ContribChain: A Stress-Balanced Blockchain Sharding Protocol with Node Contribution Awareness

Xinpeng Huang*†‡, Wanqing Jie*†‡, Shiwen Zhang*†, Haofu Yang*†, Wangjie Qiu*†‡, Qinnan Zhang*†, Huawei Huang§, Zehui Xiong¶, Shaoting Tang*†‡, Hongwei Zheng‖, and Zhiming Zheng*†‡
*Institute of Artificial Intelligence, Beihang University, Beijing, China
†Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing, Beihang University, Beijing, China
‡Zhongguancun Laboratory, Beijing, China
§School of Software Engineering, Sun Yat-Sen University, China
¶Singapore University of Technology and Design, Singapore
‖Beijing Academy of Blockchain and Edge Computing (BABEC), Beijing, China

{huangxp, wanqingjie, zhangshiwen}@buaa.edu.cn, {2113824}@mail.nankai.edu.cn, {wangjieqiu, zhangqn}@buaa.edu.cn, {huanghw28}@sysu.edu.cn, {zehui_xiong}@sutd.edu.sg, {tangshaoting}@buaa.edu.cn, {hwzheng, zzheng}@pku.edu.cn

***Abstract*—Existing blockchain sharding protocols have focused on eliminating imbalanced workload distributions. However, even with workload balance, disparities in processing capabilities can lead to differential stress among shards, resulting in transaction backlogs in certain shards. Therefore, achieving stress balance among shards in the dynamic and heterogeneous environment presents a significant challenge of blockchain sharding. In this paper, we propose ContribChain, a blockchain sharding protocol that can automatically be aware of node contributions to achieve stress balance. We calculate node contribution values based on the historical behavior to evaluate the performance and security of nodes. Furthermore, we propose node allocation algorithm NACV and account allocation algorithm P-Louvain, which both match shard performance with workload to achieve stress balance. Finally, we conduct extensive experiments to compare our work with state-of-the-art baselines based on real Ethereum transactions. The evaluation results show that P-Louvain reduces allocation execution time by 86% and the cross-shard transaction ratio by 7.5%. Meanwhile, ContribChain improves throughput by 35.8% and reduces the cross-shard transaction ratio by 16%.**

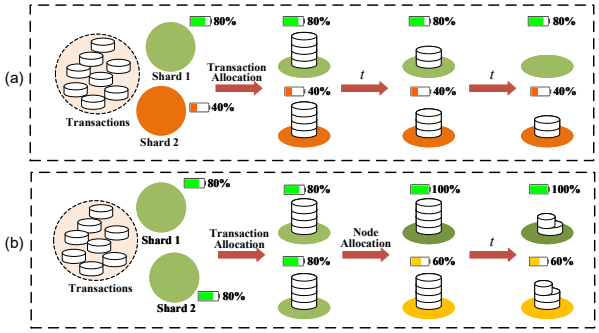


Fig. 1: The stress imbalance issue in dynamic environments. Transaction backlogs occur in shards with low processing capabilities due to the lack of consideration of shard performance during transaction and node allocation.

I. Introduction

With the rapid proliferation of distributed nodes constantly expanding, blockchain sharding has emerged as a promising a prominent technology used to improve the performance of blockchains [1]–[6]. The core idea is to partition the whole blockchain network into many subnetworks, known as shards [2]. These shards process transactions in parallel, hence enhancing the transaction throughput of the blockchain. In addition, sharded blockchains undergo periodic reconfiguration of shards [3], [4], [7] by altering the nodes within each shard to guarantee the security of whole blockchain network.

Currently, a significant amount of research has focused on reducing the ratio of cross-shard transactions and achieving load balance among shards to improve the performance of sharded blockchains [1], [6], [8], [9]. By leveraging the parallel advantages of sharding technology, these optimizations aim to increase system throughput. However, existing methods overlook the node composition within shards, focusing primarily on the transaction layer. When performance disparities exist among shards, even if their loads are balanced, their stress levels may not be. We define shard stress as the alignment between a shard's transaction processing capacity and its workload. Fig. 1 (a) and (b) highlight the impact of neglecting shard performance in transaction and node allocation, leading to transaction backlogs in shards with lower processing capabilities. This issue is referred to as stress imbalance among shards, occurring when a shard's processing capacity and load do not align.

* Wangjie Qiu and Qinnan Zhang are the **corresponding authors**.

Motivation. Some studies [6], [8], [9] employ graph algorithms or community detection techniques for transaction allocation to reduce cross-shard transaction ratios and achieve load balancing among shards. However, in the account/balance model, the presence of popular accounts complicates the achievement of perfect load balance through transaction allocation alone. Recent work [1] employs state splitting and aggregation to allow users to hold accounts across multiple shards, while [10], [11] propose account migration mechanisms for rapid account reallocation. Nonetheless, these approaches are still focused on the transaction layer, leading to the two problems mentioned in Fig. 1.

Regarding shard reconfiguration, several studies [12]–[15] have applied reinforcement learning for adaptive node allocation. However, these methods have limitations: they require large amounts of data and long training time, and they do not address cross-shard transaction problems. Furthermore, these works lack consideration for shard load circumstances.

To summarize, current performance optimizations focus on either node or transaction allocation, without accounting for their combined impact. Additionally, these optimizations overlook the relationship between processing capabilities and load, resulting in stress imbalances among shards. To address this, we propose a stress-balanced sharding protocol that integrates both processing capabilities and load.

Challenges. In sharded blockchains, transaction processing within shards relies on consensus protocols among nodes. Consequently, the frequent ingress and egress of nodes lead to dynamic variations in a shard's transaction processing capability. Thus, accurately assessing node performance is crucial for estimating shard performance. However, developing a robust methodology to evaluate node performance is a significant challenge.

To tackle these challenges, we introduce ContribChain, a novel stress-balanced blockchain sharding protocol featuring dynamic node and account allocation algorithms based on node contribution values. The main **contributions** of this work are summarized as follows:

- ***A stress-balanced blockchain protocol (ContribChain):*** We propose dynamically updated node contribution values to comprehensively evaluate node performance and security. Additionally, our account and node allocation algorithms assess shard stress at both the node and transaction levels, ensuring stress balance across shards.
- ***Node allocation algorithm based on node contribution values (NACV) and performance-based account allocation algorithm (P-Louvain):*** NACV aims to balance the performance of allocated nodes and shard load, factoring in security (Section III-C). P-Louvain reduces the cross-shard transaction ratio while ensuring alignment between shard load and performance (Section III-D).
- ***System implementation:*** We implement P-Louvain and ContribChain on the open-source blockchain testbed *BlockEmulator* [16]. Evaluation results demonstrate that, compared to state-of-the-art baselines, P-Louvain reduces allocation execution time by 86% and the cross-shard transaction ratio by 7.5%. Meanwhile, ContribChain improves throughput by 35.8% and reduces the cross-shard transaction ratio by 16%. ContribChain also shows superior stress balance and security performance.

The remainder of this paper is organized as follows: Section II reviews related work, Section III describes the protocol design, Section IV analyzes the security and other properties of ContribChain, Section V presents performance evaluation results, and Section VI concludes the paper.

II. Related Work

A. Blockchain Sharding

To address scalability limitations in blockchain systems, numerous sharding solutions have been proposed [17]. A review of key approaches is provided below. In 2016, Elastico [2] introduced the first blockchain sharding scheme, leveraging Proof of Work (PoW) for node selection and periodic reallocation. OmniLedger [3] mitigates node storage burdens by employing full sharding, where each shard is responsible for storing non-overlapping ledger data. RapidChain [4] adopts full sharding and reconfigures nodes based on the Cuckoo rule [18], [19]. Later works [1], [6], [8] have expanded upon this reconfiguration model, focusing on reducing the costs associated with cross-shard transactions and ensuring more balanced workload distribution across shards in the context of full sharding.

B. Node Allocation

Node allocation is critical in blockchain sharding, involving both node assignment and reassignment. Regularly rotating nodes is essential to limit adversarial control, such as the 1/3 threshold in PBFT [20]. Additionally, the allocation procedure must be unpredictable, unbiased and publicly verifiable [21].

Node allocation strategies can be categorized into random, rule-based, and adaptive reconfiguration [12], [21]. Pseudo-random number generators are used in [3], [7] to select nodes for reassignment. Other methods [22], [23] use criteria like activity level or chronological order. However, transferring nodes between shards incurs communication overhead due to ledger synchronization [11]. RapidChain [4] mitigates this by transferring only a subset of nodes at a time, known as the Cuckoo exchange mechanism. Despite the advances, the Cuckoo exchange mechanism lacks clear criteria for active nodes and does not consider shard performance, complicating post-reconfiguration performance assessments.

C. Account Allocation

Account allocation assigns accounts to specific shards, with each shard handling transactions related to its accounts. Early methods, such as [5], [24], used hash-based approaches for transaction allocation, which could lead to uneven workloads and excessive cross-shard transactions.

To address these issues, [25], [26] model transactions or accounts as nodes in a graph, leveraging historical data and graph theory for allocation. Techniques in [8], [9] enhance community detection for more effective account assignment.

BrokerChain [6] introduces brokers—users with accounts across multiple shards—to facilitate cross-shard transactions. Estuary [1] distributes accounts across shards, reducing cross-shard transactions. [27] employs Deep Reinforcement Learning (DRL) to optimize state placement. While these approaches enhance account allocation, they fail to consider performance disparities among shards, and thus do not achieve stress balance.

III. ContribChain Protocol Design

We introduce ContribChain, a protocol designed to assess node contributions and achieve stress balance through dynamic allocation of nodes and accounts. This section outlines the system model, workflow, and key algorithms.

A. System Model And Workflow

ContribChain operates on the account/balance model, consisting of two types of shards: one A-Shard and K W-Shards. The protocol is designed to withstand a gradually adapting Byzantine adversary. The specific definitions are as follows:

- **W-Shard:** Each W-Shard processes client transactions in parallel. W-Shards use PBFT [20] to achieve intra-shard consensus and ensures atomicity of cross-shard TXs through the Relay Transaction mechanism [5].
- **A-Shard:** The A-Shard is responsible for allocating nodes and accounts to W-Shards and updating the global contribution values of all nodes.
- **Node Contribution Values:** Node contribution values consist of security and performance contribution values, calculated based on historical behaviors. Stage contribution values are generated at each epoch to update the global contribution values.
- **S-Blockchain:** The S-Blockchain in A-Shard records system-wide state changes, including node and account allocation results, and global node contribution values.
- **W-Blockchain:** Each W-Shard maintains a W-Blockchain to record transaction processing results.

As illustrated in Fig. 2, ContribChain operates in *Epochs* [3], with the following workflow:

Phase 1: Identity Establishment. To prevent Sybil attacks [28], nodes must obtain valid identities for the current epoch. Following previous methods [3], [4], [6], nodes use Verifiable Random Functions (VRF) to generate randomness, which is then used in a Proof of Work (PoW) protocol to solve a hash puzzle. PoW may also be replaced with Proof of Stake (PoS) for energy efficiency.

Phase 2: Node Allocation. In Epoch 0, node identities are mapped to integers $\phi = [0, 2^{256} - 1]$ and divided into $K + 1$ intervals. Nodes in the $(K + 1)$th interval join the A-Shard, while others join the W-Shards [1]. From Epoch 1 onwards, the same method is first used to determine nodes in A-Shard. A-Shard then runs node allocation algorithm based on node contribution value (NACV) (Section III-C) to update the nodes in W-Shards.

Phase 3: Account Allocation. Every f epochs, the A-Shard runs the P-Louvain algorithm (Section III-D) to allocate accounts to W-Shards.

Phase 4: Shard Update. The A-Shard packages node and account allocation results into a State Block. After consensus, this block is added to the S-Blockchain and broadcast to all consensus nodes. W-Shards update nodes and accounts based on this block and generate the Genesis Block for the current epoch (Section III-E).

Phase 5: Shard Consensus. EW-Shards aggregate transactions into TX-Blocks. The leader of each W-Shard adds committed TX-Blocks to its W-Blockchain and sends them to the A-Shard in real-time. Then the leader records nodes' voting behaviors and block submission results.

Phase 6: Data Collection. Each W-Shard calculates the stage contribution values of nodes (Section III-B) and packages them with pending transactions ($TX_{pending}$) into a Shard Summary Block. After consensus within the W-Shard, this block is sent to the A-Shard, which updates global contribution values. The contribution values, $TX_{pending}$, and other data are aggregated into a System Summary Block and added to the S-Blockchain (Section III-E).

B. Node contribution values

Node contribution values consist of performance contributions and security contributions, calculated based on voting behavior, node identity, and block submission results. At each epoch, nodes will have stage contribution values used to update the global node contribution values.

The stage performance contribution value $\Delta p_e(n_i)$ quantifies the TPS (Transactions Per Second) contributed by node i during Epoch e. Successful block submission is defined as a *yes* vote, while failure is a *no* vote. For failed block submissions, the correct behavior is defined as a *no* vote, and the wrong behavior as a *yes* vote. The formula for $\Delta p_e(n_i)$ is:

$$\Delta p_e(n_i) = \frac{\sum_{j=1}^{N_{\text{ALL}}} \left[\frac{TX_j}{n_{R_j}} e_{i,j} - \delta_j \frac{TX_j}{n - n_{R_j}} (1 - e_{i,j}) \right]}{t_e}, \tag{1}$$

$$\delta_j = \begin{cases} 0, & \text{if block } j \text{ submission succeeded;} \\ 1, & \text{if block } j \text{ submission failed.} \end{cases} \tag{2}$$

$$e_{ij} = \begin{cases} 1, & \text{if node } i \text{ behaved correctly;} \\ 0, & \text{if node } i \text{ behaved incorrectly.} \end{cases} \tag{3}$$

Among them, $TX(i)$ represents the transactions contributed by node i. t_e denotes the duration of Epoch e. N_{ALL} denotes the total number of successful and failed block submissions in Epoch e. n_{Rj} denotes the number of correctly behaving nodes during the consensus for block j. TX_j denotes the transaction number in block j.

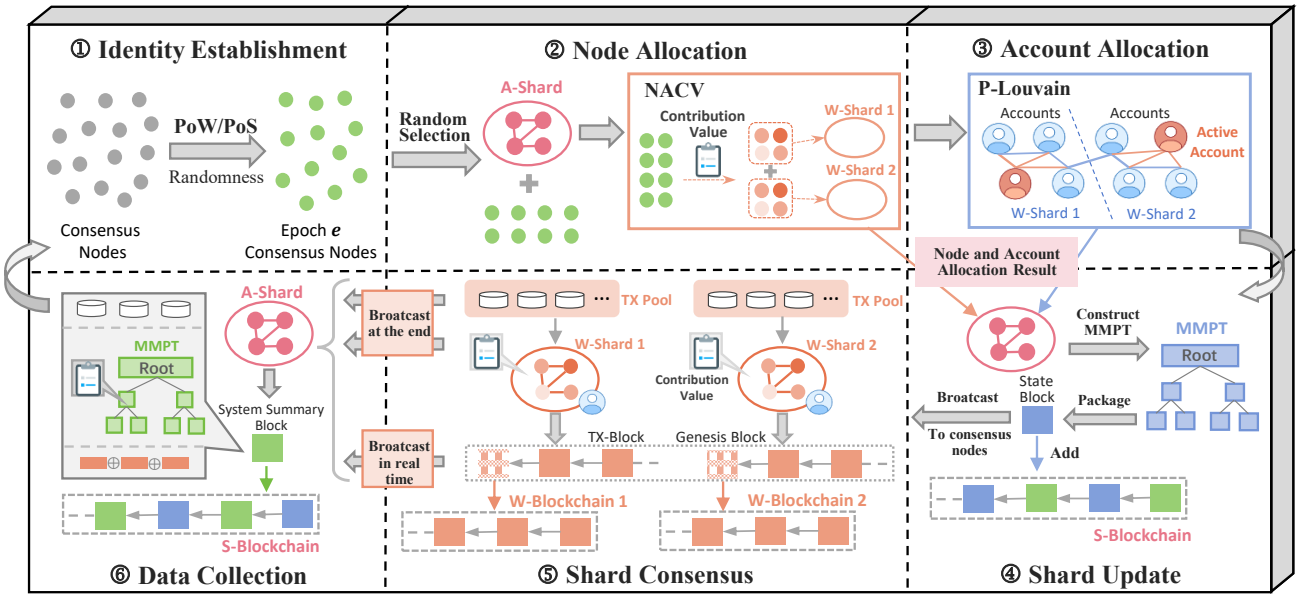


Fig. 2: Workflow of ContribChain during epoch e. In ContribChain, Phase 3 is executed every f epochs.

The stage security contribution value, $\Delta s_e(n_i)$, reflects the security performance of node i in Epoch e, and is computed as:

$$\Delta s_e(n_i) = \frac{\mu \cdot (\lambda \cdot N_{MRi} + N_{FRi}) - \theta \cdot (\lambda \cdot N_{MWi} + N_{FWi})}{\lambda \cdot (N_{MRi} + N_{FRi}) + N_{MWi} + N_{FWi}}. \tag{4}$$

Among them, N_{MRi} and N_{MWi} are the numbers of correct and incorrect behaviors by node i as a leader, and N_{FRi} and N_{FWi} are the numbers of correct and incorrect behaviors by node i as a follower. μ and θ are the reward and penalty weights, respectively. λ is the leader's weight factor, set greater than 1 due to the higher impact of leader errors. From Eq. (4), it can be seen that $\Delta s_e(n_i) \in [-\theta, \mu]$. Further analysis of Eq. (1) and Eq. (4) is provided in Section IV-A.

Global node contribution values are updated as follows:

$$s_e(n_i) = \alpha \cdot s_{e-1}(n_i) + (1-\alpha) \cdot \Delta s_e(n_i), \tag{5}$$

$$p_e(n_i) = \alpha \cdot p_{e-1}(n_i) + (1-\alpha) \cdot \Delta p_e(n_i), \tag{6}$$

where $s_{e-1}(n_i)$ and $p_{e-1}(n_i)$ are the security and performance global contribution values of node i at the end of Epoch $e-1$. $\alpha \in [0,1]$ is the retention factor, generally set to be greater than 0.5. Note that nodes in the A-Shard only update security contribution values, as they do not participate in transaction processing.

C. The NACV Algorithm

The A-Shard employs the Node Allocation Algorithm based on Node Contribution Values (NACV) to allocate nodes to W-Shards. The algorithm proceeds as follows:

Step 1: Information Collection. Obtain the security contribution value $s(n)$, performance contribution value $p(n)$ of each node, and the current node-shard mapping M from the State Block. Calculate the security value $s(S)$, performance value $p(S)$, and estimated processing time $t(S)$ of each shard based on the workloads. Sort W-Shards twice by $s(S)$ and $t(S)$.

Step 2: Handling New Nodes. For new nodes without contribution values, assign the average security and performance contributions $\overline{p_{all}}$ and $\overline{s_{all}}$, respectively. For each node n not present in the previous epoch, perform the following: If $s(n) \geq \overline{s_{all}}$, add n to $\arg\min_S s(S)$. If $p(n) \geq \overline{p_{all}}$, add n to $\arg\max_S t(S)$. Otherwise, add n to $\arg\max_S s(S)$. Update $s(S)$, $p(S)$, $t(S)$, M and resort the shards.

Step 3: Security Adjustment. Set the initial and threshold values for the number of iterations, shard security variance and the output result of node-shard mapping: I, I_{thre}, $Var_r^s = \mathrm{Var}(s(S))$, Var_{thre}^s, $M_r = M$. Iterate the following steps until $Var_r^s \leq Var_{thre}^s$ or $I \geq I_{thre}$: (a) Sort shards by $s(S)$. (b) Randomly select n_i satisfying $s(n_i) \leq \overline{s_{all}}$ and $n_i \in \arg\min_S s(S)$. Randomly select n_j satisfying $s(n_j) \geq \overline{s_{all}}$ and $n_j \in \arg\max_S s(S)$. (c) Swap n_i and n_j, update $Var(s(S)), M, I$ and resort shards by $s(S)$. (d) If $Var_r^s < Var(s(S))$, update Var_r^s and M_r.

Step 4: Performance Adjustment. Set the initial and threshold values for shard processing time variance and reset the number of iterations: $Var_r^t = \mathrm{Var}(t(S))$, Var_{thre}^t, $I = 0$. Iterate the following steps until $Var_r^t \leq Var_{thre}^t$ or $I \geq I_{thre}$: (a) Calculate $Var(t(S))$. (b) Randomly select n_i satisfying $p(n_i) \geq \overline{p_{all}}$ and $n_i \in \arg\min_S t(S)$. Randomly select n_j satisfying $p(n_j) \leq \overline{p_{all}}$ and $n_j \in \arg\max_S t(S)$. (c) If not found, select n_i satisfying $\arg\max_{n_i \in \arg\min_S t(S)} p(n_i)$. Randomly select n_j satisfying $p(n_j) < p(n_i)$ and $n_j \in \arg\max_S t(S)$. (d) If not found, select n_i satisfying $\arg\max_{n_i \notin \arg\max_S t(S)} p(n_i)$.

Randomly select n_j satisfying $p(n_j) < p(n_i)$ and $n_j \in \arg\max_S t(S)$. (e) Swap n_i and n_j if suitable nodes are found. Otherwise, abandon node allocation and trigger account allocation. (f) Update $Var(t(S)), Var(s(S)), M_r, I$. If $Var(t(S)) < Var_r^t$ and $Var(s(S)) < Var_{thre}^s$, update Var_r^t, M_r.

The algorithm terminates with the output of the updated node-shard mapping M_r. The time complexity is $O(N' \cdot K + I_{\text{thre}} \cdot K \log K)$, where N' is the number of new nodes, I_{thre} is the iteration threshold, and K is the number of shards.

D. The P-Louvain Algorithm

We propose the P-Louvain algorithm for account allocation, which extends the community detection algorithm *Louvain* [29] by considering the performance of shards. The leader in A-Shard executes P-Louvain to minimize disparities in processing time across W-Shards. As shown in Algorithm 1, we define the transaction graph as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, the number of shards as K, performance values of shards as $\mathcal{P} = \{p_1, p_2, \ldots, p_K\}$ and the accounts lists of shards as $\mathcal{V} = \{V_1, \ldots, V_K\}$. p_K is estimated by summing the performance contribution values of nodes in shard K, as demonstrated in Section IV-A.

Initialization Phase: We employ *Louvain* on $\mathcal{G}$ to iteratively divide accounts into communities. Typically, the number of communities exceeds the number of shards.

Community Movement Phase: We sort communities by size and sort shards by $\mathcal{P}$. The first K communities are added to the corresponding shards (Lines 6-9). The remaining communities are processed successively in lines 10-14. Each community is assigned to the shard with the minimum processing time, and the processing time of shards $\mathcal{T}$ is then updated. Adding this phase can lead to faster convergence of the algorithm.

Node Movement Phase: We initialize the check flag of each account to true. Subsequently, lines 17-32 loop through the account v_i whose check flag is true. First we get the list of shards where the account's neighboring accounts on $\mathcal{G}$ are located. Then for each S in the list, we calculate the new processing time for the shards that v_i came from and went to if moving v_i to S. Then we calculate the reduction of the maximum value of the processing time for both shards, recording the maximum reduction R_{max} and the corresponding shard S_{max}. Subsequently, we move v_i to S_{max} and update $\mathcal{T}$, $\mathcal{V}$. The check flag of v_i and its neighbors are set to false and true respectively (Lines 28-31). Finally, when there are no accounts to check, P-Louvain outputs $\mathcal{V}$.

The time complexity of each phase is $O(m \log n)$, $O(l \log l + lK)$ and $O(mK)$. Among them, m is the number of edges, n is the number of accounts and l is the number of communities. The total time complexity is $O(m \log n + l \log l + lK + mK)$.

We develop a verification procedure for other nodes to verify the result $\mathcal{V}$. If any boundary account can be moved to reduce shard processing times, the verification fails. If no moves are possible, the verification succeeds.

Algorithm 1: The P-Louvain Algorithm.

Input: $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, K, $\mathcal{P} = \{p_1, p_2, \ldots, p_K\}$.
Output: $\mathcal{V} = \{V_1, \ldots, V_K\}$.

1 // 0. Initialization Phase.
2 $\{c_1, c_2, \ldots, c_K, \ldots, c_l\}$ = Louvain($\mathcal{G}$).
3 // 1. Community Movement Phase.
4 Sort communities by size $\rightarrow \{c_1, c_2, \ldots, c_K, \ldots, c_l\}$.
5 Sort shards by $\mathcal{P} \rightarrow \{S_1, S_2, \ldots, S_K\}$.
6 **for** $i = 1$ *to* K **do**
7 Move community c_i to shard S_i.
8 Update shard processing time $\mathcal{T} = \{t_1, t_2, \ldots, t_K\}$.
9 **end**
10 **for** $i = K + 1$ *to* l **do**
11 Find shard $S_{\text{minTime}}$with the minimum t.
12 Move community c_i to shard S_{minTime}.
13 Update $\mathcal{T}$.
14 **end**
15 // 2. Account Movement Phase.
16 needCheck$[v_i]$ = true, each $v_i \in \mathcal{V}$.
17 **while** $v_i \in \mathcal{V}$ *and needCheck*$[v_i] ==$ *true* **do**
18 Get neighbor's non-repetitive shard list $\mathcal{NS}_{v_i} = \{\mathcal{NS}_{v_i}^1, \mathcal{NS}_{v_i}^2, \ldots, \mathcal{NS}_{v_i}^j\}$.
19 $R_{\text{max}} \leftarrow 0$, $S_{\text{max}} \leftarrow$ the shard where v_i is located.
20 **for** $S \in \mathcal{NS}_{v_i}$ **do**
21 **if** $v_i \notin S$ **then**
22 Calculate two shards' new processing time $t'_{\text{from}}, t'_{\text{to}}$, if move v_i to S.
23 **if** $\max(t'_{from}, t'_{to}) - \max(t_{from}, t_{to}) > R_{max}$ **then**
24 Update $R_{\text{max}}, S_{\text{max}}$.
25 **end**
26 **end**
27 **end**
28 Move v_i to S_{max}.
29 Update $\mathcal{T}$, $\mathcal{V}$.
30 needCheck$[u]$ = true, each neighbor u of v_i.
31 needCheck$[v_i]$ = false.
32 **end**
33 **return** $\mathcal{V} = \{V_1, V_2, \ldots, V_K\}$.

E. System state update mechanism

This subsection describes the state update mechanism during epoch switches. The leader of each W-Shard calculates the stage contribution values for Epoch e using Eq. (1) and Eq. (4) and A-Shard packages them with $TX_{pending}$ into the Shard Summary Block. Two Modified Merkle Patricia Trees (MMPT) [30] are created, with their roots added to the header of the Shard Summary Block.

A-Shard generates the System Summary Block for Epoch e, storing the hash of the latest TX-blocks from W-Shards in the block header, as illustrated in Fig. 3. Additionally, A-Shard constructs the updated global contribution values into MMPT and adds the root of MMPT, the root of confirmed transactions $TX_{confirm}$, the root of $TX_{pending}$, and other fields into the header of the block. Other fields contain information such as block height. After consensus, the block is submitted to the S-Blockchain.

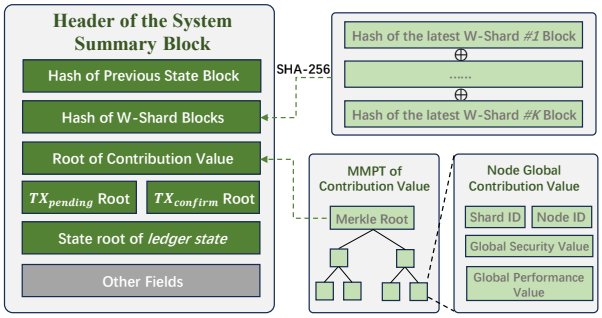


Fig. 3: Data Structure of the System Summary Block.

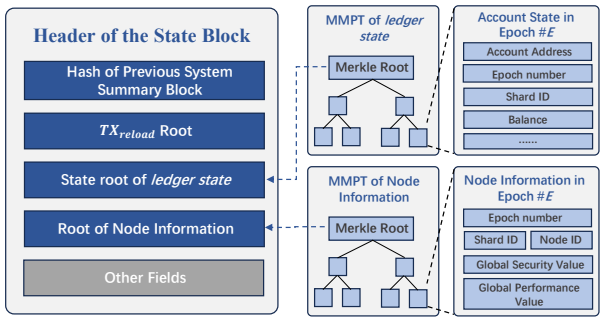


Fig. 4: Data Structure of the State Block.

Upon entering Epoch $e+1$, after node and account allocation, A-Shard constructs two MMPTs with the updated information of nodes and accounts. The roots of the two MMPTs are stored in the header of the State Block, as illustrated in Fig. 4. Depending on the result of P-Louvain, $TX_{pending}$ in Epoch e are organized as TX_{reload}, with their root included in the header of this block. After consensus, the State Block is submitted to S-Blockchain and broadcast. Nodes determine their respective W-Shards and TX_{reload} based on this block. Each W-Shard generates a Genesis Block to achieve consensus on the updated state, and begins processing transactions.

IV. Analysis

This section presents the rationale behind the node contribution values, shard security, and ledger security.

A. The Reasonableness of the Node Contribution Values

We analyze the reasonableness of the security and performance contribution values to ensure they accurately reflect the nodes' contribution.

Theorem 1. *The security contribution value of an honest node remains constant, while that of a malicious node decreases.*

Proof: Consider node i as a follower during Epoch e. Eq. (4) can be organized as:

$$\begin{aligned}\Delta s_e(n_i) &= \mu \cdot \frac{N_{FRi}}{N_{FRi}+N_{FWi}} - \theta \cdot \frac{N_{FWi}}{N_{FRi}+N_{FWi}} \\ &= \mu \cdot f_{Ri} - \theta \cdot f_{Wi},\end{aligned} \tag{7}$$

where f_{Ri} and f_{Wi} represent the ratios of accurate and inaccurate behaviors of node i during Epoch e, respectively. Consequently, $f_{Ri} + f_{Wi} = 1$. Given that $\theta > \mu$, an increase in f_{Wi} results in a decrease in $\Delta s_e(n_i)$, meaning $s_e(n_i) < s_{e-1}(n_i)$ for malicious nodes.

In PBFT, the leader moves to the next step after collecting commit messages from 2/3 of nodes. Therefore, even if all nodes are honest, there will still be 1/3 of nodes considered to be behaving incorrectly. In this case, $s_e(n_i) \in \left[\frac{2\mu-\theta}{3} - \epsilon, \frac{2\mu-\theta}{3} + \epsilon\right]$, where ϵ represents a small fluctuation. Thus, we set $\theta > \mu \geq \frac{\theta}{2}$ to ensure that the security contribution value of an honest node is greater than or equal to 0. And we can derive the condition for $\Delta s_e(n_i) < 0$ from Eq. (7):

$$\Delta s_e(n_i) < 0 \iff f_{Wi} > \frac{\mu}{\mu+\theta}. \tag{8}$$

Therefore, by changing μ and θ, the system's tolerance for malicious behaviors can be adjusted. If $\mu = \frac{\theta}{2}$, then $s_e(n_i) \in [-\epsilon, \epsilon]$, $\Delta s_e(n_i) < 0 \iff f_{Wi} > \frac{1}{3}$, which represents the strictest condition.

Theorem 2. *The sum of the stage performance contribution values of the nodes within a shard during an epoch equals the shard's TPS.*

Proof: Let N_{ALL} denote the number of blocks in shard S during Epoch e, with TX_j being the number of transactions in block j and t_e the duration of Epoch e. Combining Eq. (2), the TPS of shard S during Epoch e can be expressed as:

$$TPS_e^S = \frac{1}{t_e}\sum_{j=1}^{N_{\text{ALL}}}(TX_j - \delta_j \cdot TX_j). \tag{9}$$

Based on the reward rules (Section III-B), the transaction contribution $TX(n_i)$ for node i is calculated. If block j is successfully submitted, $TX(n_i)$ of each node that behaved correctly is $\frac{TX_j}{n_{R_j}}$, and there are no penalties. If the submission fails, $TX(n_i)$ of each node that behaved correctly is $\frac{TX_j}{n_{R_j}}$, while that of each node that behaved incorrectly is $\frac{-TX_j}{n-n_{R_j}}$.

Combining Eq. (2) and Eq. (3), we can calculate the stage contribution value $\Delta p_e(n_i)$ as follows:

$$\Delta p_e(n_i) = \frac{1}{t_e}\sum_{j=1}^{N_{\text{ALL}}}\left[\frac{TX_j}{n_{R_j}}e_{i,j} - \delta_j\frac{TX_j}{n-n_{R_j}}(1-e_{i,j})\right]. \tag{10}$$

The total number of nodes in S is represented as n. Then, the sum of all node contributions can be calculated as:

$$\begin{aligned}\sum_{i=0}^{n-1}\Delta p_e(n_i) &= \frac{1}{t_e}\sum_{j=1}^{N_{\text{ALL}}}\sum_{i=0}^{n-1}\left[\frac{TX_j}{n_{Rj}}e_{ij} - \frac{\delta_j \cdot TX_j}{n-n_{Rj}}(1-e_{ij})\right] \\ &= \frac{1}{t_e}\sum_{j=1}^{N_{\text{ALL}}}\left[\frac{TX_j}{n_{Rj}}n_{Rj} - \frac{\delta_j \cdot TX_j}{n-n_{Rj}}(n-n_{Rj})\right] \\ &= \frac{1}{t_e}\sum_{j=1}^{N_{\text{ALL}}}(TX_j - \delta_j \cdot TX_j).\end{aligned} \tag{11}$$

Combining Eq. (9), *Theorem 2* concludes.

B. Shard Security

The A-Shard node selection method follows the random selection approach from [1], [4]. Given that PBFT requires the proportion of malicious nodes to be less than 1/3, the failure probability of A-Shard can be derived from the cumulative hypergeometric distribution [21], as expressed below:

$$Pr[X \geq \lceil n_A/3 \rceil] = \sum_{x=\lceil n_A/3 \rceil}^{n_A} \frac{\binom{\rho n}{x}\binom{n(1-\rho)}{n_A-x}}{\binom{n}{n_A}}. \quad (12)$$

Among them, n is the total number of nodes, n_A is the number of nodes in A-Shard, X is the number of malicious nodes in A-Shard and ρ is the proportion of malicious nodes in the system. Next, we analyze the security of W-Shard.

Theorem 3. *The NACV algorithm is unpredictable and verifiable.*

Proof: The random selection operation in NACV can be described as choosing a node that meets a specific condition from the set $Set_w^{special}$ within a W-shard. The selection process is as follows: First, the randomness from the current epoch serves as the seed for a pseudo-random number generator to generate $Output \in \{1, 2, \ldots, |Set_w^{special}|\}$. The nodes in $Set_w^{special}$ are then sorted in ascending order by their addresses, and the node corresponding to $Output$ is selected. Since the same seed results in the same $Output$, all S-Shard nodes will select the same node, ensuring that NACV is deterministic and verifiable. Furthermore, the use of randomness guarantees the unpredictability of the selection outcome.

Theorem 4. *If A-Shard is secure, W-Shards will be secure.*

Proof: According to ***Theorem 1***, a decrease in a node's security value indicates an increase in its malicious behavior, which may result from degraded network conditions, reduced computational capacity, or adversarial attacks. If A-Shard is secure, the NACV algorithm will distribute nodes with lower security contributions across W-Shards to minimize the variance in shard security. This ensures that the security of W-Shards is maintained.

C. Node Contribution and Ledger Security

Malicious nodes may attempt to manipulate node contribution values and account balances. In ContribChain, both stage and global contribution values are represented as MMPTs, with their roots placed at the block headers (Section III-E). These blocks are propagated throughout the system for consensus, allowing nodes to verify the integrity of contribution values using the Merkle path [30]. Similarly, the correctness of account balances in the state blocks can be validated in the same manner [8].

V. EVALUATION

A. Experiment Settings

We implemented P-Louvain and ContribChain on an open-sourced blockchain testbed *BlockEmulator* [16].

Dataset: We used historical Ethereum transactions [31] from block height 10,000,000 to 10,999,999 (from May 4, 2020, to October 6, 2020). The maximum number of transactions used reached 30 million.

Transaction Processing: Each block is set to contain a maximum of 2000 TXs. The block interval is set to 5 seconds. Transactions are sent to the system at a specific injection rate.

Baselines: P-Louvain was compared against TxAllo [9] and CLPA [8]. TxAllo is another account allocation algorithm based on *Louvain*. In the system experiments, we set ContribChain-1 to use NACV and P-Louvain, and ContribChain-2 to use only P-Louvain. Then we compared them with CLPA and Monoxide [5], where Monoxide allocates accounts based on the first few digits of the account address.

Other Settings: The parameters μ, θ, λ, α, and β were set to 0.9, 1.5, 2, 0.7, and 2, respectively.

TABLE I: Node Safety and Delay Settings. Num_{n_w} denotes the number of potential malicious nodes per shard, P_{n_w} represents the probability of malicious behaviors, and (L_s, L_n) indicates the initial latency for shards and nodes (ms).

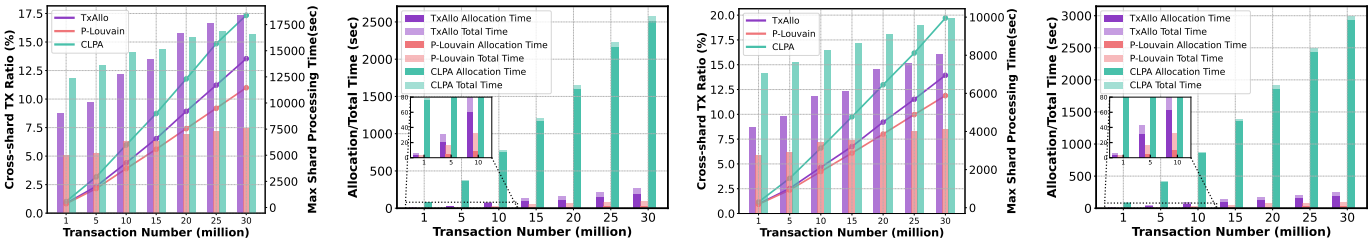
Setting	Num_{n_w}	P_{n_w}	L_s, L_n
1	1	5%	0, 0
2	1	5%	10, 5
3	{0, 1, 2}	5%	50, 5
4	{0, 1, 2}	5%	200, 10
5	{0, 1, 2}	[20%, 54%]	200, 10
6	{0, 1, 2}	100%	300, 10
7	{0, 2, 2}	100%	300, 10

B. Performance of the Account Allocation Algorithm

After adjusting the shards to different TPS values, we measured the algorithm's time consumption and allocation performance. Fig. 5a and 5c show the maximum shard processing time and the cross-shard transaction ratio. P-Louvain consistently outperforms the others, with its advantage growing as N_{TX} increases. For N_{TX} = 30 million and K = 8, P-Louvain reduces the cross-shard transaction ratio by 7.5 % compared to TxAllo. Fig. 5b and 5d depict the time consumption of each algorithm and their allocation components. CLPA shows significant growth in execution time, while TxAllo and P-Louvain remain relatively stable. For N_{TX} = 30 million and K = 8, P-Louvain reduces the allocation execution time by 86% compared to TxAllo. Thus, P-Louvain is both more time-efficient and more effective.

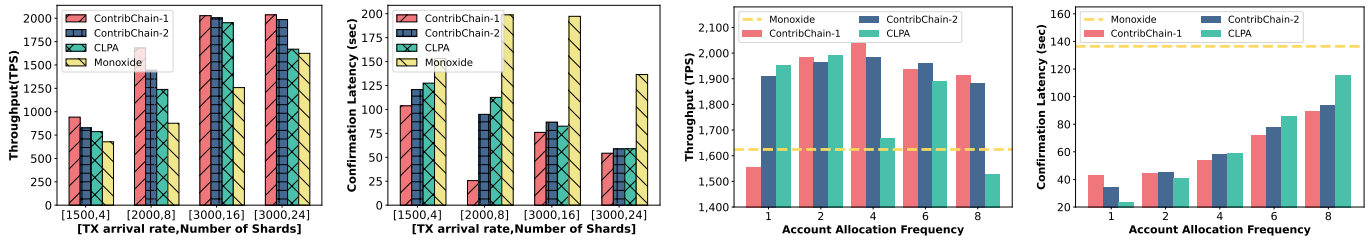
C. System Throughput and TX Confirmation Latency

We first set the account allocation frequency to every four epochs (f=4) and the node allocation period to 80 seconds (T_{NA}=80s). As shown in Fig. 6a and 6b, ContribChain-1 outperforms CLPA and Monoxide in terms of TPS and TX confirmation latency as both the TX arrival rate and K increase. At a rate of 2000 tx/s and K=8, ContribChain-1 achieves a 92% and 35.8% improvement in TPS over Monoxide and CLPA, respectively. When the TX arrival rate reaches 3000 tx/s and K increases from 16 to 24, the TPS of CLPA drops significantly, while ContribChain-1 exhibits minimal variation, demonstrating its superior scalability. Comparing ContribChain-1 and ContribChain-2, the former shows



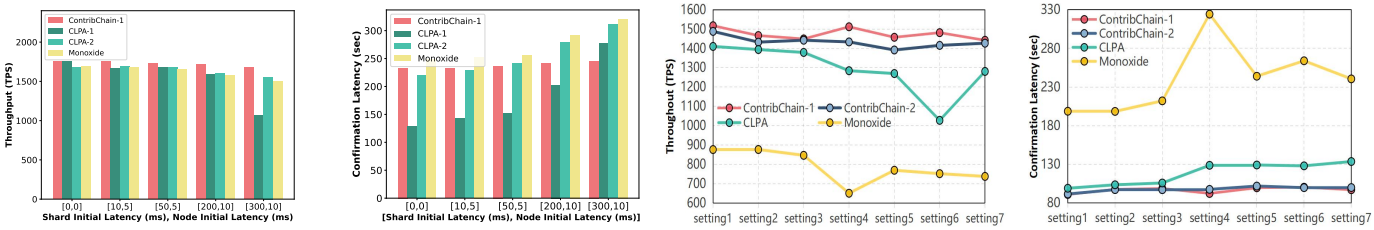
(a) Effect of allocation, K=4 (b) Time consumption, K=4 (c) Effect of allocation, K=8 (d) Time consumption, K=8

Fig. 5: Performance of account allocation algorithms with varying N_{TX}. In (a) and (b), assume the TPS of shards are {1000, 800, 800, 600}. In (c) and (d), assume the TPS of shards are {1000, 900, 900, 800, 800, 700, 700, 600}.



(a) TPS (b) Latency (c) TPS, $T_{NA} = 80$s (d) Latency, $T_{NA} = 80$s

Fig. 6: TPS and transaction confirmation latency changes with TX arrival rate, N_{TX} and account allocation frequency (f).



(a) TPS (b) Latency (c) TPS (d) Latency

Fig. 7: TPS and transaction confirmation latency changes with node delay settings or both node security and delay settings.

better performance, highlighting the synergistic enhancement provided by P-Louvain and NACV cooperatively.

Next, we adjust f while T_{NA}=80s. As seen in Fig. 6c and 6d, Monoxide, without account allocation, serves as the yellow baseline. Overall, the TPS increases as f rises from 1 to 4. However, as f increases beyond 4, the TPS begins to decrease due to more frequent account allocation, which negatively impacts consensus time, while a larger f increases the cross-shard transaction ratio. Consequently, we set the default value of f to 4.

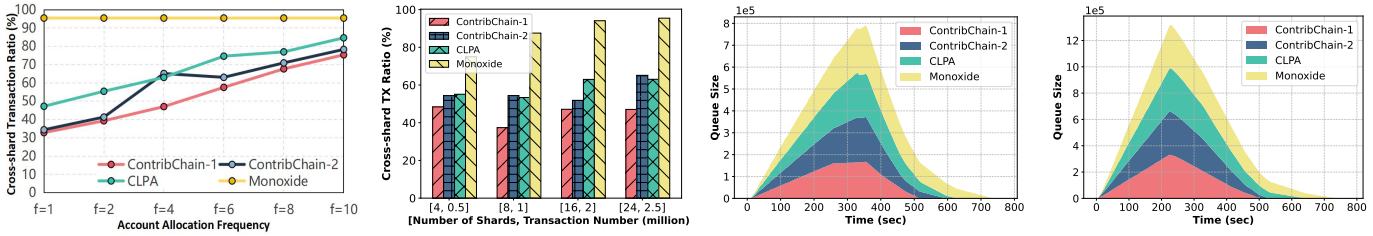
We also evaluated the system performance under different node delays, running 8 epochs of 200 seconds each, with a TX arrival rate of 2500 tx/s. For CLPA-1, f is set to 1, while for CLPA-2 and ContribChain-1, f is set to 4. As shown in Fig. 7a and 7b, ContribChain-1 exhibits minimal fluctuation in TPS and TX confirmation latency as delays increase. Despite having higher TX confirmation latency than CLPA-1 at low delays, ContribChain-1 maintains the highest TPS, benefiting from fewer account migrations and reduced overhead.

Finally, we adjust both the node security and delay settings (Table I). The delay of node i in shard j is $L_s * j + L_n * i$. With N_{TX} = 1 million, a TX arrival rate of 2000 tx/s, and T_{NA} = 80 seconds, Fig. 7c and 7d show that, compared to CLPA and Monoxide, ContribChain-1 and ContribChain-2 maintain more stable TPS and TX confirmation latency, performing better even as node security and performance degrade.

D. Cross-shard TX Ratio and TX Pool Queue Size

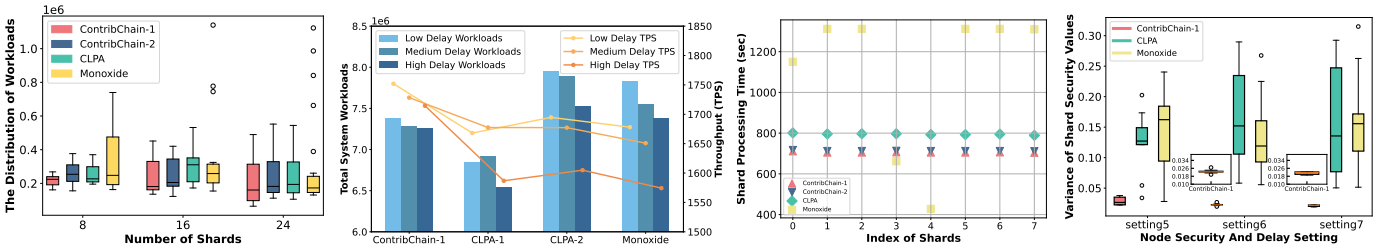
Cross-Shard TX Ratio: We evaluate the impact of f, K, and N_{TX} on the cross-shard TX ratio. As shown in Fig. 8a and 8b, ContribChain-1 consistently exhibits the lowest cross-shard TX ratio. At f=6, ContribChain-1 reduces the cross-shard TX ratio by 16% compared to CLPA. Moreover, as K and N_{TX} increase, the cross-shard TX ratio in ContribChain-1 remains nearly constant, indicating strong scalability.

TX Pool Queue Size: We also analyze the TX pool queue size for N_{TX} = 500,000 and K=4. Transactions are injected into the TX pool at rates of 1500 tx/s and 2500 tx/s until



(a) Cross-shard TX Ratio (b) Cross-shard TX Ratio (c) TX arrival rate=1500 TXs/Sec (d) TX arrival rate=2500 TXs/Sec

Fig. 8: Cross-shard Ratio and queue size of the TX pool. In (a), T_{NA}=80 seconds. In (b), N_{TX} = 500,000 and K=4.



(a) Distribution of workloads (b) Total system workloads (c) Shard processing time (d) Difference of shard security

Fig. 9: (a) and (b) show the impact of K and delay settings on workload distribution and total system workload. (c) shows the processing time of shards. (d) shows the variance in shard security values under three node security and delay settings.

all transactions are processed. Fig. 8c and 8d show that once the TXs are fully injected, the queue size decreases. Among all methods, ContribChain-1 has the shortest queue, with the lowest peak and the fastest time to empty the queue, demonstrating superior efficiency in handling transaction backlogs.

E. Balance among Shards

Workload Balance: We assess the workload distribution among shards for K=8, 16, and 24. Fig. 9a reveals that Monoxide suffers from poor workload distribution, exhibiting many outliers. In contrast, ContribChain-1 shows a more uniform load distribution with no outliers, indicating better workload balance. We then evaluate total workload and TPS after running 8 epochs with different node delay settings (Settings 2, 3, and 4 in Table I). The parameters are 8 shards, 2500 tx/s, and 200 seconds per epoch. Fig. 9b shows that CLPA-1 achieves lower TPS and higher workload due to frequent account allocations, which reduce consensus time. Under the same f, ContribChain-1 achieves lower workload than CLPA-2, owing to fewer cross-shard transactions. As node delays increase, ContribChain-1 demonstrates superior self-regulation compared to other algorithms.

Stress and Security Balance: With N_{TX} fixed, we analyze the processing time of all shards (Fig. 9c). Through inter-shard comparison, we can observe a significant disparity in processing times in Monoxide. The transaction backlog problem depicted in Fig. 1 occurs in shards other than shard 3, 4. While ContribChain-1 have more uniform and shorter processing time, showing better stress balancing capabilities. Fig. 9d shows the distribution of the variance of shard security values across different epochs. The shard security value is calculated from the average of the security contribution values of the nodes within the shard. We study this distribution in three node security and delay settings (Setting 5, 6, 7 in Table I). ContribChain-1 consistently maintains low and stable variance after each node allocation, indicating excellent security balancing capabilities. In contrast, CLPA and Monoxide exhibit larger and more fluctuating variances.

VI. Conclusion

ContribChain introduces a stress-balanced sharding protocol for blockchain systems that dynamically evaluates node performance and security through the update of node contribution values. Additionally, we propose novel account and node allocation algorithms to achieve optimal stress balance across the network. Experimental results demonstrate that ContribChain outperforms existing protocols in key metrics, including transaction throughput, cross-shard transaction ratio, confirmation latency, TX pool queue size, and stress balance. In future work, we aim to further enhance the system's adaptability by incorporating dynamic adjustments to the number of shards and the frequency of account allocation.

Acknowledgment

This work was partially supported by the National Key R&D Program of China (2021YFB2700300), the National Natural Science Foundation of China (62141605, 62372493), the Beijing Natural Science Foundation (Z230001), the Beijing Advanced Innovation Center for Future Blockchain and Privacy Computing (GJJ-23-001, GJJ-23-002), the China Postdoctoral Fellowship Fund 373500, the Beihang Dare to Take Action Plan KG16336101, and the science and technology project of State Grid Corporation of China (5400-202255416A-2-0-ZN).

REFERENCES

[1] 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.

[2] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 17–30.

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

[4] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 931–948.

[5] 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.

[6] 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*. IEEE, 2022, pp. 1968–1977.

[7] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi, "Towards scaling blockchain systems via sharding," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 123–140.

[8] 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," in *2022 41st International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2022, pp. 284–294.

[9] 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.

[10] 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*, vol. 34, no. 10, pp. 2797–2810, 2023.

[11] H. Huang, Y. Lin, and Z. Zheng, "Account migration across blockchain shards using fine-tuned lock mechanism," pp. 271–280, 2024.

[12] 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*, 2024.

[13] J. Yun, Y. Goh, and J.-M. Chung, "Dqn-based optimization framework for secure sharded blockchain systems," *IEEE Internet of Things Journal*, vol. 8, no. 2, pp. 708–722, 2020.

[14] S. Yuan, J. Li, J. Liang, Y. Zhu, X. Yu, J. Chen, and C. Wu, "Sharding for blockchain based mobile edge computing system: A deep reinforcement learning approach," in *2021 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2021, pp. 1–6.

[15] Z. Yang, R. Yang, F. R. Yu, M. Li, Y. Zhang, and Y. Teng, "Sharded blockchain for collaborative computing in the internet of things: Combined of dynamic clustering and deep reinforcement learning approach," *IEEE Internet of Things Journal*, vol. 9, no. 17, pp. 16 494–16 509, 2022.

[16] H. Huang, G. Ye, Q. Chen, Z. Yin, X. Luo, J. Lin, Q. Yang, and Z. Zheng, "Blockemulator: An emulator enabling to test blockchain sharding protocols," *arXiv preprint arXiv:2311.03612*, 2023.

[17] H. Huang, W. Kong, S. Zhou, Z. Zheng, and S. Guo, "A survey of state-of-the-art on blockchains: Theories, modelings, and tools," *ACM Computing Surveys (CSUR)*, vol. 54, no. 2, pp. 1–42, 2021.

[18] B. Awerbuch and C. Scheideler, "Towards a scalable and robust dht," in *Proceedings of 18th Annual ACM Symposium on Parallelism in Algorithms and Architectures*, 2006, pp. 318–327.

[19] 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.

[20] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *OsDI*, vol. 99, no. 1999, 1999, pp. 173–186.

[21] Y. Liu, J. Liu, M. A. V. Salles, Z. Zhang, T. Li, B. Hu, F. Henglein, and R. Lu, "Building blocks of sharding blockchain systems: Concepts, approaches, and open problems," *Computer Science Review*, vol. 46, p. 100513, 2022.

[22] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and A. Spiegelman, "Solida: A blockchain protocol based on reconfigurable byzantine consensus," *arXiv preprint arXiv:1612.02916*, 2016.

[23] 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 *25th USENIX Security Symposium (USENIX Security '16)*, 2016, pp. 279–296.

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

[25] 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.

[26] E. Fynn and F. Pedone, "Challenges and pitfalls of partitioning blockchains," in *2018 48th Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W)*. IEEE, 2018, pp. 128–133.

[27] P. Li, M. Song, M. Xing, Z. Xiao, Q. Ding, S. Guan, and J. Long, "Spring: Improving the throughput of sharding blockchain via deep reinforcement learning based state placement," in *Proceedings of the ACM on Web Conference 2024*, 2024, pp. 2836–2846.

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

[29] 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.

[30] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, vol. 3, no. 37, pp. 2–1, 2014.

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