose a shard auditing, which aims to detect potential malicious shard behaviors, such as cross-shard double-spending attacks, and punish the malicious validators. During the shard auditing phase, each shard's intra-shard transactions are verified by other validators, and the snapshots recorded in the global block is used to prevent malicious shards from tampering with history transactions.

The subsequent subsections introduce the essential components of the proposed system, including *cross-shard transactions processing*, *hybrid sharding consensus*, *shards auditing*, and *dynamic shard management mechanism*.

B. Cross-Shard Transactions Processing

In the proposed sharding protocol, a cross-shard transaction $crTx$ involves inputs consuming UTXOs from designated *input shards* and outputs generating new UTXOs in designated *output shards* and is considered valid only if all inputs are unspent and all the outputs are valid. The input shards and output shards are both referred to as *related shards*. To achieve instant atomicity across all shards, cross-shard transactions are processed globally and recorded in global blocks. Consequently, the verification process of cross-shard transactions consists of two stages. The first stage is referred to as *partial verification*, involving each related shard checking for *local legality*. The second stage is the *final verification*, where all validators summarize the partial verification results to determine the overall legitimacy of cross-shard transactions.

- *Partial verification.* Each related shard verifies the legality of the corresponding inputs/outputs within its domain, defined as the *local legality* of $crTx$. Specifically, the input shard examines whether the corresponding UTXOs are unspent outputs of valid transactions within its domain, and the output shards verify the integrity and validity of the output UTXOs, i.e. whether the sum of

inputs is not less the sum of outputs. The related shard leaders broadcast the partial verification results and the signatures of intra-shard validators to all validators.

- *Final verification.* Upon a validator obtaining the partial verification outcomes from all related shards, he/she can verify the overall legitimacy of $crTx$ by checking if it passes all the partial verifications. Conversely, if any partial verification result is deemed invalid, $crTx$ will be classified as an invalid transaction.

Cross-shard transactions that pass the final verification can be executed and packed into the global block through the hybrid sharding consensus mechanism. More detailed information about this mechanism can be found in Section IV-C. Conversely, invalid cross-shard transactions are discarded.

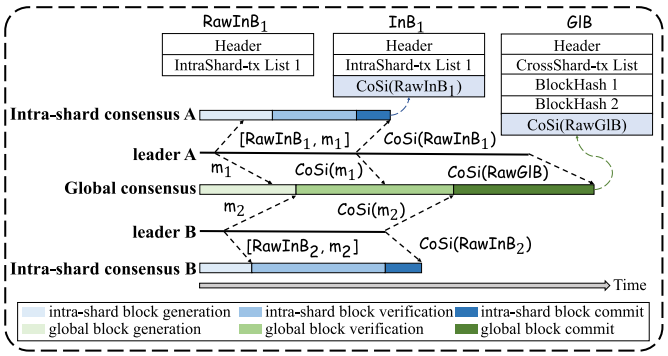


Fig. 3. The process of the hybrid sharding consensus for a two-shards blockchain system.

C. Hybrid Sharding Consensus

The system running time is divided into multiple fixed lengths, called *epochs*. Each epoch consists of a number of *slots*, and each slot corresponds to a consensus round. In a slot, each shard randomly selects its leader based on epoch randomness and runs a hybrid sharding consensus.

1) Hybrid Sharding Consensus Process: Generally speaking, a lightweight global consensus is embedded into general intra-shard consensus processes that operate in parallel. The intra-shard consensus is used to process intra-shard transactions. Also, similar to most leader-based consensus protocols, such as PBFT [30], PoS [39]. Intra-shard consensus can be abstracted into three stages:

1) *Block generation.* The leader packs valid transactions into a raw block and broadcasts it to other validators.
2) *Block verification.* Validators verify the proposed raw block and send their signatures to the leader if the raw block is validated.
3) *Block commit.* After receiving enough signatures, the leader can generate the final block with confirmation proof and broadcast it to all validators. In our work, a collective signing protocol [40], [41] that can generate a multi-signature co-signed by a decentralized group of nodes is used for scalability concerns.

The global consensus is used to handle cross-shard transactions and record shard snapshots. To realize a lightweight and efficient global consensus, the communication cost is distributed to every shard. Specifically, during the global block generation stage, global messages $\{m_1, \ldots, m_K\}$, where K is the number of shards in the system, are sent from each shard leader. Each global message m_i contains shard S_i's related cross-shard transaction list $crTxL_i$ and the shard snapshot $BlockHash_i$, which is the hash of the intra-shard block header. Given $\{m_1, \ldots, m_K\}$, the valid cross-shard transaction list can be calculated by using the cross-shard transaction verification method mentioned in subsection IV-B. Validators calculate the global block and subsequently wait for the majority of nodes within a shard to confirm its m_i. Once all shards send confirmations for their global messages (or timeout), the global block verification stage is completed. Besides, there is an error-handling mechanism to deal with abnormal cases and correct the global block. After the majority of nodes sign the global block, a leader can aggregate the signatures and commit the global block.

The hybrid consensus protocol is run by all the shards in a consensus round. As shown in Fig. 3, a simple blockchain system contains two shards, named shard A and shard B, is used as an example to describe the specific process of the hybrid sharding consensus. Both shard A and shard B execute the same procedures. Therefore, we use shard A and its intra-shard consensus leader $\overline{L}_A$ to describe the consensus protocol.

1) *Intra-shard block generation.* $\overline{L}_A$ verifies intra-shard transactions of shard A in the transaction pool, packs valid ones into $IntraShard-txList_1$, and generates a raw intra-shard block

$$RawInB_1 :=< header, IntraShard-txList_1 >$$

, and computes $BlockHash_i = Hash(header)$. The block header contains the hash of the parent block and the Merkle tree root of the body. Then $\overline{L}_A$ sends $RawInB_1$ to shard A's other validators.
2) *Global block generation.* Upon verifying related cross-shard transactions in the transaction pool, $\overline{L}_A$ packages ones satisfying *local legality*, which is defined in Section IV-B, into the cross-shard transaction list $crTxL_1$, and generates the global message

$$m_1 = [crTxL_1||BlockHash_1].$$

Then $\overline{L}_A$ broadcasts m_1.
Validators collect $\{m_1, m_2\}$ and use them to calculate the raw global block, which contains snapshots of the two shards and a valid cross-shard transaction list.
3) *Intra-shard block verification.* Validators in shard A verify $RawInB_1$ and reply $\overline{L}_A$ with their signatures $Sig(RawInB_1)$ if $RawInB_1$ is valid.
4) *Global block verification.* Validators in shard A verify m_1 and reply $\overline{L}_A$ with their signatures $Sig(RawInB_1)$ and $Sig(m_1)$ if these components pass the validation process. Upon receiving more than $1/2$ of $Sig(m_1)$ sent from shard A, $\overline{L}_A$ generates a collective signature $CoSi(m_1)$ and broadcasts it. Validators collect $CoSi(m_1), CoSi(m_2)$ to confirm the raw global block $RawGlB$ before the timeout. Then use the error handling mechanism (see the next paragraph) to correct

$RawGlB$, and send all shards' leaders their signatures $Sig(RawGlB)$.

5) *Intra-shard block commit.* $\overline{L_A}$ generates a collective signature after collecting more than $1/2$ $Sig(RawInB_1)$ and sends it to other validators in shard A so that they can add it to $RawInB_1$ and get the final valid intra-shard block InB_1.
6) *Global block commit.* A shard leader $\overline{L_G}$ generates a collective signature $CoSi(RawGlB)$ on the global block after receiving more than half of the signatures and broadcasts $CoSi(RawGlB)$. Validators add it to the raw global block and get the final valid global block GlB.

2) Error Handling Mechanism: In the above design, a normal global consensus requires each shard leader to correctly broadcast a global message and its shard confirmation in time. Suppose there is a shard leader $\overline{L_i}$ that did not send m_i or its confirmation of m_i before the timeout. Once the confirmation timeout has transpired, validators calculate the global block utilizing other global messages that are already confirmed. The corresponding shard snapshot of S_i in the global block is set as null and S_i's related cross-shard transactions cannot be processed in the consensus round.

D. Shards Auditing

The shards auditing mechanism is designed to prevent malicious shards from performing cross-shard double-spending attacks. It is started when the system is idle and requires each validator to submit a deposit before joining the system. In general, during the shards auditing phase, all validators synchronize the intra-shard blocks to detect malicious shard behaviors. The detecting results and the punishment of malicious shards' validators are processed through a global consensus.

In detail, after synchronizing all the intra-shard blocks that were generated during the previous transaction processing phase within the system, validators verify the global blocks and the intra-shard blocks of each shard. Firstly, they confirm whether the intra-shard block matches the corresponding shard snapshot recorded in the global block, preventing malicious shards from tampering with the intra-shard block. Then, validators examine whether there are cross-shard double-spending attacks by checking whether there are intra-shard transactions using the same input as cross-shard transactions on the global block. Note that transaction recipients can identify intra-shard double-spend attacks by checking intra-shard blocks. As a result, these types of attacks fall outside the purview of shards auditing. Upon detecting cross-shard double-spending attacks, the malicious validators who signed for the double-spending transactions recorded on the intra-shard block will not be allowed to join the system. Besides, part of their deposit will be transferred to the victim user(s) based on the auditing regulations, which can be flexibly formulated according to the system's actual needs. The remaining validators and users will be randomly assigned to other shards by the managing committee. Finally, validators in the system reach a global consensus on the shards auditing results, including malicious shards detecting results and the punishment of malicious validators.

1) Cost Analysis of Shards Auditing: During the shards auditing phase, all shards are merged into one single chain, and all intra-shard transactions are synchronized and confirmed between shards. The cost of shards auditing is similar to using non-sharding blockchain to process these transactions. Therefore, our solution is most applicable for systems with peak and low trading periods and an overall sufficient average computing power. The system sharding occurs during peak trading periods to speed up transaction processing, and shards consolidation and auditing are performed during low trading periods.

E. Dynamic Shard Management Mechanism

1) Motivation: The number of shards in the system and the method of assigning validators and users can both have an impact on the system's performance and security. For example, the system's transaction processing capacity increases as the number of shards grows. However, it also leads to a greater security threat to each single shard. Therefore, it is essential to dynamically achieve a balance between performance cost and security cost based on the system's current needs. We model the system cost, which is divided into performance cost and security cost. The performance cost C_{latency} is measured by the average transaction confirmation latency, and the security cost C_{security} is quantified by the expected proportion of validators who are in malicious shards in the system. Given that the total number of validators in the system is N, the proportion of corrupt validators is f. Validators are assigned to K shards, with the number of validators in each shard denoted as $\{N_1, \ldots, N_K\}$. The average transaction arrival rate in the system is set as v_{arrival}, and the average transaction processing speed is v_{process}. Then the performance cost can be calculated as $C_{\text{latency}} = max(v_{\text{arrival}}/v_{\text{process}} - 1, 0)$. The security cost is calculated as:

$$C_{\text{security}} = \sum_{i=1}^{K} \sum_{j=\lfloor N_i/2 \rfloor}^{N_i} \frac{N_i \binom{f \cdot N}{j} \cdot \binom{(1-f) \cdot N}{N_i - j}}{K \cdot \binom{N}{N_i}}.$$

The overall cost of the system $C_{\text{overall}} = w_1 \cdot C_{\text{latency}} + w_2 \cdot C_{\text{security}}$, where w_1 and w_2 are weight factors assigned to balance the influence of performance and security costs. Our goal is to reduce the overall system cost and achieve a balance between performance and security.

2) Shard Management Strategy: To enhance transaction processing speed and alleviate congestion during transaction bursts, an intuitive idea is to reduce the number of cross-shard transactions that are processed by global consensus and further accelerate intra-shard consensus through shard splitting. Besides, it is also important to merge shards when the system's transaction arrival rate is low, therefore increasing consensus capacity and enhancing security. Building upon the prior analysis, a dynamic shard management mechanism is designed to adjust the shard component according to the system status, avoiding transaction congestion and also more serious consensus shrinkage brought by excessive

sharding. In the dynamic shard management mechanism, shard adjustment is divided into the following three types:

- *Shards split.* $S_i \xrightarrow{Split} \{S_{i_1}, \ldots, S_{i_m}\}$. Splitting S_i means randomly reallocating S_i users and validators into multiple smaller shards $\{S_{i_1}, \ldots, S_{i_m}\}$. Note that each shard needs to have enough validators after the division.
- *Shards merge.* $\{S_i, \ldots, S_j\} \xrightarrow{Merge} S_i'$. Merging the shards means combining the shards' validators to collectively process transactions for the shards' users.
- *Reassign users.* Users are reassigned to shards by utilizing graph-partitioning tools to reduce cross-shard transactions.

The details of the dynamic shard management strategy are introduced in the following. Algorithm 1 illustrates the approach, with the input of the current shards' composition, including validator and user lists for each shard, the unprocessed transactions within the transaction pool, and system parameters, it outputs an adjusted shards composition. First, it checks whether each shard is a *congested shard*, which is defined as a shard whose number of unprocessed transactions, including intra-shard transactions and related cross-shard transactions, is higher than the threshold number n_c. Subsequently, it splits each congested shard into several smaller shards and keeps the shards with more than s validators. Then it checks for *underloaded shard*, whose number of unprocessed transactions is less than $r \cdot n_c$, $r \in (0, 0.5)$. For all underloaded shards, it sorts them according to the transaction queue size, merges the first and last shards, and loops. Finally, it uses the unprocessed transactions to build the transaction graph, where each vertex represents a user's address, and the edge weight is defined as the number of transactions that involve the corresponding pair of addresses as inputs and outputs. It is supposed to use graph-partitioning tools, such as Metis [42], to partition the transaction graph into non-overlapping K parts with a reduced number of cross-shard transactions. The division of vertexes in the transaction graph into subgraphs corresponds to the users divided into corresponding shards.

3) Shards Adjustment: In the shard managing phase, $\mathcal{MC}$ leader broadcasts the current state of the system and the shard adjustment result, which is calculated based on the dynamic shard management algorithm, to other $\mathcal{MC}$ members. Upon verifying the correctness of the shard management result, other members in $\mathcal{MC}$ sign it and return the signatures to the leader. Once the leader collects signatures from more than half of the $\mathcal{MC}$ members, a valid $\mathcal{MC}$ intra-block is generated, which includes the shard adjustment result and a collective signature from the majority of $\mathcal{MC}$ members. Subsequently, the leader broadcasts the block to all nodes in the system, and the validators in the adjusted shard perform the corresponding operations based on the type of shard adjustment after receiving the block.

- *Shards split.* $S_i \xrightarrow{Split} \{S_{i_1}, \ldots, S_{i_m}\}$. For S_i's validators who are reassigned to S_{i_x}, $x \in \{1, \ldots, m\}$, they update their UTXO lists and only keep UTXOs belonging to users of S_{i_x}.

Algorithm 1 Dynamic Shard Management

Input: Current shards composition;
Transaction pool state;
System parameters n_c,r,s ;
Output: Adjusted shards composition;

```
US_SET ← ∅;
for each shard S_i do
    if S_i's transactions queue size is above n_c then
        Split S_i into min(⌈(n_tx/n_c)⌉, ⌊n_v/s⌋) shards;
    end
    if S_i's transactions queue size is below r · n_c then
        Add S_i into US_SET;
    end
end
Sort the shards in US_SET according to the average
  transaction queue size;
while US_SET is not empty do
    Merge the first and the last shard and remove them
      from US_SET;
end
Use the graph partition tool to reassign users to shards.
```

- *Shards merge.* $\{S_i, \ldots, S_j\} \xrightarrow{Merge} S_i'$. Validators of shard $\{S_i, \ldots, S_j\}$ synchronize their UTXO datasets with each other.
- *Reassign users.* For each reassigned user, if user U_i is reassigned from shard S_i to shard S_j, the process requires S_i's validators to send all UTXOs owned by the user to S_j's validators. Subsequently, the validators of S_j download the UTXOs and update their local UTXO dataset.

V. Security Analysis

In this section, we first prove the global consensus always achieves *safety* and *liveness*. Safety indicates honest validators agree on the same valid block in each consensus round and liveness indicates every block will be finally committed or aborted. We then analyze possible attacks and how our system defends against them.

A. Analysis of Hybrid Sharding Consensus

Theorem 1: The global consensus achieves safety if the fraction of malicious validators in the system is less than $1/2$.

Proof: To generate a valid global block with a collective signature, the global consensus leader must collect signatures from more than half of the validators. All honest validators can use the confirmed global messages to calculate the contents of the global block and send their signatures to shard leaders. Given that the fraction of malicious validators within the system is less than $1/2$, the global consensus leader cannot receive more than $1/2$ of validators' signatures on a global block with invalid contents. Therefore, the global consensus achieves safety. □

Theorem 2: The global consensus achieves liveness if the fraction of malicious validators in the system is less than $1/2$.

Proof: Note that a global block, which contains the cross-shard transaction list and $BlockHash$ of each shard, is calculated based on confirmed global messages that are agreed upon by more than half of the shard validators. According to our hybrid sharding consensus protocol in Section IV, all honest validators can use the global messages and the error handling mechanism to calculate a consistent global block and send their signature on the global block to shard leaders. Our partially synchronous peer-to-peer network assumption ensures the signatures can be received by leaders within an optimistic bounded time. Since the shard leaders are chosen randomly and the randomness is unbiased, there will be $K/2$ honest leaders, where K is the number of shards, every round in expectation. The honest leaders will generate the valid global block with collected signatures and broadcast it to each node. Thus, all honest nodes will receive the finalized global block at the end of the global consensus process. □

Theorem 3: The intra-shard consensus achieves safety and liveness if the fraction of malicious validators within the shard is less than $1/2$.

Proof: To generate a valid intra-shard block InB_i, a shard consensus leader $\overline{L}_i$ needs to acquire signatures from a majority of validators within the shard. Given that the proportion of malicious validators in the shard is less than $1/2$, $\overline{L}_i$ cannot collect enough signatures on invalid InB_i since honest validators only sign on blocks containing valid transactions. Therefore, the intra-shard consensus achieves safety. Similarly, drawing parallels to our proof on the liveness of the global consensus, the completion of intra-shard consensus is reliant upon the presence of an honest intra-shard consensus leader. As the shard leaders are randomly selected within the shard, it can be deduced that each shard will have an honest intra-shard leader approximately every 1.5 rounds on average. Consequently, all honest nodes are able to finalize the block within a bounded timeframe. □

Therefore, the hybrid sharding consensus implemented within our system attains both safety and liveness, given that the fraction of malicious validators is less than $1/2$ in the entire system, as well as within each shard. It is noteworthy that even if certain shards surpass the security threshold in terms of the proportion of malicious validators, the intra-shard consensuses within other shards remain unaffected. Besides, the global consensus can also be effectively achieved through the utilization of the error handling mechanism elucidated in section IV, as long as the system has less than $1/2$ malicious validators.

B. Instant Atomicity for Cross-Shard Transactions Processing

Monoxide [28] handles cross-shard transactions by using relay transactions to allow the withdraw operation to be executed first and the corresponding deposit operation to be settled later. It achieves the *eventual atomicity*, which means the deposit operation must be executed eventually once the withdraw operation is confirmed. However, the confirmation time is not limited. BrokerChain [35] introduces brokers to process cross-shard transactions and also allows the withdraw operation and its corresponding deposit operation to be executed separately. BrokerChain ensures the time between the two operations is limited within a predefined time, so that reaches *duration-limited eventual atomicity*.

In our system, by contrast, a cross-shard transaction can be executed only if it is verified by all related shards and is recorded on a global block. Thus, the withdraw and deposit operations for a cross-shard transaction either all occur or none of them occur at the same time. Therefore, our cross-shard transaction handling mechanism achieves instant atomicity.

C. Tackling Potential Attacks

In the following, we list some potential attacks and analyze how our system handles them.

Attack 1: A malicious shard leader broadcasts an incorrect global message m_i during the global consensus process. Suppose the shard has no more than $1/2$ malicious validators.

Honest validators in the shard will verify m_i and not sign on it after noticing that m_i is incorrect, i.e. containing an incorrect shard snapshot or double-spending cross-shard transactions. As a result, the leader cannot collect enough signatures to generate a valid collective signature. Then other validators will discard the global message and recalculate the global block after timeout according to the error handling mechanism. Thus, the global consensus will not be affected, only the shard's intra-shard consensus will fail.

Attack 2: Malicious shards launch double-spending cross-shard attacks. (Generate cross-shard transactions with an input that has spent intra-shard.)

We denote the cross-shard transaction and the intra-shard transaction that have a shared input as cTx and iTx separately. Suppose iTx is recorded on an intra-shard block InB_i. The malicious shard deceive other shards by broadcasting a global message $m_i = [crTxL_i||BlockHash_i]$ during the consensus round processing cTx.

- Case 1: $BlockHash_i = Hash(InB_i)$.
  During the audit phase, InB_i needs to be broadcast to other shards. Otherwise, other validators can compute the hash of the block and detect the malicious behavior by comparing $BlockHash$. Then other validators can be aware that cTx and iTx compose a double spending attack. The deposit of validators from the malicious shard will be transferred to the victim user(s) as per the auditing regulations.
- Case 2: $BlockHash_i \neq Hash(InB_i)$.
  The malicious shard may try to fake an intra-shard block InB_i' without iTx and broadcast $Hash(InB_i')$ to cheat other shards' validators to avoid being punished during the shards auditing phase. The recipient of iTx (or any node within the shard) can discover the malicious behavior by comparing the $BlockHash$ recorded in the global block and $Hash(InB_i)$. Then he/she can be aware that the shard is malicious and avoid real-world financial losses due to the invalid transaction.

In conclusion, the correctness of global messages and the processing of intra-shard transactions are guaranteed by the safety of the corresponding shard. Moreover, the global consensus consistently maintains safety as long as the overall

system has a limited number of corrupted nodes that fall below the safety threshold. Consequently, even with the appearance of corrupted shards, other shards possess the capability to resist their malicious attacks and safeguard users' funds by leveraging the recorded $BlockHash$ on the global block during the shards auditing phase.

D. Analysis of the Managing Committee

Since the managing committee ($\mathcal{MC}$) is randomly selected from all validators at the beginning of each epoch, there is a certain probability that more than half of the validators in the $\mathcal{MC}$ are malicious. In this case, the security of the $\mathcal{MC}$'s consensus process is compromised. Suppose at the beginning of epoch e, there are a total of N validators in the system, with a corruption ratio of f, and the number of validators in the $\mathcal{MC}$ is N_{mc}. The probability that the majority of validators in the $\mathcal{MC}$ are malicious can be calculated as:

$$P_{Malicious} = \sum_{i=\lfloor N_{mc}/2 \rfloor}^{N_{mc}} \frac{\binom{f \cdot N}{i} \cdot \binom{(1-f) \cdot N}{N_{mc} - i}}{\binom{N}{N_{mc}}}.$$

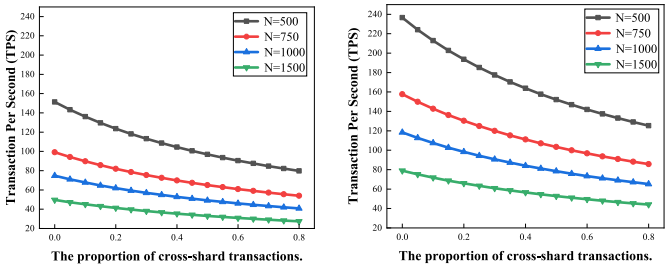
The possible behavior and impact on the system when $\mathcal{MC}$ consists of more than half malicious validators are analyzed below.

1) Lazy $\mathcal{MC}$: A lazy $\mathcal{MC}$ fails to make necessary shard adjustments during the shard managing phase. This lazy behavior results in the inability to optimize system costs in epoch e, leading to higher average transaction confirmation latency or lower shards security.

2) Malicious $\mathcal{MC}$: Malicious validators in $\mathcal{MC}$ may collude with each other and send malicious shard adjustment instructions to the system, which can compromise both the security and performance of the system. The specific malicious shard adjustment instructions and their impacts are shown in the following.

- Concentrates malicious validators in individual shards to create more malicious shards, which may launch double-spending cross-shard transaction attacks. The malicious shard behavior can be detected and punished during the shard auditing phase.
- A malicious $\mathcal{MC}$ can manipulate the allocation of validators and users to shards, leading to a greater imbalance in the workload across shards or an increased proportion of cross-shard transactions. Consequently, it results in reduced transaction processing efficiency and worsened transaction congestion. The degradation in system performance can be addressed when the next honest $\mathcal{MC}$ accurately adjusts the shards.

To quantitatively assess the impact of introducing $\mathcal{MC}$ to the system, we calculate the expected change in the overall cost of the system under different system states in Section VI. The evaluation results illustrate the significant effect of introducing $\mathcal{MC}$ in reducing the system cost, even in rare epochs where $\mathcal{MC}$ is malicious.



(a) K=30 (b) K=50

Fig. 4. Remove the dynamic shard management mechanism, system throughput varies with shard number, validator number, and the proportion of cross-shard transactions.

VI. Evaluation Results and Performance Analysis

A. Implemention and Settings

We implement an experimental prototype for performance evaluation using C++. To separately evaluate the effectiveness of our proposed approaches, we evaluate the system performance with and without the dynamic shard management mechanism. Besides, for comparison, we also implement a non-sharding prototype and a representative sharding prototype [28]. Similar to RapidChain [20], to simulate distributed nodes in P2P networks, we set the bandwidth of all connections between nodes to 20 Mbps and add random links latency of 100 ± 10 ms. Based on the historical transaction data of Bitcoin, we set the average size of a transaction to 300 bytes, and the average number of related shards for cross-shard transactions is set as 4.

B. Performance of Hybrid Sharding Consensus

We evaluate the performance of the hybrid sharding consensus by measuring the transaction throughput in transactions per second (TPS) for our refined system that removes the dynamic shard management mechanism. We set varying shard numbers K, validator numbers N, and the proportion of cross-shard transactions α and observe the corresponding impact on the TPS. As shown in Fig. 4, the shard numbers are set as 30 and 50 separately, validator numbers are set to $\{500, 700, 1000, 1500\}$, and the proportion of cross-shard transactions varies from $[0, 0.8]$. Through the evaluation result, it is evident that the throughput experiences a decline as the proportion of cross-shard transactions increases. Given a fixed shards number $K = 50$ and validator nodes number $N = 500$, when the proportion of cross-shard transactions α is set to 0, the throughput achieves 236.59 TPS. However, as the cross-shard transaction ratio α rises to 0.8, the throughput diminishes to 125.26 TPS, reflecting a reduction of approximately 47.06% compared to the non-cross-shard transactions scenario. This decline can be attributed to the involvement of all validator nodes in handling cross-shard transactions and the broadcast of local verification results across all nodes in the system, which brings additional communication time. Besides, the shards number and the validator nodes number decide the size of consensus groups, thereby impacting the efficiency of intra-shard transactions' processing and the overall system

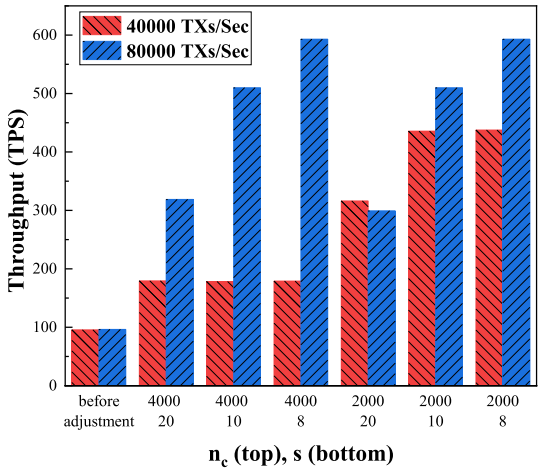


Fig. 5. Throughtput of the system varies with different system threshold parameters: n_c, s, and transaction arrival rate.

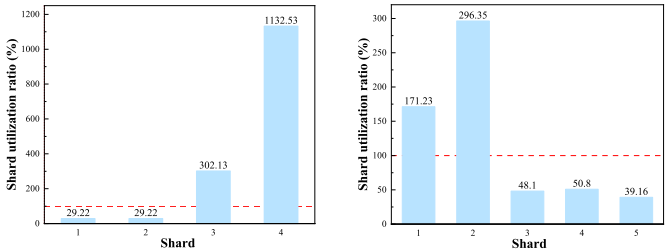
throughput. A smaller consensus group size corresponds to a higher system throughput.

C. Comprehensive Performance With Dynamic Shard Management Mechanism

In this subsection, we conduct comprehensive simulations to evaluate system performance with the dynamic shard management mechanism. In the first group of simulations, we investigate the impact of the congested shards threshold transaction number n_c, and the minimum number of validators per shard s on the efficiency of the dynamic shard management mechanism under various transaction arrival rates. It is suggested to set n_c based on the average number of transactions in a block n_{tx}, the average block generation time t_b, and the desired transaction confirmation latency time (t), using the equation $n_c = \frac{n_{tx} \cdot t}{t_b}$, and set s based on the security needs. A higher value of s corresponds to a smaller probability of an adversary gaining control over a shard. By varying the values of n_c, s, and the transaction arrival rate, while keeping other parameters constant, we measure the transaction throughput in terms of transactions per second (TPS).

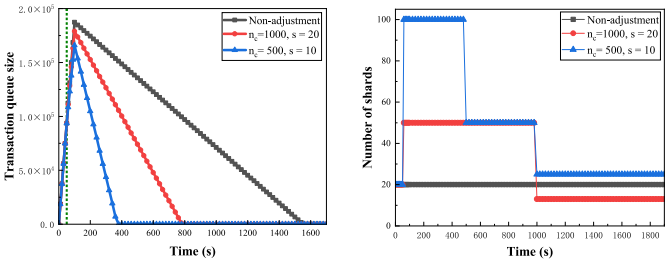
For the initialization, 20000 users and 500 validators are evenly divided into 10 shards, the transaction arrival rate is set to 40000, 80000 transactions per second separately, n_c is set to 2000, 4000, and s is set to $20, 10, 8$. The result is illustrated in Fig. 5, under a fixed transaction arrival rate, a smaller value for n_c and s, leads to more shard splits and therefore a higher throughput. Under fixed n_c and r, a faster transaction arrival rate will bring more unprocessed transactions per shard and a greater number of shard splits, thereby achieving higher throughput. The result shows that the dynamic shard management mechanism can flexibly provide higher throughput under stronger transaction processing requirements. Besides, by adjusting the parameters n_c and s, one can strike a balance between ensuring the security of individual shards and maximizing the overall throughput of the system.

We then study the ratio of shard utilization, defined as the transaction arrival rate divided by the shard throughput, both



(a) Before shard adjustment. (b) After shard adjustment.

Fig. 6. Shards utilization before and after shard adjustment.



(a) Queue size of the transaction pool. (b) The number of shards.

Fig. 7. Queue size of the transaction pool and the number of shards vary with time after a transaction burst.

before and after the shard adjustment process. The number of shards is set as 10, with transaction arrival rates varying from $[50, 2000]$, to simulate unbalanced workload scenarios. The threshold parameter n_c is set to 500, r is set to 0.2, and s is set as 20. We select four shards in the above process and calculate the number of unprocessed intra-shard transactions divided by the intra-shard transaction processing speeds as the shard utilization ratio, as shown in Fig. 6. The average distance to the perfect 100% utilization is shortened from 3.44 to 0.86, indicating the efficacy of the dynamic shard management mechanism in improving shard utilization.

In the next group of simulations, to evaluate the performance of our system in efficiently handling transaction bursts under different managing parameters n_c and s, we use $200,000$ transactions to feed the transaction pool with an arrival rate of 2000 transactions per second. We set the number of users to 20000, the number of validators as 1000, the initial number of shards K as 20, parameter r to 0.2, and record the queue size of the transaction pool and the shards numbers in the system. There are three system settings in these simulations: a). without shard adjustment. b). n_c is set to 1000, s is set to 20. c). n_c is set to 500, s is set to 10. For the last two situations, we adjust the shards at time $= 50, 200, 1000$ seconds. As shown in Fig. 7a, the queue size keeps growing when the transactions are continually injected at the first hundred seconds, and after all the transactions are consumed, the queue size begins to shrink for all three settings. For comparison, we can see it takes about 1551.9 seconds to process all the transactions for the non-adjustment situation, and only 784.3 seconds and 371.5 seconds for the last two settings, reducing the processing time by 49.5% and 76.1% respectively. Besides, Fig. 7b illustrates the shard numbers in the system over time. In the first adjustment when there is a transaction

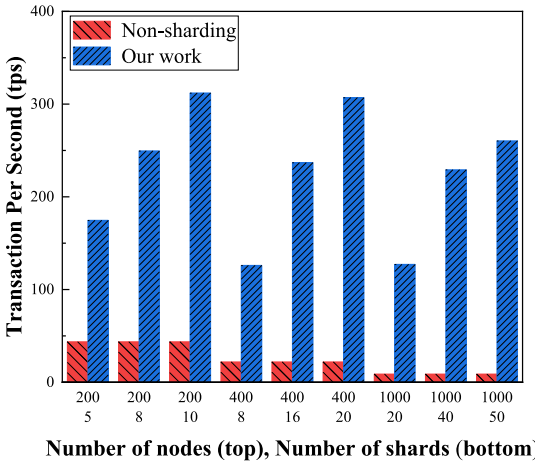


Fig. 8. Compare the transaction throughput with the traditional non-sharding scheme.

burst, situations b and c split shards into 50 shards and 100 shards respectively. As the transaction queue subsequently diminishes below the predetermined threshold, situations b and c merge shards and reduce the shards number to 13 and 25 respectively. The experiment results demonstrate the dynamic shard management mechanism can effectively accelerate the processing for congested transactions and also reduce the number of underloaded shards when the transaction load decreases.

We calculate the overall cost of the system using the model described in Section IV-E, considering different system states such as peak, plateau, and low transaction periods. Specifically, we set the transaction arrival rate range as $\{50, 500, 2000\}$, with weight factor w_1 set to 10 and w_2 set to 100 for normalization. The system parameters used in the dynamic shard management mechanism are set to $n_c = 1000$, $r = 0.2$, and $s = 20$. We test the transaction processing rate and calculate the overall system cost before and after the adjustment. In addition to the normal shard adjustment based on the dynamic shard management mechanism, we also consider a malicious adjustment carried out by a malicious managing committee. The malicious adjustment aims to maximize the number of malicious shards and increase the consensus group size for other normal shards, thus reducing the transaction processing efficiency. The results are shown in Table I, where C_0 represents the overall system cost before the adjustment, C_1 represents the overall cost after a normal adjustment, and C_2 represents the overall cost after a malicious adjustment. Furthermore, we calculate the expected change in the overall system cost by introducing the managing committee as follows:

$$E(\Delta C) = (1 - p) \cdot \Delta C_{\text{overall}} - p \cdot \Delta C'_{\text{overall}},$$

where p represents the probability that most validators in the managing committee are malicious, $\Delta C_{\text{overall}}$ represents the reduction in the overall cost achieved by adjusting the shards normally according to the dynamic shard management mechanism, and $\Delta C'_{\text{overall}}$ represents the increment in the overall cost when the managing committee is malicious. Table I shows that introducing the managing committee- has

TABLE I
THE SYSTEM OVERALL COST UNDER DIFFERENT TRANSACTION ARRIVAL RATES BEFORE/AFTER ADJUSTMENTS

Transaction arrival rate	50	500	2000
C_0	0.87	42.85	198.81
C_1	0.53	27.14	90.83
C_2	79.56	285.57	972.25
$E(\Delta C)/C_0$	0.23%	34.04%	52.38%

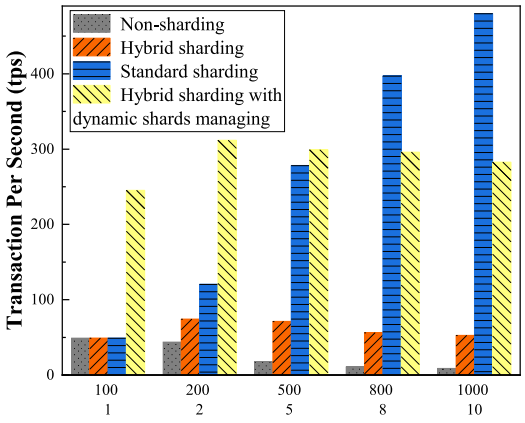


Fig. 9. Comparison of transaction throughput of non-sharding scheme, hybrid sharding scheme, the standard Sharding scheme and hybrid sharding scheme with dynamic shard management.

a positive impact on reducing the overall system cost under different system statuses, particularly during transaction bursts.

To figure out our work's transaction processing capacity and efficiency relative to these existing approaches, we compare the throughput of the proposed system with the traditional single blockchain (non-sharding) scheme and a representative sharding scheme named Monoxide [28], which uses relay transaction mechanism to process cross-shard transactions.

We first test the system transaction throughput under different shard numbers and consensus node numbers and compare our work with the traditional single blockchain (non-sharding) scheme. The number of consensus nodes ranges from $\{200, 400, 1000\}$, and the number of shards varies from $[5, 50]$. To achieve a fair comparison of the system structures' influence on the throughput, the non-sharding system takes the same consensus process as our system's intra-shard consensus. As shown in Fig. 8, the throughput of the non-sharding system drops seriously as the number of nodes increases. In contrast, our system has a much higher increasing throughput with the number of shards. When there are 1000 consensus nodes in the system, the TPS of a traditional non-sharding scheme is 8.74, while ours (50 shards) is 260.39, which is about 30 times its throughput.

D. Comparison Results

Then, we compare our work with the representative sharding scheme. There are two main differences between our system and the standard sharding system. One is the shard components, Monoxide is a static sharding scheme

with a fixed number of shards, while ours uses a dynamic shard management mechanism. The other difference is the way to handle cross-shard transactions. Our system uses the refined global consensus to reach joint cross-shard transaction processing, while Monoxide uses relay transactions to process cross-shard transactions, which can be referred to in section II. To evaluate the throughput under different system capacities, we set the number of consensus nodes to vary from $\{100, 200, 500, 800, 1000\}$, the initial size of each shard is set as 100, and the smallest shard size s is set as 20. As shown in Fig. 9, both our solution and the standard sharding scheme can improve the system throughput of a traditional non-sharding blockchain system by multiple times as the number of nodes increases. Besides, note that standard sharding systems have limits on the minimum size of shards to reduce the possibility of shard corruption caused by consensus shrinkage. In contrast, in our solution, the security of shards is guaranteed by the security of the entire system, therefore enabling to splitting of smaller shards. Therefore, when the number of nodes remains at or below 500, our solution exhibits a superior throughput compared to the standard sharding scheme, primarily due to the more and smaller shards employed in our approach. However, as the number of nodes grows, the standard sharding scheme demonstrates greater efficiency in terms of performance. This discrepancy arises from the tradeoff we made to prioritize and enhance the security of the sharding system.

E. Discussion

1) Theoretical Performance Analysis and Comparison: Assuming a system consists of N validators, divided into K shards, with n transactions, where the proportion of cross-shard transactions is α, and the average number of related shards for each cross-shard transaction is x. Using our proposed protocol, the average time complexity for transaction processing is $O\left(n \cdot N \cdot (\frac{(1-\alpha)}{K^2} + \frac{\alpha \cdot x}{K})\right)$. In contrast, the average time complexity for transaction processing in classical blockchain sharding protocols is $O(\frac{n \cdot N}{K^2})$ and in non-sharding single-chain systems, it is $O(n \cdot N)$. Therefore, in the worst case, where the proportion of cross-shard transactions approaches 1, the time complexity for our solution is $O(\frac{n \cdot N \cdot x}{K})$, still resulting in a K/x fold reduction in time compared to non-sharding single-chain systems. In the best case, when α approaches 0, the proposed protocol has the same time complexity as classical blockchain sharding protocols, both being $O(\frac{n \cdot N}{K^2})$.

2) Optimal Application Scenario: The proposed solution improves transaction processing efficiency by performing system sharding during transaction bursts and merging shards to do global auditing when the system is idle. It is most applicable for systems with both high and low transaction peaks and a sufficient average computational power. Additionally, compared to most existing sharding schemes that ensure security through probabilistic methods [20], [27], our proposed solution is more tolerant towards the probability of malicious shards. Therefore, it supports smaller shard sizes, allowing blockchain systems with fewer nodes to be split into more parallel shards, resulting in greater efficiency improvements.

VII. Conclusion

In this work, we presented a dynamic blockchain sharding scheme with stronger security. Our core objective is to address the consensus shrinkage and shards isolation issue within blockchain sharding systems without compromising the efficiency improvements brought by sharding. To achieve this goal, we first proposed a hybrid sharding consensus mechanism that seamlessly integrates global consensus into multiple parallel intra-shard consensuses. By leveraging global consensus, we fixed the sharding security issue by facilitating shards auditing, which defended against cross-shard double spending attacks from malicious shards, and achieved cross-shard transaction atomicity. Moreover, we designed a dynamic shard management mechanism, which reduces transaction congestion, improves the utilization of shards, and avoid security degradation caused by over-sharding. The security analysis proved our system can achieve consensus safety and liveness even with corrupted shards. Finally, the evaluation results show that our work achieved significant improvement in increasing transaction throughput and handling transaction bursts, surpassing the performance of traditional blockchain systems.

Acknowledgment

The authors would like to thank the anonymous referees for their invaluable suggestions that have led to the present improved version of this article.

References

[1] K. Wüst and A. Gervais, "Do you need a blockchain?" in *Proc. Crypto Valley Conf. Blockchain Technol. (CVCBT)*, Jun. 2018, pp. 45–54.

[2] L.-Y. Yeh, W.-H. Hsu, and C.-Y. Shen, "GDPR-compliant personal health record sharing mechanism with redactable blockchain and revocable IPFS," *IEEE Trans. Dependable Secure Comput.*, early access, 2023, doi: 10.1109/TDSC.2023.3325907.

[3] B. Chen, T. Xiang, D. He, H. Li, and K. R. Choo, "BPVSE: Publicly verifiable searchable encryption for cloud-assisted electronic health records," *IEEE Trans. Inf. Forensics Security*, vol. 18, pp. 3171–3184, 2023.

[4] G. D. Bashar, J. Holmes, and G. G. Dagher, "ACCORD: A scalable multileader consensus protocol for healthcare blockchain," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 2990–3005, 2022.

[5] J. Xu et al., "HealthChain: A blockchain-based privacy preserving scheme for large-scale health data," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 8770–8781, Oct. 2019.

[6] Y. Miao, Z. Liu, H. Li, K. R. Choo, and R. H. Deng, "Privacy-preserving Byzantine-robust federated learning via blockchain systems," *IEEE Trans. Inf. Forensics Security*, vol. 17, pp. 2848–2861, 2022.

[7] J. Weng, J. Weng, J. Zhang, M. Li, Y. Zhang, and W. Luo, "DeepChain: Auditable and privacy-preserving deep learning with blockchain-based incentive," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 5, pp. 2438–2455, Sep. 2021.

[8] C. Lin, D. He, X. Huang, and K. R. Choo, "OBFP: Optimized blockchain-based fair payment for outsourcing computations in cloud computing," *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 3241–3253, 2021.

[9] J. Zou, D. He, S. Zeadally, N. Kumar, H. Wang, and K. R. Choo, "Integrated blockchain and cloud computing systems: A systematic survey, solutions, and challenges," *ACM Comput. Surv.*, vol. 54, no. 8, pp. 1–36, Nov. 2022.

[10] K. Xue, X. Luo, Y. Ma, J. Li, J. Liu, and D. S. L. Wei, "A distributed authentication scheme based on smart contract for roaming service in mobile vehicular networks," *IEEE Trans. Veh. Technol.*, vol. 71, no. 5, pp. 5284–5297, May 2022.

[11] K. Xue, X. Luo, H. Tian, J. Hong, D. S. L. Wei, and J. Li, "A blockchain based user subscription data management and access control scheme in mobile communication networks," *IEEE Trans. Veh. Technol.*, vol. 71, no. 3, pp. 3108–3120, Mar. 2022.

[12] A. Vangala, A. K. Das, A. Mitra, S. K. Das, and Y. Park, "Blockchain-enabled authenticated key agreement scheme for mobile vehicles-assisted precision agricultural IoT networks," *IEEE Trans. Inf. Forensics Security*, vol. 18, pp. 904–919, 2023.

[13] L. Zhou, A. Fu, G. Yang, Y. Gao, S. Yu, and R. H. Deng, "Fair cloud auditing based on blockchain for resource-constrained IoT devices," *IEEE Trans. Dependable Secure Comput.*, vol. 20, no. 5, pp. 4325–4342, May 2023.

[14] Y. Jiang and J. Zhang, "Distributed detection over blockchain-aided Internet of Things in the presence of attacks," *IEEE Trans. Inf. Forensics Security*, vol. 18, pp. 3445–3460, 2023.

[15] V. Mishra and D. Sadhya, "Height and punishment: Toward accountable IoT blockchain with network sanitization," *IEEE Trans. Inf. Forensics Security*, vol. 18, pp. 5665–5677, 2023.

[16] D. Khan, L. T. Jung, and M. A. Hashmani, "Systematic literature review of challenges in blockchain scalability," *Appl. Sci.*, vol. 11, no. 20, p. 9372, Oct. 2021.

[17] L. Zhang, H. Xu, O. Onireti, M. A. Imran, and B. Cao, "How much communication resource is needed to run a wireless blockchain network?" *IEEE Netw.*, vol. 36, no. 1, pp. 128–135, Jan. 2022.

[18] M. Kuzlu, M. Pipattanasomporn, L. Gurses, and S. Rahman, "Performance analysis of a hyperledger fabric blockchain framework: Throughput, latency and scalability," in *Proc. IEEE Int. Conf. Blockchain (Blockchain)*, Jul. 2019, pp. 536–540.

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

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

[21] 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. Int. Conf. Manage. Data (SIGMOD)*, 2019, pp. 123–140.

[22] A. Liu et al., "CHERUBIM: A secure and highly parallel cross-shard consensus using quadruple pipelined two-phase commit for sharding blockchains," *IEEE Trans. Inf. Forensics Security*, vol. 19, pp. 3178–3193, 2024.

[23] Y. Xu, J. Zheng, B. Düdder, T. Slaats, and Y. Zhou, "A two-layer blockchain sharding protocol leveraging safety and liveness for enhanced performance," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2024, doi: 10.14722/ndss.2024.24006.

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

[25] Y. Liu et al., "Building blocks of sharding blockchain systems: Concepts, approaches, and open problems," *Comput. Sci. Rev.*, vol. 46, pp. 1–44, Nov. 2022, Art. no. 100513.

[26] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An overview of blockchain technology: Architecture, consensus, and future trends," in *Proc. IEEE Int. Congr. Big Data (BigData Congress)*, Jun. 2017, pp. 557–564.

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

[28] J. Wang and H. Wang, "Monoxide: Scale out blockchains with asynchronous consensus zones," in *Proc. 16th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2019, Feb. 2019, pp. 95–112.

[29] P. Zheng, Q. Xu, Z. Zheng, Z. Zhou, Y. Yan, and H. Zhang, "Meepo: Sharded consortium blockchain," in *Proc. IEEE 37th Int. Conf. Data Eng. (ICDE)*, Apr. 2021, pp. 1847–1852.

[30] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proc. 3rd USENIX Symp. Operating Syst. Design Implement.*, 1999, pp. 173–186.

[31] S. Nakamoto. (2008). *Bitcoin: A Peer-to-Peer Electronic Cash System*. Accessed: Dec. 2023. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[32] G. Wood. (2022). *ETHEREUM: A Secure Decentralised Generalised Transaction Ledger*. Accessed: Dec. 2023. [Online]. Available: https://cryptodeep.ru/doc/paper.pdf

[33] L. N. Nguyen, T. D. T. Nguyen, T. N. Dinh, and M. T. Thai, "OptChain: Optimal transactions placement for scalable blockchain sharding," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2019, pp. 525–535.

[34] Z. Hong, S. Guo, P. Li, and W. Chen, "Pyramid: A layered sharding blockchain system," in *Proc. Int. Conf. Comput. Commun. (INFOCOM)*, 2021, pp. 1–10.

[35] H. Huang et al., "BrokerChain: A cross-shard blockchain protocol for account/balance-based state sharding," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, May 2022, pp. 1968–1977.

[36] Y. Liu et al., "A flexible sharding blockchain protocol based on cross-shard Byzantine fault tolerance," *IEEE Trans. Inf. Forensics Security*, vol. 18, pp. 2276–2291, 2023.

[37] M. Andrychowicz and S. Dziembowski, "Pow-based distributed cryptography with no trusted setup," in *Advances in Cryptology—CRYPTO 2015*. Berlin, Germany: Springer, 2015, pp. 379–399.

[38] S. Micali, S. Vadhan, and M. Rabin, "Verifiable random functions," in *Proc. 40th Annu. Symp. Found. Comput. Sci.*, 1999, pp. 120–130.

[39] S. King and S. Nadal. (Aug. 2012). *PPCoin: Peer-to-Peer Crypto-Currency With Proof-of-Stake*. [Online]. Available: https://bitcoin.peryaudo.org/vendor/peercoin-paper.pdf

[40] E. Syta et al., "Keeping authorities," in *Proc. IEEE Symp. Secur. Privacy (S&P)*, Jul. 2016, pp. 526–545.

[41] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing Bitcoin security and performance with strong consistency via collective signing," in *Proc. 25th USENIX Secur. Symp.*, Aug. 2016, pp. 279–296.

[42] G. Karypis and V. Kumar, "A fast and high quality multilevel scheme for partitioning irregular graphs," *SIAM J. Sci. Comput.*, vol. 20, no. 1, pp. 359–392, Jan. 1998.

Meiqi Li received the B.S. degree in information security from the School of Cyber Science and Technology, University of Science and Technology of China (USTC), in 2021, where she is currently pursuing the Ph.D. degree in information security. Her research interests include blockchain, network security, and applied cryptography.

Xinyi Luo (Graduate Student Member, IEEE) received the B.S. degree in information security from the School of the Gifted Young, University of Science and Technology of China (USTC), in 2020. She is currently pursuing the Ph.D. degree in information security with the School of Cyber Science and Technology, USTC. Her research interests include blockchain, network security, and applied cryptography.

Kaiping Xue (Senior Member, IEEE) received the bachelor's degree from the Department of Information Security, University of Science and Technology of China (USTC), in 2003, and the Ph.D. degree from the Department of Electronic Engineering and Information Science (EEIS), USTC, in 2007.

From May 2012 to May 2013, he was a Post-Doctoral Researcher with the Department of Electrical and Computer Engineering, University of Florida. Currently, he is a Professor with the School of Cyber Science and Technology, USTC. He is also the Director of the Network and Information Center, USTC. His research interests include next-generation internet architecture design, transmission optimization, and network security. He serves on the editorial board for several journals, including IEEE TRANSACTIONS ON DEPENDABLE AND SECURE COMPUTING, IEEE TRANSACTIONS ON WIRELESS COMMUNICATIONS, and IEEE TRANSACTIONS ON NETWORK AND SERVICE MANAGEMENT. He is an IET Fellow.

Yingjie Xue received the bachelor's degree from the Department of Information Security, University of Science and Technology of China (USTC), in 2015, the master's degree from the Department of Electronic Engineering and Information Science (EEIS), University of Science and Technology of China (USTC), in 2018, and the Ph.D. degree in computer science from Brown University, in 2023. Currently, she is an Assistant Professor with the Financial Technology Thrust, The Hong Kong University of Science and Technology (Guangzhou). Her research interests include blockchain, such as blockchain interoperability, security and privacy, and decentralized finance.

Wentuo Sun (Graduate Student Member, IEEE) received the bachelor's degree from the School of Cyber Science and Technology, University of Science and Technology of China (USTC), in 2020, where he is currently pursuing the Ph.D. degree in information security. His research interests include blockchain, network security, and applied cryptography.

Jian Li (Senior Member, IEEE) received the bachelor's degree from the Department of Electronics and Information Engineering, Anhui University, in 2015, and the Ph.D. degree from the Department of Electronic Engineering and Information Science (EEIS), University of Science and Technology of China (USTC), in 2020. From November 2019 to November 2020, he was a Visiting Scholar with the Department of Electronic and Computer Engineering, University of Florida. From December 2020 to December 2022, he was a Post-Doctoral Researcher with the School of Cyber Science and Technology, USTC. He is currently an Associate Researcher with the School of Cyber Science and Technology, USTC. His research interests include future internet technologies, network security, and quantum networks. He serves as an Editor for *China Communications*.