A Survey on Cross-Shard Transaction Allocation Optimization and Processing Mechanisms

Zitong Zhang, Haoxiang Han, Zheng Yan, *Fellow, IEEE,* Erol Gelenbe, *Life Fellow, IEEE*

***Abstract*—Sharding is one of the most effective technologies for improving blockchain scalability by dividing a blockchain network into multiple shards that process transactions in parallel. However, many existing sharded blockchain systems can include a large number of cross-shard transactions, resulting in high communication overhead and processing latency. Two solutions have emerged to address these issues. One of them focuses on optimizing the allocation of transactions to reduce the cross-shard communication while ensuring workload balance. The other emphasizes the design of an efficient cross-shard transaction processing mechanism to reduce the latency caused by confirming cross-shard transactions. However, the literature still lacks a comprehensive review on the advances of Cross-shard Transaction Allocation Optimization and Processing (CTAOP) mechanisms, which would be a valuable addition to the literature. This paper is a comprehensive survey of existing CTAOP mechanisms, and first introduces cross-shard transactions and related concepts. Then, we propose two sets of criteria for evaluating the two types of CTAOP mechanisms, and provide a taxonomy of CTAOP mechanisms, followed by a thorough review based on our proposed criteria. A list of open issues is also highlighted, and corresponding future research directions are suggested to advance our understanding of CTAOP mechanisms.**

***Index Terms*—Blockchain sharding, Transaction allocation optimization, Cross-shard transaction processing.**

I. Introduction

BLOCKCHAIN, a pivotal technology for distributed ledgers, has been extensively applied in various domains, such as finance, communications and the Internet of Things (IoT) [1]. However, existing blockchain consensus protocols typically mandate each transaction to be broadcast across the network of the blockchain, with each network node autonomously verifying its validity. A transaction is confirmed only when a majority of nodes reach consensus, as illustrated in Fig. 1(a). Thus, transaction processing delays can increase significantly as the number of nodes and transactions increase. Furthermore, each node needs to communicate with all others and jointly participate in transaction verification, leading to an exponential rise in communication cost, a reduction in throughput, and increased confirmation latency. As a result, current blockchain systems, such as Bitcoin [2], are limited in practice to a maximum processing capacity of seven Transactions Per Second (TPS) [3], and the throughput of blockchain is several orders of magnitude smaller than common centralized electronic credit card-based payment systems such as Visa [4].

Sharding is one of the most promising solutions for improving blockchain scalability [5], and it is widely used for enhancing horizontal scalability in traditional database systems. First applied to blockchain by Luu *et al.* [6], sharding partitions the blockchain network nodes into distinct clusters or "shards" which work in parallel to reach a consensus regarding transactions and to generate a new block efficiently. Normally, shards prevent centralization by operating with independent nodes, thereby avoiding resource bottlenecks and reducing the risk of shard collusion. As illustrated in Fig. 1(b), blockchain sharding can be categorized into three types, i.e., network sharding, transaction sharding, and state sharding. Network sharding divides nodes of the blockchain network into different shards, each of which can process transactions in parallel. Transaction sharding assigns a set of transactions to distinct shards, to enable parallel transaction processing and allow shards to collaborate on cross-shard transactions. State sharding partitions the global state of the blockchain into distinct shards to enhance scalability, where each shard maintains only a portion of the entire ledger. Thus, sharding schemes must improve parallel processing capabilities without compromising security [7], and cross-shard transactions must be optimized and processed efficiently.

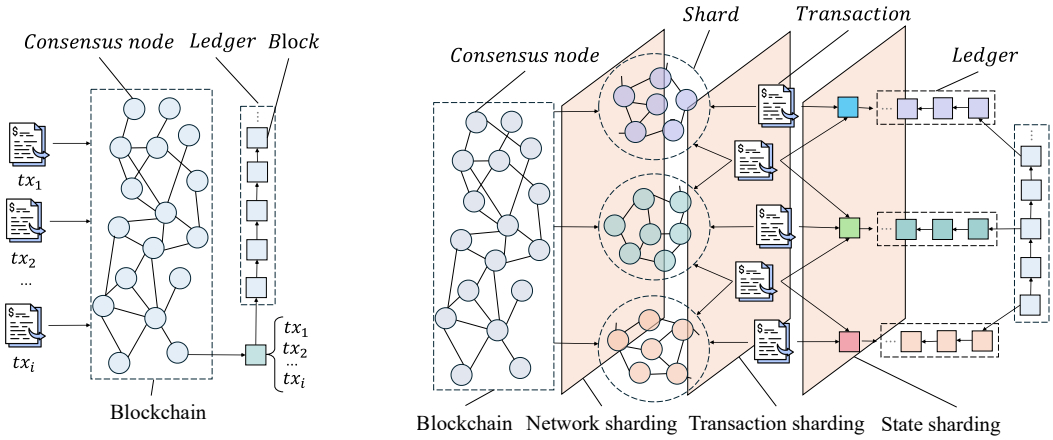
Although sharding can significantly improve blockchain performance, its design faces two key challenges. Since each shard contains a relatively small number of nodes, attackers can launch Sybil attacks [8] to forge legal identities, increase the number of malicious nodes within a shard, and launch a shard takeover attack [9]. To address these security risks, researchers have proposed a permissionless mechanism such as Proof-of-Work (PoW) [2] or Proof-of-Stake (PoS) [10] to periodically select a statistically representative group of validators. Second, it has been suggested that shards are periodically reconfigured to ensure that the probability of any shard being compromised over the system's lifetime is minimized. Thus OmniLedger [11] leverages random numbers generated by RandHound to determine node allocation in each epoch, while SGX-Shard [12] employs secure and unbiased random numbers produced within a Trusted Execution Environment (TEE) [13] to allocate nodes.

In a sharded blockchain system, transaction allocation and

Zitong Zhang and Haoxiang Han are with the State Key Lab of ISN, School of Cyber Engineering, Xidian University, Xi'an, Shaanxi, 710026 China. (email: zhangzt@stu.xidian.edu.cn, haoxianghan0228@gmail.com), ORCID: 0009-0004-5598-8324, 0009-0000-4228-9402

Zheng Yan (corresponding author) is with the State Key Lab of ISN, School of Cyber Engineering, Hangzhou Institute of Technology, Xidian University, Xi'an, Shaanxi, 710026 China. (email: zyan@xidian.edu.cn), ORCID: 0000-0002-9697-2108.

Erol Gelenbe is with the Institute of Theoretical and Applied Informatics, Polish Academy of Sciences, Gliwice, Poland. He is also a Visiting Professor at King's College London, and an Honorary Professor with Lab. I3S, Université Côte d'Azur, Nice, France (email: seg@iitis.pl), ORCID: 0000-0001-9688-2201.



(a) Non-sharded blockchain. (b) Sharded blockchain.

Fig. 1. Non-sharded blockchain versus sharded blockchain. Instead of processing all tx by all blockchain nodes in a conventional manner, a sharded blockchain partitions nodes into a number of smaller committees, and each committee processes only a subset of tx and stores a portion of the ledger.

cross-shard transaction processing should be well addressed. Each shard operates relatively independently, with its own independent verifiers, transaction set, and state, lacking the capability to directly access or modify the states of other shards. Therefore, the shard intending to access and modify the state of another shard must initiate cross-shard transactions. Since shard ledgers operate in isolation, validating cross-shard transactions necessitates an additional consensus round, and distinct shards must achieve ledger synchronization prior to submitting transactions. This imposes transaction confirmation latency and significant communication overhead. Furthermore, load imbalance degrades system performance because transactions blocked in hot shards with heavy transaction allocation experience high confirmation latency.

Many existing transaction allocation and processing solutions make simple decisions that randomly allocate transactions to different shards according to their addresses, inevitably resulting in an excessive number of cross-shard transactions and an uneven shard workload. In the case of the Unspent Transaction Output (UTXO) model, Rapidchain [14] indicates that, when the number of shards is 16, approximately 99.98% of the transactions involve cross-shard consensus. For the Account/Balance transaction model, when feeding 8×10^4 transactions, the Monoxide [15] yields unbalanced transaction distributions among all shards, and the proportion of cross-shard transactions can reach 90% when the number of shards exceeds 64 [16], [17]. Excessive cross-shard communication overhead and load imbalance emerge as key bottlenecks restricting system scalability.

To address the challenges posed by cross-shard transactions, particularly their significant communication overhead, researchers have proposed solutions in both the transaction sharding and the state sharding layers. One type of solution focuses on optimizing transaction allocation to minimize cross-shard communication. For example, Tao *et al.* [18] aggregate the accounts with similar transaction characteristics into the same shard, while Nguyen *et al.* [19] employs a graph-based approach to allocate transactions. Another type of solution emphasizes designing efficient cross-shard transaction processing mechanisms to improve concurrency and communication efficiency. For instance, Liu *et al.* [20], [21] introduced batch processing and pipelining techniques to enhance the efficiency of cross-shard transaction processing based on the Two-phase Commit (2PC) protocol [22]. In this paper, we refer to these two types of solutions as Cross-shard Transaction Allocation Optimization and Processing (CTAOP) mechanisms. The core objective of CTAOP is to optimize transaction allocation and processing workflows to significantly reduce cross-shard communication and processing overhead.

We can find a number of surveys about blockchain sharding in the literature from different perspectives. Wang *et al.* [23] identified five fundamental components of sharding schemes and analyzed the major challenges associated with each. Yu *et al.* [24] conducted a comprehensive comparison and quantitative evaluation on major sharding mechanisms, evaluating essential characteristics, including intra-consensus security and cross-shard transaction atomicity. Hafid *et al.* [25] proposed a taxonomy based on shard formation and intra-shard consensus, and compared typical existing protocols. Han et al. [26] deconstructed the blockchain sharding protocol into four foundational layers with distinct functionalities and offered insights into the structural coherence of system configurations. Huang *et al.* [27] summarized the key theories and methods of sharding techniques, including cross-shard consensus protocols, and outlined advantages and drawbacks of various sharding schemes. Liu *et al.* [28] decomposed existing sharded blockchain systems into functional components and classified them in terms of system model. Liu *et al.* [29] developed a classification framework for blockchain sharding schemes based on blockchain type and sharding technology, alongside a set of evaluation criteria. Li *et al.* [30] outlined the fundamental building blocks and summarized countermeasures to mitigate potential attacks in blockchain sharding. Zhang *et al.* [31] analyzed the characteristics of classical sharding techniques from both performance and implementation perspectives, and summarized the key mechanisms of sharding. Yang *et al.* [32] discussed the performance of sharding technologies and

TABLE I
COMPARISON OF OUR SURVEY WITH EXISTING RELATED SURVEYS

Paper	Year	Covered topics	①	②	③	④	⑤	⑥
Wang *et al.* [23]	2019	Analyze the key components and its major challenges.	○	○	○	●	○	○
Yu *et al.* [24]	2020	Focus on the intra-consensus protocol and atomicity of cross-shard.	○	○	○	●	○	○
Hafid *et al.* [25]	2020	Focus on the shard formation and the intra-shard consensus.	○	○	○	○	○	○
Han *et al.* [26]	2021	Focus on the foundational layers and the coherence of system setting.	○	○	○	●	○	◐
Huang *et al.* [27]	2022	Summarize the key theories and methods of sharding techniques.	○	○	○	●	◐	○
Liu *et al.* [28]	2022	Analyze the functional components in sharding schemes.	○	○	○	●	◐	○
Liu *et al.* [29]	2023	Classify the sharding schemes based on blockchain sharding type.	○	○	○	○	○	○
Li *et al.* [30]	2023	Focus on the sharding design models and key components.	○	○	○	●	○	○
Zhang *et al.* [31]	2023	Analyze the sharding techniques from performance and implementation perspectives.	○	○	○	●	○	○
Yang *et al.* [32]	2024	Focus on the discussion on the performance of sharding technologies.	○	○	◐	◐	○	◐
This paper	2025	Focus on the cross-shard transaction allocation optimization and processing.	●	●	●	●	●	●

●: Fully supported; ◐: Partially supported; ○: Not supported; ① Propose a clear technical framework for enhancing shard performance through the holistic optimization of cross-shard transaction performance and reliability. ② Give a detailed and differentiated review on the cross-shard transactions based on the UTXO model and the Account/Balance model. ③ Give a review on the transaction allocation optimization schemes. ④ Give a review on the cross-shard transaction processing mechanisms. ⑤ Present a taxonomy of the cross-shard transaction optimization and processing mechanisms. ⑥ Propose two sets of criteria that transaction allocation optimization schemes and cross-shard transaction processing mechanisms should satisfy, respectively.

proposed future directions accordingly.

Up to now, most surveys related to blockchain sharding focus on node allocation and shard formation technologies, while paying little attention to the technologies for transaction allocation and processing. Existing related surveys fail to present a clear technical framework for enhancing shard performance through holistic optimization of cross-shard transaction performance and reliability. Specifically, these surveys lack a detailed and differentiated review of cross-shard transactions based on the UTXO model and the Account/Balance model. Existing surveys [23], [24], [26]–[28], [30]–[32] only explore the cross-shard transaction processing mechanisms, lacking review and analysis on transaction allocation optimization schemes. Furthermore, although some surveys [26]–[28] classify cross-shard transaction processing mechanisms and evaluate their performance and security, they are not comprehensive, without classification or evaluation of transaction allocation optimization schemes. Thus, we are motivated to provide a thorough survey on the recent advances of CTAOP mechanisms in order to guide its future investigation. Table I provides a comparison between our survey and other highly related surveys. Our work is the first to establish a comprehensive technical framework for reviewing sharding performance from the aspects of CTAOP. Additionally, it is the first to offer a detailed and differentiated analysis on cross-shard transactions based on the UTXO and Account/Balance models, as well as the first to review transaction allocation optimization schemes.

In this paper, we perform a systematic survey on the CTAOP mechanisms. We first provide a detailed and differentiated overview on cross-shard transactions based on the above two transaction models and introduce theoretical framework related to cross-shard transactions. Then, we propose two sets of criteria that transaction allocation optimization mechanisms and cross-shard transaction processing mechanisms should satisfy, respectively. After that, we provide a taxonomy of the CTAOP mechanisms and outline their advantages and disadvantages. Based on the aforementioned taxonomy and the proposed evaluation criteria, we comprehensively review the CTAOP mechanisms in the literature and analyze their strengths and weaknesses. Finally, we highlight a number of unsolved issues and suggest future research directions. Specifically, the main contributions of this paper can be summarized as follows:

- We present a clear technical framework for enhancing shard performance through the holistic optimization of cross-shard transaction performance and reliability.
- We propose two sets of criteria that should be satisfied by two types of CTAOP mechanisms, respectively.
- We present a taxonomy of CTAOP mechanisms, and conduct an in-depth review on existing works by employing the proposed criteria to analyze their strengths and weaknesses.
- Based on our review and evaluation, we identify a list of open issues and further propose future research directions to promote future research on CTAOP.

The remainder of this survey is organized as follows. Section II gives a detailed and differentiated overview of cross-shard transactions based on the UTXO model and the Account/Balance model, and introduces important concepts related to cross-shard transactions. In Section III, we propose two sets of criteria for evaluating transaction allocation optimization schemes and cross-shard transaction processing mechanisms, respectively. Section IV presents a taxonomy of CTAOP mechanisms, followed by a comprehensive review on the literature of CTAOP by employing the proposed evaluation criteria. Based on the literature review, we identify open issues and suggest future research directions in Section VI. Finally, we draw a conclusion in the last section.

TABLE II
LIST OF ACRONYMS

Acronym	Full Form
CTAOP	Cross-Shard Transaction Allocation Optimization and Processing
IoT	Internet of Things
TPS	Transactions Per Second
PoW	Proof of Work
PoS	Proof of Stake
TEE	Trusted Execution Environment
UTXO	Unspent Transaction Output
2PC	Two-Phase Commit
MIMO	Multiple Input and Multiple Output
LPA	Label Propagation Algorithm
BFT	Byzantine Fault Tolerance
ACID	Atomicity, Consistency, Isolation, and Durability
CAP	Consistency, Availability, and Partition tolerance
PCC	Pessimistic Concurrency Control
OCC	Optimistic Concurrency Control
2PL	Two-Phase Locking
SISO	Single Input Single Output
DAG	Directed Acyclic Graph
BFS	Breadth-First Search
CoSi	Collective Signatures
PBFT	Practical Byzantine Fault Tolerance
LSTM	Long Short-Term Memory
A-Msign	Aggregated Multi-signature
MPT	Merkle Patricia Trie
DCC	Deterministic Concurrency Control
DeFi	Decentralized Finance

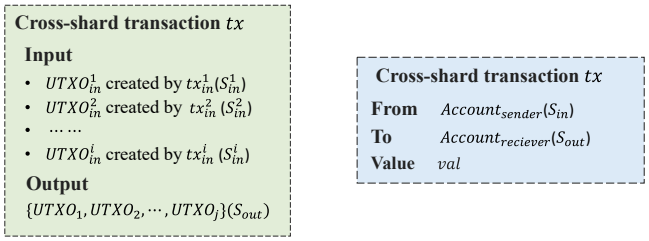
II. BACKGROUND KNOWLEDGE

This section provides useful background knowledge about blockchain sharding. We first present an overview of cross-shard transaction processing based on the two primary data models. Then, we introduce transaction graphs and summarize the relevant graph analysis methods. Finally, we present some key concepts related to cross-shard transaction processing, including transaction consistency and concurrency.

A. Cross-shard Transactions

Cross-shard transactions are the transactions that occur between different shards. Their consensus requires a collaboration of the shards that are involved in the transaction. Blockchain systems use two main data models, each designed for specific purposes and functionalities, which in turn determine the types of transactions that can be carried out in that blocks and the operations that are executed. One is the UTXO model, designed for Bitcoin, and the Account/Balance model adopted by Ethereum [33]. Here, we will now outline their structure.

UTXO model [2]: A UTXO transaction is a "fragments of assets" representation, containing a designated amount for each asset, and the corresponding ownership details. Assets are stored in UTXOs, and a transaction consumes one or more UTXOs as its inputs and generates new UTXOs as outputs. A consumed (or spent) UTXO is removed from the ledger and ceases to exist.



(a) A UTXO-based cross-shard transaction.

(b) An Account/Balance-based cross-shard transaction.

Fig. 2. Cross-shard transactions based on two transaction models.

Account/Balance model [33]: This model is widely employed for user accounts and smart contracts. Each user and contract is assigned a unique, fixed address. A user's account is associated with a non-negative balance. A contract address records the binary codes and the state data of the contract [34].

UTXO-based and Account/Balance-based cross-shard transactions have different structures:

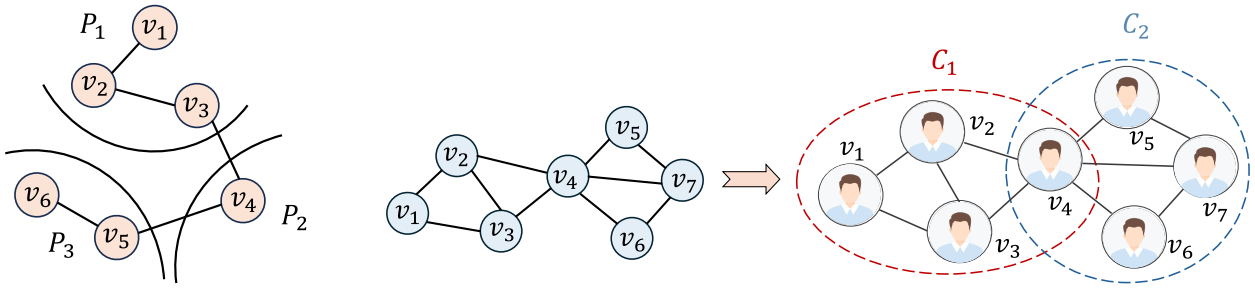
UTXO-based cross-shard transactions: In Fig. 2(a), a transaction tx with multiple inputs is created by $tx^1_{in}, tx^2_{in}, \cdots, tx^i_{in}$. Let $S^1_{in}, S^2_{in}, \cdots, S^i_{in}$ and S_{out} denote the shards that contain $tx^1_{in}, tx^2_{in}, \cdots, tx^i_{in}$ and tx, respectively. If the shards $S^1_{in}, S^2_{in}, \cdots, S^i_{in}$ and S_{out} are not the same, a UTXO-based cross-shard transaction will occur.

Account/Balance-based cross-shard transactions: Fig. 2(b) shows a transaction that is initiated between two accounts $Account_{sender}$ and $Account_{receiver}$ that are located in S_{in} and S_{out}; it is called a cross-shard transaction based on the Account/Balance model, if the shards S_{in} and S_{out} are not the same. Notably, a single transaction could be associated with multiple users.

B. Transaction Graph

A graph structure, referred to as a "transaction network" is constructed separately based on the characteristics of the UTXO and Account/Balance transaction models. Depending on the distinct characteristics of the UTXO and Account/Balance transaction networks, appropriate graph optimization techniques are applied to simultaneously achieve workload balance and minimize cross-shard communication.

1) Graph Construction: As mentioned previously, there is a distinction in the transaction data structure between the UTXO-based model and the Account/Balance-based model. When executing a transaction based on the UTXO model, a sufficient number of UTXOs should be selected from a set of UTXOs as the input of the transaction. Newly generated UTXOs are appended to the set, while spent UTXOs are subsequently removed [2]. Consequently, each transaction becomes intrinsically linked to prior transactions, resulting in a chained transactional structure. In contrast, the Account/Balance model



(a) Graph partitioning. (b) Community detection.

Fig. 3. Two methods of graph analysis.

is comparatively streamlined. During a transaction, the corresponding account balance is adjusted directly based on the transaction type (payment or deposit). In the UTXO model, each transaction depends on other transactions, whereas in the Account/Balance model, transactions only involve the transfer of funds between different accounts without considering dependencies. Based on these characteristics, we define two types of transaction graphs, respectively.

The graph of UTXO-based transactions: Since UTXOs and transactions based on them do not involve double spending and exhibit dependencies, the transaction graph can be represented as a directed graph $G = (V, E)$, where V denotes the set of transactions and E comprises UTXOs, with $(u, v) \in E$ if transaction u uses the UTXO created by transaction v [19].

The graph of Account/Balance-based transactions: An account is an entity, and transactions are more like interactions between entities. The transaction graph can be established as a directed graph $G = (V, E)$, where V represents accounts and E denotes transactions, with $(v_i, v_j) \in E$ if the transactions involve the transfer of assets between these two accounts [27]. Additionally, to identify the community structure among accounts, a weight attribute can be assigned to indicate transaction frequency.

2) Graph Analysis Methods: Graph partitioning and community detection [35] techniques are two fundamental techniques in transaction allocation optimization. As shown in Fig. 3(a), graph partitioning aims to divide the transaction graph into non-overlapping parts, ensuring that each entity belongs to only one part. The objective is to minimize the number of edges between subsets (i.e., cross-shard communication) while considering additional factors such as load balance [36]. In contrast, community detection focuses on identifying tightly connected entities and uncovering the community structure within a network, as illustrated in Fig. 3(b).

The UTXO-based transaction graph is inherently dynamic and sequential, whereas the Account/Balance-based transaction graph is characterized by statistical relationship patterns among accounts. Streaming graph partitioning technology [37] is particularly well-suited for the UTXO-based transaction graph, whereas community detection technology serves the Account/Balance-based transaction graph better.

For example, in the context of graph partitioning for UTXO-based transactions, Nguyen *et al.* modeled the transaction graph as Transaction-as-Node (TaN) and proposed a streaming graph algorithm called OptChain. This algorithm effectively allocates transactions to an optimal shard by evaluating fitness scores between transactions and shards [19]. For the Account/Balance model, Li *et al.* accurately modeled transaction load and cross-shard communication and introduced a limiting factor to enhance a label propagation algorithm. This algorithm identifies community structures that limit transaction load scale and achieves the synchronous optimization of cross-shard communication and workload balance [27].

C. Transaction Consistency

As a unique distributed ledger, blockchain must consider the attributes of distributed systems and databases. From a distributed system perspective, blockchains adhere to the Consistency, Availability, and Partition Tolerance (CAP) theorem, which states that a system can simultaneously satisfy at most two of the three core properties at any given time [38]. From a database perspective, transactions must follow the Atomicity, Consistency, Isolation, and Durability (ACID) properties [39]. Consequently, consistency represents not only the critical CAP property, but also the ACID property, showing a shared requirement across both paradigms in blockchain systems.

Achieving consistency is a primary goal of transactions involving multiple shards. In distributed operations, the consistency property ensures that all nodes within a given shard access and produce the same consistent state. Additionally, updates are executed in timestamp order or according to a predefined sequence whenever possible [40]. Therefore, cross-shard transaction submission protocols must address two key challenges while maintaining transactional atomicity. First, a conflict identification mechanism should be implemented to handle potentially conflicting transactions initiated by multiple nodes simultaneously, ensuring that only one is accepted for execution. Second, accepted transactions must be processed in a predefined sequence across all participating nodes to guarantee global data consistency [41].

In blockchain sharding, the ledger of each shard operates independently, with consensus protocols such as PoW and Byzantine Fault Tolerance (BFT) [42], [43]. Blockchain sharding techniques enable a trade-off between consistency and availability while preserving partition tolerance. The consistency of cross-shard transaction processing mechanisms can be classified into two categories:

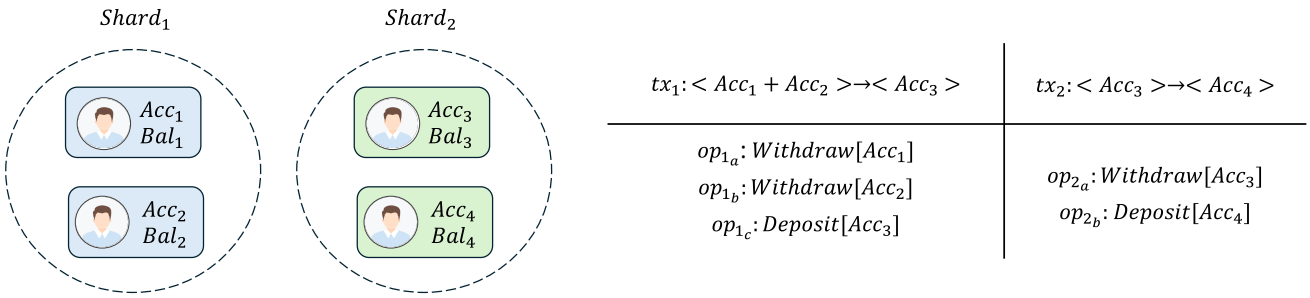


Fig. 4. An example of transaction conflict. If tx_1 reads the intermediate state of tx_2 or tx_2 reads the intermediate state of tx_1, it results in a violation of transaction isolation.

- Strong consistency [44]: all operations involved in a cross-shard transaction must either commit or abort as a single atomic unit, ensuring that there are no intermediate states in the system [45]. This ensures that at any instant, the system is in a consistent state. For example, the SGX-Shard employs 2PC for cross-shard transactions, which although ensures strong consistency, increases transaction confirmation latency and reduces system throughput.
- Eventual consistency [46]: Eventual consistency is usually adopted in the eventual sharded blockchain. This category allows for temporary inconsistency among shards, ensuring that the final state becomes consistent over time [47]. For instance, Monoxide [15] uses eventual atomicity and relay transactions to enhance transaction concurrency. However, this approach could introduce transaction conflicts, requiring additional concurrency control.

Achieving strong consistency typically results in higher latency. Overall, cross-shard transaction processing protocols require a careful trade-off between consistency and latency, aligning with the PACELC theory[1] [48]. Developing consensus protocols that can efficiently handle cross-shard transactions while minimizing latency is a key issue that should be explored.

D. Transaction Concurrency

A sharded blockchain system allows for the concurrent execution of multiple transactions, thereby improving system throughput. However, it inevitably raises the problem of transaction conflicts, especially in the context of cross-shard transactions. Concurrent execution becomes infeasible when transactions access shared resources. In the UTXO model, this means accessing the same elements within the UTXO set, whereas in the Account/Balance model, it means accessing the same account and/or state [49].

In the sharded blockchain system, the shards reach consensus independently and operate without sharing their states. Consequently, local shards maintain only partial cross-shard transaction records rather than complete data. Moreover, no coordinator exists to manage the overall transaction sequence. When concurrent transactions from other shards attempt to modify a shard's state, conflicts with local transactions become highly possible.

Building on a case introduced by [12], we consider an account/balance-based asset transfer scenario illustrated in Fig. 4. Let $tx_1 : \langle Acc_1 + Acc_2 \rangle \rightarrow \langle Acc_3 \rangle$ be a transaction transferring assets from accounts Acc_1 and Acc_3 to Acc_2. Whereas $tx_2 : \langle Acc_3 \rangle \rightarrow \langle Acc_4 \rangle$ be another transaction transferring assets from account Acc_3 to Acc_4 submitted roughly at the same time as tx_1, where Acc_1 and Acc_2 belong to $Shard_1$, Acc_3 and Acc_4 belong to $Shard_2$. If the transaction sequence of the two asset transfers is $\langle op_{1_a}, op_{1_b}, op_{2_a}, op_{1_c}, op_{2_b} \rangle$, it breaks isolation. Because the withdrawal operation of tx_2 is executed prior to the deposit operation of tx_1, tx_2 observes the state of the partially completed transaction tx_1, potentially resulting in inconsistent balance views across different transactions.

To prevent inconsistent states caused by transaction conflicts and ensure isolation, sharded blockchain systems require concurrency control [50] to correctly process transactions that may be in conflict. This means that transactions being executed are not affected by other concurrent transactions. In sharded blockchain systems, most of these concurrency control algorithms use one of two basic mechanisms: Pessimistic Concurrency Control (PCC) [51] and Optimistic Concurrency Control (OCC) [52].

For instance, SGX-Shard uses a 2PC and 2PL-based approach [51] to concurrency control, which requires one shard to wait for the readiness of related shards before handling cross-shard transactions. Conversely, X-shard [53] adopts an optimistic approach, permitting concurrent transactions in an input shard. Upon committing a cross-shard transaction, each recipient shard validates dependencies. If conflicts arise, the shard aborts and rolls back the transaction. Both methods exhibit inherent limitations. In SGX-Shard, account balances are locked during transaction execution, thereby stalling other transactions dependent on these balances and leading to elevated latency and diminished throughput. While OCC improves concurrency, its higher abort rate in smart contract environments increases rollback frequency, which adversely impacts overall throughput [54].

III. Evaluation Criteria

We propose two sets of criteria for evaluating the performance of CTAOP mechanisms. We first summarize five

[1]The PACELC theorem is an extension to the CAP theorem. It states that in case of network partitioning (P) in a distributed computer system, one has to choose between availability (A) and consistency (C) (as per the CAP theorem), but else (E), even when the system is running normally in the absence of partitions, one has to choose between latency (L) and loss of consistency (C).

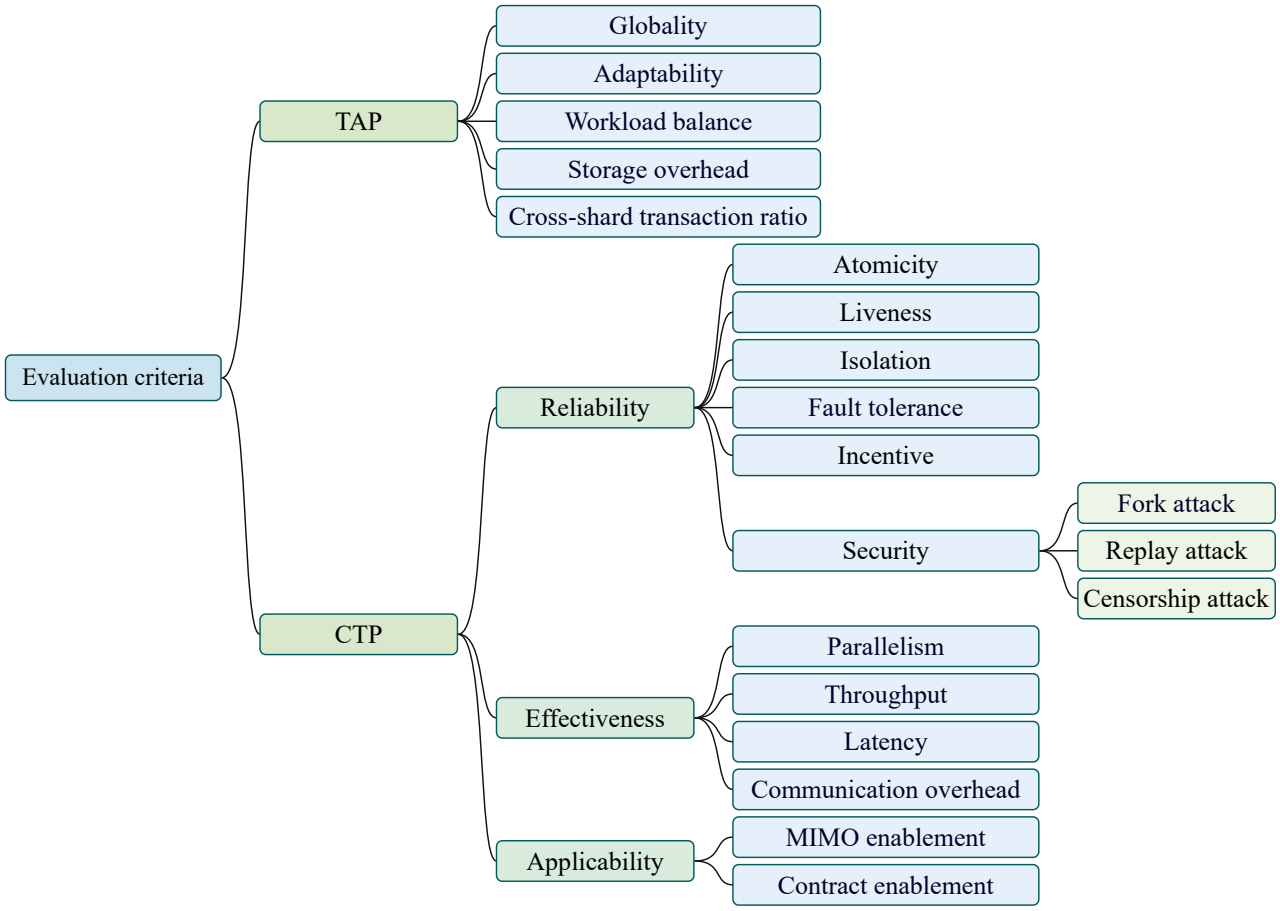


Fig. 5. Evaluation criteria of CTAOP mechanisms. TAP refers to transaction allocation optimization, and CTP refers to cross-shard transaction processing.

specific evaluation criteria for evaluating transaction allocation optimization schemes. Then, we propose a set of evaluation metrics for cross-shard transaction processing mechanisms. These evaluation metrics are shown in Fig. 5.

A. Transaction Allocation Optimization Criteria

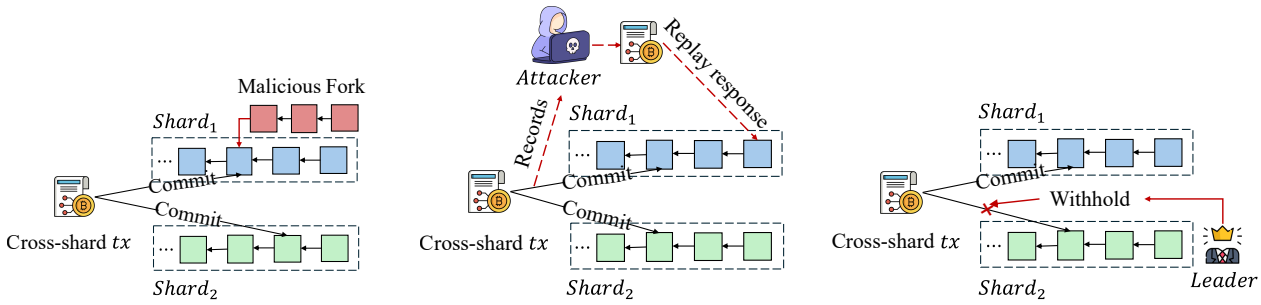
We propose the following criteria to evaluate transaction allocation optimization schemes.

1) Globality (GL): Globality entails the consideration of transaction dependencies and historical account interaction data. By analyzing transaction dependencies, optimal shard allocation can be determined for UTXO-based transactions. Similarly, by analyzing account interaction data, frequently interacting accounts can be grouped together. This approach provides precise guidance for transaction allocation based on long-term statistical analysis, reducing cross-shard transactions and achieving anticipated performance gains. Therefore, globality is a critical criterion for transaction allocation optimization schemes.

2) Adaptability (AP): Adaptability refers to the ability of transaction allocation to be dynamically adjusted according to fluctuations in network load during the period of consensus [55]. It enables the system to modify its allocation strategy based on current conditions, such as account migration. Adaptability significantly enhances the performance of transaction allocation optimization and mitigates shard overload caused by network fluctuations during a consensus epoch. As such, adaptability is an indispensable criterion for transaction allocation optimization schemes.

3) Workload Balance (WB): Workload balance ensures equitable workload distribution across shards, optimizing overall performance and resource utilization. In high-load "hot" shards, transaction congestion increases confirmation latency, while low-load "cold" shards underutilize computing resources and fail to achieve optimal throughput. Workload balance improves shard utilization, enhancing system throughput and reducing acknowledgment latency. Therefore, an effective transaction allocation scheme must balance the workload across shards.

4) Storage Overhead (SO): Storage overhead refers to the additional burden on nodes for storing transaction relationships during the transaction allocation process. High storage overhead consumes substantial resources and results in increased transaction allocation latency. To facilitate comparison and analysis, space complexity serves as an indicator for measuring storage consumption. It is affected by the data structure of a



(a) Fork attack. (b) Replay attack. (c) Censorship attack.

Fig. 6. Attacks against cross-shard transactions.

transaction set. This data structure may be either a transaction dependency graph containing x transactions, an account interaction graph containing y accounts, or other structures.

5) Cross-shard Transaction Ratio (CTR): In contrast to intra-shard transactions, cross-shard transactions require coordination of state across multiple shards, which increases confirmation latency, communication overhead, and potential security risks. A key objective of the transaction allocation optimization schemes is to reduce the frequency of cross-shard transactions, while minimizing unnecessary overhead to boost overall system performance. The effectiveness of these schemes is directly measurable using this criterion.

B. Cross-shard Transaction Processing Criteria

We propose a set of criteria for evaluating the performance of cross-shard transaction processing mechanisms in terms of reliability, effectiveness, and applicability.

1) Reliability-related Criteria: Cross-shard transaction processing mechanisms should ensure consistency and efficiency in transaction execution, even in the presence of transaction conflicts, malicious nodes, and potential attacks. Therefore, we propose the following criteria to evaluate the reliability of cross-shard transaction processing mechanisms.

a) Atomicity (AM): Atomicity refers to the principle that transactions are indivisible, that is, they are either executed completely or not at all. If a transaction's components are executed on different shards and one part fails, the transaction is deemed a failure, and all changes are rolled back [45]. The atomic commit mechanism guarantees that cross-shard transactions are recorded in the ledgers of all related shards, thus ensuring consistency in the ledger state. Therefore, atomicity is an indispensable criterion for a cross-shard transaction processing mechanism.

b) Liveness (LN): Liveness refers to that a cross-shard transaction can be eventually processed, i.e., the involved shards continuously make progress. Liveness guarantees that the sharded blockchain system responds to clients within a certain period of time. Even in the presence of network delays, communication issues between shards, or malicious nodes, the system ensures that transactions are not indefinitely blocked [45]. Liveness is critical for maintaining system responsiveness and reliability.

c) Isolation (IL): Isolation refers to the condition in which multiple transactions can be executed simultaneously, but the execution of each transaction is independent of the others. Cross-shard transactions may access and modify ledger state simultaneously with other concurrent transactions, which inevitably leads to transaction conflicts and ledger state inconsistencies. The cross-shard transaction mechanism should provide a level of isolation to ensure that cross-shard transactions are not interfered with by other concurrent transactions. This is typically achieved through concurrency control mechanisms such as locking or timestamp ordering, which prevent issues like data inconsistency.

d) Fault Tolerance (FT): Fault tolerance describes the capability of each shard to operate stably in the presence of Byzantine nodes [56]. Fault tolerance is evaluated using the fault tolerance ratio, expressed as f/n, where f denotes the maximum number of Byzantine nodes within a shard, and n represents the total number of nodes in the shard. Byzantine nodes always broadcast error messages and deny services to hinder transaction consensus. Therefore, a reliable processing mechanism requires high fault tolerance of each shard to ensure its availability. We assume that the BFT consensus protocol can achieve 1/3 fault tolerance while the PoW consensus protocol can achieve 1/2 fault tolerance.

e) Security (SE): Security refers to the capability of the cross-shard processing mechanism to handle cross-shard transactions even in the face of various potential attacks. The security of a cross-shard transaction processing mechanism is measured by the variety of attacks it can defend against. A reliable cross-shard processing mechanism must be able to provide countermeasures against potential attacks. The more comprehensive the attacks it can withstand, the safer the transaction processing becomes. Next, we present several attacks against cross-shard transactions, while their characteristics are shown in Fig. 6.

- Fork Attack (FA): In a PoW consensus system, a cross-shard transaction is initially recorded in the ledgers of multiple shards. Subsequently, miners in a shard may extend a longer chain to revert the transaction, leading to inconsistencies among shards [57]. The attacker exploits the inconsistency of the shard ledgers, which causes the shards to have conflicting views on cross-shard transactions.

- Replay Attack (RA): The attacker records the response of shards and replays it during other protocol instances, thus deceiving other entities involved in the transaction, including shards and clients [58]. This attack exploits the weak binding between shards and transactions, causing double spending and resource locking.
- Censorship Attack (CA): The leaders within shards operate honestly within their own shards, but engage in malicious behavior when it comes to cross-shard transactions, such as withholding certificates [21]. This attack exploits the inability of single shard to maintain a global view, disrupting the normal execution of cross-shard transactions and leading to transaction blocking.

f) Incentive (IN): Incentive is designed to regulate the behavior of system entities from an economic perspective. Its goal is to encourage system entities to achieve specific objectives [59], such as actively participating in cross-shard transactions and honestly behaving. In the context of blockchain, an incentive usually increases the utility of nodes by rewarding them, thereby motivating them to join the blockchain's execution. On the one hand, the incentive can encourage nodes to maintain the atomicity and liveness of cross-shard transaction processing mechanisms. On the other hand, it can also prevent various attacks, ensuring that the cross-shard shard transaction processing mechanisms operate normally and as expected. Therefore, incentive plays a crucial role in the process of cross-shard transactions.

2) Effectiveness-related Criteria: Effectiveness refers to the efficiency and overhead of a cross-shard processing mechanism when handling cross-shard transactions. Quantitative evaluation is necessary to accurately assess whether a mechanism can satisfy this type of criteria.

a) Parallelism (PL): Parallelism refers to the ability of a single shard to process multiple cross-shard transactions simultaneously. Cross-shard transaction processing mechanisms typically include two parallel processing methods, batch and pipeline [60]. Supporting either one or both methods means that parallelism is satisfied. Batch means that a single block can contain the input or output of multiple cross-shard transactions [61]. Pipelining splits transaction processing into multiple stages, allowing a single shard to perform different stages of processing in parallel [62]. If a cross-shard transaction processing mechanism supports parallelism, it can significantly enhance the efficiency of the sharded blockchain.

b) Throughput (TP): Throughput measures the volume of transactions a blockchain system can handle within a specified time period. A cross-shard transaction processing mechanism aims to maximize transaction throughput within a given time. Consequently, throughput serves as a key metric for evaluating processing efficiency.

c) Latency (LT): Latency is the time elapsed from when a transaction enters the transaction pool to its final recording in the ledger. Low latency implies quick confirmation and execution of transactions, thus reflecting the real-time process capability of the cross-shard transaction processing mechanism. This is particularly critical for applications that demand immediate confirmation, such as financial transactions. Therefore, latency is a crucial metric for evaluating the real-time efficiency and responsiveness of the cross-shard transaction processing mechanism.

d) Communication Overhead (CO): The communication overhead denotes the data exchanged among shards during the processing of cross-shard transactions. This overhead is influenced by various factors, including the size of each shard (m) and the total number of shards (n) involved in cross-shard transactions. Excessive communication overhead not only undermines the system's scalability, but also impairs the overall performance and responsiveness of the blockchain network. Therefore, minimizing communication overhead is essential for optimizing the efficiency of the cross-shard processing mechanism, ensuring rapid transaction finality and a positive user experience.

3) Applicability-related Criteria: Applicability refers to the ability of the cross-shard transaction processing mechanism to be applied in practice to handle complex transactions involving multiple accounts, multiple assets, and interactions with smart contracts. An applicable cross-shard transaction processing mechanism must effectively manage this complexity.

a) Contract Enablement (CE): Contract enablement refers to the capability of a sharded blockchain to process smart contract-based cross-shard transactions. A smart contract is a program executed on a blockchain that automatically enforces contractual terms when predefined conditions are met [63]. These contracts are applicable across various domains, such as financial, supply chain, voting systems, identity verification, and beyond [64]. Therefore, to accommodate diverse application requirements, the cross-shard transaction processing mechanism must support smart contracts. Notably, UTXO-based transactions do not inherently support smart contracts.

b) MIMO Enablement (ME): Multiple Input and Multiple Output (MIMO) enablement denotes the capability of a sharded blockchain system to handle MIMO cross-shard asset transactions. Notably, all UTXO-based transactions inherently support MIMO. The flexibility and concurrency of MIMO transactions enable cross-shard transaction processing mechanisms to no longer be limited to inefficient Single Input Single Output (SISO) transactions, thereby also reducing cross-shard communication. Therefore, MIMO enablement serves as a valuable measure of the applicability of the cross-shard transaction processing mechanisms.

IV. Taxonomy of CTAOP Mechanisms

This section gives a detailed taxonomy of CTAOP mechanisms, as shown in Fig. 7. We first classify CTAOP mechanisms into transaction allocation optimization schemes and cross-shard transaction processing mechanisms based on the differences of optimization objectives. Subsequently, we further propose detailed taxonomies of the two main types of mechanisms and explore the advantages and disadvantages of their respective sub-taxonomies, as shown in Table III.

A. Taxonomy of Transaction Allocation Optimization Schemes

The primary objective of transaction allocation optimization schemes is to minimize cross-shard transactions while maintaining workload balance. As illustrated in Fig. 7, existing

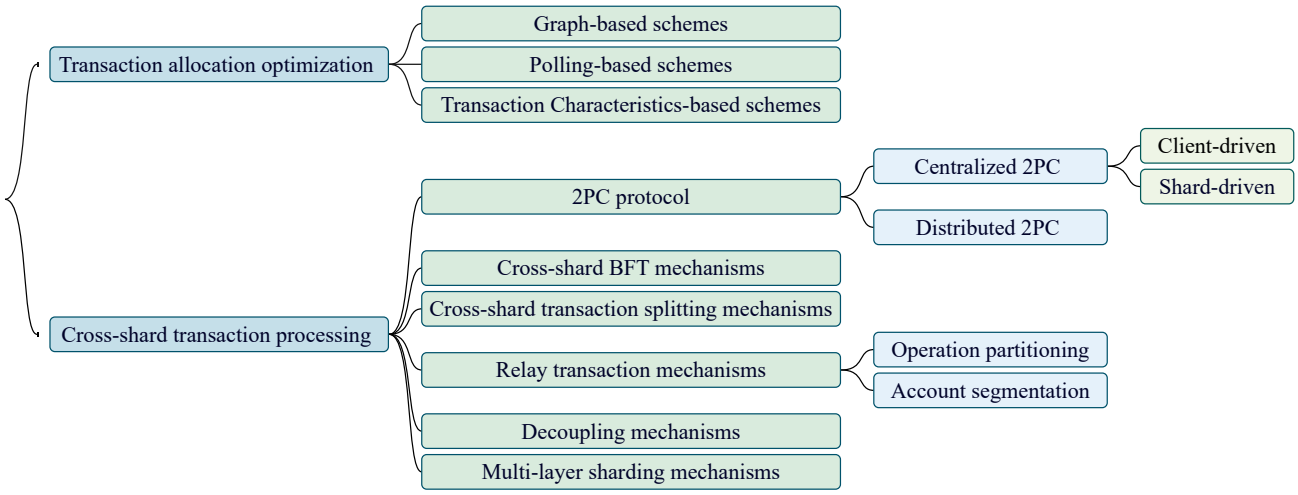


Fig. 7. Taxonomy of CTAOP mechanisms.

transaction allocation optimization schemes can be categorized into three types based on their allocation strategies: graph-based, polling-based, and transaction characteristics-based schemes. Additionally, we summarize the respective advantages and limitations of each approach.

1) Graph-based Schemes: Graph-based transaction allocation schemes transform historical transaction networks into graph structures and utilize graph algorithms to assign accounts and transactions. The core principle is to co-locate dependent transactions and frequently interacting accounts within the same shard. This approach captures global interaction patterns and performs combinatorial optimization of transaction placement [65]. However, constructing and processing graph structures introduces significant computational and storage overhead, thereby increasing the burden on blockchain nodes.

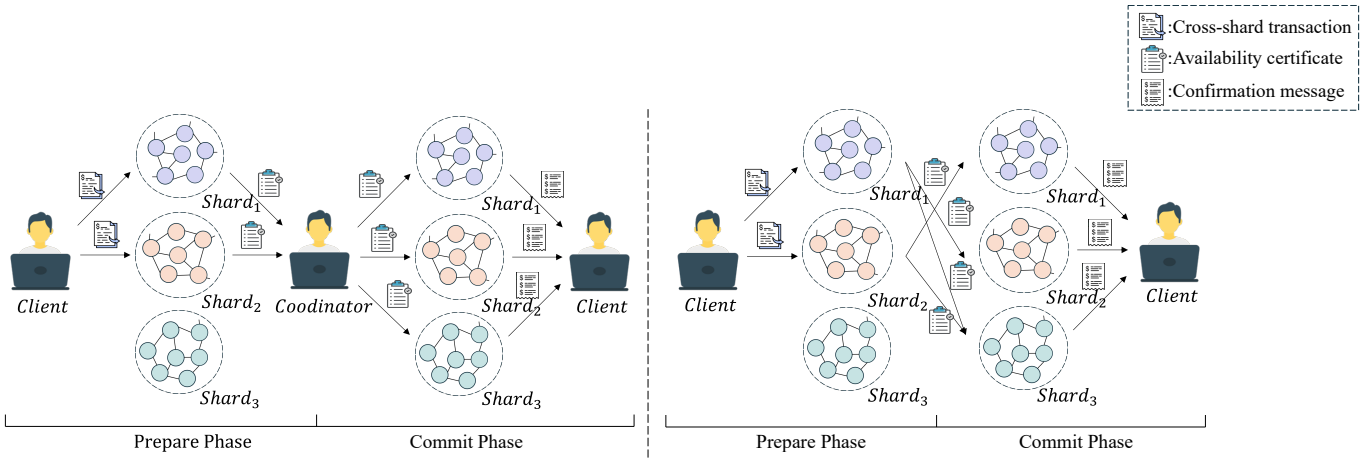
2) Polling-based Schemes: Polling-based transaction allocation schemes dynamically assess whether to migrate participating accounts to different shards for each transaction. While this approach can partially achieve load balancing, it neglects the relationships among existing accounts, thereby limiting its effectiveness in reducing cross-shard transactions. Moreover, account migration may disrupt previously optimized configurations, resulting in suboptimal cross-shard transaction processing.

3) Transaction Characteristics-based Schemes: Transaction characteristics-based allocation schemes assign transactions based on specific features of transactions or accounts, such as involved participants or transaction types. Although simple and efficient, this approach has limited applicability and may not be suitable for diverse transaction patterns. Additionally, it often lacks comprehensive performance considerations, necessitating further refinement in future designs.

B. Taxonomy of Cross-shard Transaction Processing Mechanisms

The objective of cross-shard transaction processing mechanisms is to enhance the processing efficiency of cross-shard transactions. We have categorized existing cross-shard transaction processing mechanisms into six types based on different design principles. These types are further summarized according to their differences in applicable transaction models: the general models, the UTXO model, and the Account/Balance model. The general model category includes the 2PC protocol and the Cross-shard Byzantine Fault Tolerance (Cross-Shard BFT) mechanism; the UTXO model category includes the cross-shard transaction splitting mechanism; and the Account/Balance model category consists of the relay transaction mechanism, the decoupling mechanism, and the multi-layer sharding mechanism, as shown in Fig. 7. Finally, for each of the six types, we summarize its basic process and discuss its advantages and disadvantages, as shown in Table III.

1) 2PC Protocol: The 2PC protocol is the most widely adopted method for handling cross-shard transactions, which can be deployed in both models. As shown in Fig. 8, this protocol consists of a preparation phase and a commitment phase. In the preparation phase, all input shards must generate an availability certificate to confirm the availability of an input. Each shard then transmits its agreement results, along with the certificate, to a coordinator or coordinators [45]. Once an input is confirmed as available, it is locked to prevent double-spending by other transactions. In the commitment phase, the coordinator(s) distribute the collected certificates to all relevant shards, which then verify the availability of the inputs. If all inputs are deemed available, the transaction is validated and committed. Conversely, if any shard reports an input as unavailable, the transaction is invalidated, and the other shards release the previously locked inputs [11]. There are



(a) Centralized 2PC protocol. (b) Distributed 2PC protocol.

Fig. 8. Centralized 2PC protocol versus distributed 2PC protocol. The primary distinction between the two types of 2PC protocols lies in their communication structure, specifically whether a third party is involved in coordinating cross-shard transactions.

several different ways to implement the 2PC protocol, either centralized or distributed based on communication structure, specifically whether a third party is involved in coordinating cross-shard transactions.

a) Centralized 2PC Protocol: As illustrated in Fig. 8(a), a central transaction coordinator oversees the execution of cross-shard transactions. Its responsibilities include collecting proofs and disseminating them to the relevant shards. Communication is centralized, with all shards communicating directly with the coordinator, resulting in relatively high communication efficiency. Based on the role of the centralized coordinator, the protocol can be classified into two types: client-driven and shard-driven.

- Client-driven: The client is responsible for collecting and transmitting proofs. However, this type of protocol also inevitably increases the burden on clients. In case the client fails, the transaction may be blocked [12].
- Shard-driven: Specific shards run the BFT protocol to coordinate cross-shard transactions. Although liveness is guaranteed, concurrency is hard to satisfy [12].

b) Distributed 2PC Protocol: As illustrated in Fig. 8(b), the coordinators are composed of shards participating in transactions, requiring the exchange of proofs of fund availability. The communications among them are distributed, without any central coordinator. Compared to the centralized 2PC, the burden on a specific coordinator is released [6]. The design of the distributed 2PC is more complex than that of the centralized 2PC. Multiple rounds of message exchange increase communication overhead.

2) Cross-Shard BFT Mechanism: The Cross-Shard BFT mechanism is designed to achieve consensus among nodes across multiple shards. As illustrated in Fig. 9, this mechanism involves two main roles: primary participant and backup participants. Specifically, the primary participant, also known as a leader, is typically the leader node of the input shard and is responsible for handling client requests and initiating proposals. The backup participants, typically consensus nodes within all the shards that participate in the transaction, adhere

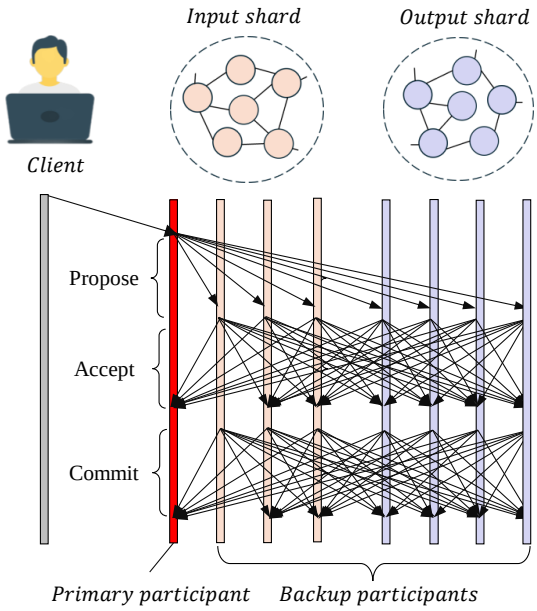


Fig. 9. Cross-Shard BFT mechanism.

to the principle that "the minority is subordinate to the majority," and need to accept messages from $2f + 1$ or more of the participants [45].

This mechanism eliminates the need for a coordinator to collect and distribute proofs of availability of funds. Additionally, Cross-Shard BFT mechanisms effectively address challenges such as transaction conflicts, deadlocks, and participant failures [66]. However, message exchanges across all nodes involved in shard transactions result in high communication overhead.

3) Cross-shard Transaction Splitting Mechanism: As shown in Fig. 10, the cross-shard transaction splitting mechanism processes the general MIMO transaction by splitting cross-shard transactions into multiple SISO sub-transactions.

TABLE III
SUMMARY AND COMPARISON OF CTAOP MECHANISMS

Categories		Advantages	Disadvantages
TAOS	GS	1. Analyze global transaction patterns; 2. Optimize transaction allocation combinatorially [65].	Generating graph structures incurs additional costs, causing extra node burden.
	PS	Partially achieve load balance.	1. Limited ability to reduce cross-shard transactions; 2. Account migration may disrupt previously optimized configurations.
	TCS	Simple and fast.	1. Limited application scope; 2. Require further performance optimization.
CTPM	C2PC	Relatively high communication efficiency.	1. Increase the burden of clients [12]; 2. Insufficient concurrency [12].
	D2PC	Release the burden of a centralized coordinator.	Multiple rounds of message exchange increase communication overhead.
	CSBFT	Solve transaction conflict, deadlock, and failure [66].	Direct participation of nodes in cross-shard communication brings high overhead.
	CSTSM	Simplify consensus [6].	Sub-transactions introduce additional communication overhead.
	OP-RTM	1. Enhance transaction processing flexibility; 2. Improve system concurrency [15].	1. Difficult to handle multi-input transactions; 2. Sub-transactions cause an increase of communication overhead; 3. Multi-step leads to high latency.
	AS-RTM	1. Quick cross-shard transaction confirmation [17]; 2. Shard load can be adjusted.	1. Cannot support multiple input transactions; 2. Fine-grained partition of account states increases protocol complexity; 3. Ensuring account trust presents a significant challenge.
	DM	Facilitate collaboration on complex cross-shard transactions [67].	Increase system implementation complexity.
	MLSM	Prevent invalid and conflicting transactions.	1. Storage bottleneck; 2. The complexity of cross-shard consensus affects processing efficiency and communication overhead.

TAOS: Transaction allocation optimization scheme; CSTPM: Cross-shard transaction processing mechanism;
GS: Graph-based schemes; PS: Polling-based schemes; TCS: Transaction characteristics-based schemes;
C2PC: Centralized 2PC protocol; D2PC: Distributed 2PC protocol; CSBFTM: Cross-Shard BFT mechanism;
CSTSM: Cross-shard transaction splitting mechanism; OP-RTM: Operation partition based relay transaction mechanism;
AS-RTM: Account segmentation based relay transaction mechanism; DM: Decoupling mechanism; MLSM: Multi-layer sharding mechanism.

For a cross-shard transaction $tx = \langle (I_1, I_2), O \rangle$ with two inputs I_1 and I_2 that belong to $Shard_1$ and $Shard_2$, respectively, and one output O that belongs to $Shard_3$. $Shard_3$ creates two types of transactions $tx_i = \langle I_i, I_i' \rangle$ and $tx' = \langle (I_1', I_2'), O \rangle$. For $i \in \{1, 2\}$, tx_i has input I_i and output I_i', where $|I_i'| = |I_i|$ (i.e., the same amounts), and I_i' belongs to $Shard_3$ [6]. While the transaction splitting mechanism simplifies cross-shard transaction processing, it introduces additional communication overhead due to the creation of sub-transactions.

4) Relay Transaction Mechanism: We categorize the relay transaction mechanism into Operation Partition and Account Segmentation, as shown in Fig. 11. The essence of the relay transaction mechanism lies in dividing the transaction state across multiple shards, where a source shard and a target shard execute different operations. Overall, the relay transaction mechanism can serve as a sub-scheme within other cross-shard transaction processing mechanisms and is commonly applicable to SISO scenarios. The above two classes of the relay transaction mechanism are described below.

a) Operation Partition: As illustrated in Fig. 11(a), the relay transaction mechanism based on operation partition decouples cross-shard transactions into multiple steps, each involving a single shard. The source shard transmits these steps to the target shard by deriving and forwarding relay transactions, similar to a relay race [15]. Operation partitioning enhances the flexibility and scalability of transactions, thereby improving the concurrency and scalability of the system.

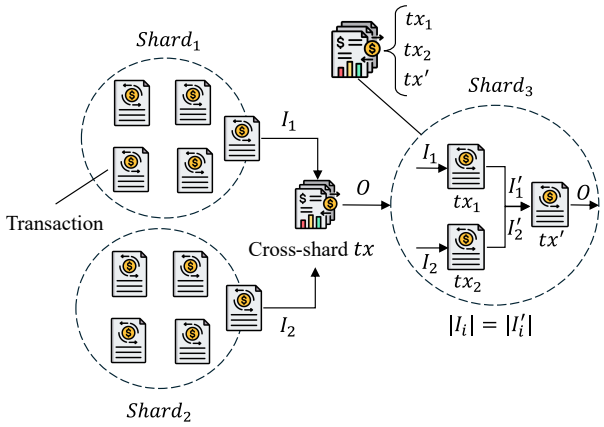
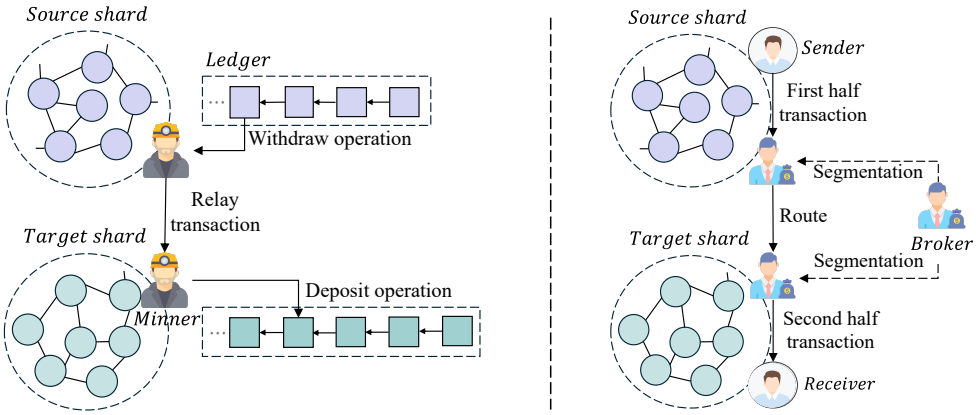


Fig. 10. Cross shard transaction splitting mechanism.



(a) Operation partition. (b) Account Segmentation.

Fig. 11. Two implementations of the relay transaction mechanisms. In the operation partition, the miners of the source shard verify a withdraw operation, record it in the ledger, and subsequently forward the relay transaction to the target shard. After the relay transaction is verified by the target shard, it is recorded in the ledger and executes a deposit operation. In the account segmentation, the cross-shard transaction process is divided into two stages. Trusted accounts are employed to execute relevant operations within each shard, thereby enabling the concurrent processing of deposit and withdrawal operations.

However, it encounters three potential challenges. First, it is difficult to handle multi-input transactions relying on the relay transaction mechanism. Second, an increase in the number of sub-transactions results in higher overhead. Finally, multi-step processing itself causes high latency to some extent.

b) Account Segmentation: Account segmentation refers to the account states of some special users being split and stored in multiple shards [16], [17], as illustrated in Fig. 11(b). If these special accounts are fully trusted, cross-shard transactions can be processed within a single shard, reducing the need for inter-shard communication and lowering latency [16]. However, implementing account segmentation requires the design of fine-grained account state partitions, which inevitably increases the complexity of the mechanism. Additionally, ensuring the trustworthiness of these special accounts poses a significant challenge.

5) Decoupling Mechanism: The core concept of the decoupling mechanism is to separate transaction recording from consensus execution by coordinating the state storage and transaction execution processes. The mechanism disrupts ledger isolation between shards, enabling each shard to share the ledger during transaction execution and facilitating collaboration on complex cross-shard transactions [67]. Inevitably, implementing the decoupling mechanism requires complex system design and coordination mechanisms, which increases the difficulty of implementation.

6) Multi-layer Sharding Mechanism: As shown in Fig. 12, this mechanism selects nodes from each shard to form a new-layer shard that coordinates all cross-shard transactions, so as to solve invalid transactions and transaction conflicts. For example, Prophet [54] addresses conflicts by establishing a reconnaissance coalition and implementing deterministic global ordering, whereas DC_Chain [68] proactively resolves conflicts through the formation of a decentralized coordinator.

Despite the advantages, the introduction of the coordination layer poses several challenges. The shards in the coordination layer store ledger data from multiple shards, leading to increased storage overhead and potential system bottlenecks. Moreover, the coordination layer requires stringent security protocols to ensure robust defenses. Finally, as cross-shard transactions traverse multiple layers, the extra complexity of cross-shard consensus may impact overall consensus efficiency and communication overhead.

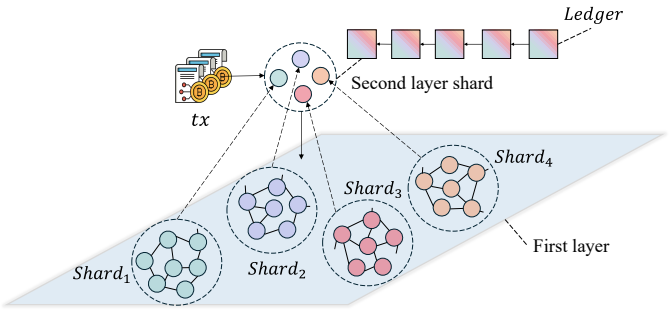


Fig. 12. Multi-layer sharding mechanism.

V. CTAOP Mechanisms

This section gives a detailed literature review on existing CTAOP mechanisms by employing our proposed criteria. We retrieve and thoroughly review a total of 29 relevant research papers published between 2017 and 2024, searched from databases including IEEE Xplore, ACM Digital Library, Google Scholar, Web of Science, and arXiv. Among these, 10 papers cover topics related to transaction allocation optimization schemes, while 19 papers focus on cross-shard transaction processing mechanisms. Finally, we discuss our findings and compare the reviewed works.

A. Review on Transaction Allocation Optimization Schemes

In this subsection, we review the transaction allocation optimization mechanisms following their taxonomy by evaluating them using our proposed criteria. Our review results are shown in Table IV.

TABLE IV
SUMMARY AND COMPARISON OF TRANSACTION ALLOCATION OPTIMIZATION SCHEMES

Category	Ref	TM	GL	AP	WB	SO
Graph-based schemes	METIS [69]	A/B	●	○	○	$O(y^2)$
	Optchain [19]	UTXO	●	●	●	$O(x^2)$
	CLPA [70]	A/B	●	○	●	$O(y^2)$
	RGP [71]	UTXO	●	○	●	$O(kx)$
	Txallo [65]	A/B	●	○	●	$O(y^2)$
	Estray [72]	A/B	●	○	●	$O(y^2)$
Pollong-based schemes	Scheduler [73]	A/B	○	●	●	$O(y)$
	LB-Chain [74]	A/B	○	●	●	$O(y)$
Transaction characteristics-based schemes	OBSC [18]	A/B	○	●	●	$O(1)$
	Sliver [75]	A/B	○	○	●	$O(t)$

●: Meet the criterion; ○: Do not meet; '-': Not Available; UTXO: UTXO model; A/B: Account/Balance model.Ref: Reference; TM: Transaction Model; GL: Globality; AP: Adaptability; WB: Workload Balance; SO: Storage Overhead; x: The number of transactions; y: The number of accounts; k: The levels of ancestral transactions; t: The number of cross-chain transactions.

1) Graph-based Schemes: Fynn *et al.* [69] introduced the classical offline graph algorithm METIS [76] to partition accounts. Similarly, Huang *et al.* [16], [17] adopted the METIS algorithm as an automated transaction distribution strategy in Brokerchain [16], [17]. The METIS algorithm initially constructs an account transaction graph, assigning weights to its vertices and edges. The edge weights are defined by the number of transactions between corresponding account pairs, while a vertex's weight is computed as the total weight of its connected edges. The algorithm periodically determines the optimal partitioning of the graph to minimize inter-partition edges while maintaining balanced shard weights. Considering the entire transaction state network, the METIS algorithm achieves GL but inevitably incurs a state graph storage overhead of $O(y^2)$. However, since the METIS algorithm does not quantify workload in detail, the account graph partitioning relies solely on edge weights and therefore does not satisfy WB. Moreover, the METIS algorithm cannot adapt to fluctuations in network load; therefore, it disregards AP. As the number of shards increases from 4 to 60, the CTR stabilizes at 20-25%.

Nguyen *et al.* [19] proposed a blockchain sharding optimization framework called OptChain, which aims to minimize the proportion of cross-shard transactions through a lightweight dynamic transaction allocation strategy. OptChain models the UTXO transaction flow as a TaN, where each transaction is represented as a node in an online Directed Acyclic Graph (DAG). It then formulates the transaction allocation problem as an online graph partitioning problem considering temporal balance. Therefore, OptChain supports GL. In addition, OptChain introduces a Temporal Fitness Scoring mechanism to evaluate the suitability of placing new transactions in each shard. Specifically, Temporal Fitness comprises two subscores: Transaction to Shard (T2S) and Latency to Shard (L2S). The T2S score is determined using the PageRank algorithm [77] to evaluate the compatibility of new transactions with each shard, while the L2S score estimates the confirmation latency of new transactions in each shard based on the probability distribution of the shard's communication and verification time. Finally, using the formula $T2S - 0.01 * L2S$, OptChain calculates the Temporal Fitness score, and the shard with the highest score is selected as the optimal placement for a new transaction. The Temporal Fitness Scoring mechanism determines the best placement of transaction shards based on the real-time load of the transaction network, thereby ensuring dynamic workload balance. Obviously, OptChain satisfies the requirements of AP and WB. Nevertheless, maintaining the TaN structure incurs an unavoidable $O(x^2)$ storage overhead when its size is x. In the Ethereum dataset containing 3 million transactions, as the number of shards increases from 4 to 64, the CTR rises from 11.26% to 36.68%.

Li *et al.* [70] proposed a community-aware account allocation strategy called the Constrained Label Propagation Algorithm (CLPA). The core idea of CLPA is to propagate labels between nodes based on their connection patterns, thereby identifying tightly connected communities with constrained sizes. During initialization, CLPA assigns an initial label to each account based on its current shard ID. Subsequently, each account iterates through its neighboring accounts, calculates a score using a scoring function, and updates its label accordingly. The scoring function considers both community size and the strength of connections between accounts, and introduces a penalty parameter to adjust the priority of the optimization goals. Therefore, WB is satisfied. To further support the proposed account partitioning mechanism, Li *et al.* designed an elastic sharding protocol called Transformers. A key innovation of Transformers is the introduction of a Main Shard (M-shard), which is responsible for processing local transactions and constructing an account graph. The M-shard then executes the CLPA algorithm to achieve a community-aware partitioning of the account graph. Ultimately, each Working Shard (W-shard) relocates accounts based on the partitioning results and updates its local ledger state. Therefore, CLPA satisfies GL, but preserving the community structure incurs an unavoidable $O(y^2)$ storage overhead. Additionally,

CLPA fails to account for AP, as it does not dynamically adapt to load fluctuations within an epoch. In the Ethereum dataset with three million transactions, when the penalty parameter is set to 0.5, the number of shards is fixed at eight, and the epoch length is approximately 100 seconds, resulting in a CTR of about 40%.

Ren *et al.* [71] proposed the Root Graph Placement (RGP) algorithm, aiming to reduce the number of cross-shard transactions by placing ancestor transactions and their associated future transactions in the same shard. RGP constructs a DAG rooted at the new transaction using Breadth-First Search (BFS) [78], which encompasses the new transaction along with its ancestors within k levels. Next, the algorithm counts the number of fully spent and partially spent ancestors in each shard, where parameter a differentiates their respective weights. The cost score and load score for each shard are then calculated considering both the number of ancestor transactions and the overall transaction volume within each shard. The cost score is multiplied by the load score to compute a final placement score for each shard, with the RGP algorithm assigning the new transaction to the shard with the highest score. If all shard scores are zero, the new transaction is placed in the shard containing the least number of ongoing transactions. The cost score and load score measure the fitness of transaction allocation from both a historical transaction and a workload perspective. Therefore, RGP satisfies the requirements of GL and WB, while AP is overlooked. Moreover, RGP only needs to consider the recent k levels of ancestors without analyzing the entire transaction history, thus the storage overhead is only $O(kx)$. When the level of ancestors is two, the CTR is approximately 20% across shard configurations ranging from 4 to 128.

Zhang *et al.* [65] introduced TxAllo, a deterministic and efficient transaction allocation mechanism designed for the dynamic allocation of accounts and their associated transactions. The TxAllo algorithm consists of two sub-algorithms: a global optimization algorithm called G-TxAllo and an adaptive adjustment algorithm called A-TxAllo. G-TxAllo optimizes the transaction network globally using historical data, while A-TxAllo efficiently updates allocations based on previous results and real-time transaction data. Thus, TxAllo achieves GL. Specifically, the G-TxAllo algorithm operates in two phases: initialization and optimization. In the initialization phase, G-TxAllo employs the Louvain [79] extension algorithm to perform community detection on the account graph to derive an initial account-to-shard mapping. During the optimization phase, the G-TxAllo algorithm iterates through each account, evaluating throughput improvements resulting from reassigning accounts across communities, and continues until the gain falls below a predefined convergence threshold. Conversely, A-TxAllo quickly adapts to data changes by computing throughput variations at the community level based solely on the previous allocation results and newly submitted transaction data. Furthermore, both sub-algorithms of TxAllo enforce workload constraints based on each shard's processing capacity, λ, thereby meeting the requirement of WB. However, TxAllo does not fully meet AP because it fails to adapt its allocation policy in response to real-time shard load fluctuations. Additionally, the introduction of the community detection algorithm inevitably incurs an $O(y^2)$ storage overhead. When the transaction size reaches 300,000, TxAllo achieves approximately a 12% CTR even with 60 shards.

Jia *et al.* [72] proposed a protocol called Estuary, which employs a multi-level state model and a state-split aggregation mechanism to minimize cross-shard transactions. Within the multi-level state model, state units are divided into primary and secondary states. The primary state facilitates transactions between users, while the secondary state is limited to receiving state units from other users or being converted into a primary state. The state split aggregation mechanism allows users to split and aggregate state units between shards, enabling dynamic state distribution adjustments based on user transaction demands. Additionally, Estuary introduces the Community Overlap Propagation Algorithm (COPRAS) to optimize user state distribution. COPRAS defines the belonging coefficient and the workload index $\Delta(D)$. The former denotes the proportion of a user's state distribution across different shards, while the latter measures the largest workload deviation of shard from the system's average workload. During the iterative process, COPRAS dynamically calculates and updates the user's belonging coefficient in each shard based on states and the belonging coefficients of neighboring users. Finally, based on the updated belonging coefficient, COPRAS dynamically reallocates user states across shards to reduce cross-shard transactions and minimize the imbalance index $\Delta(D)$. Therefore, COPRAS satisfies WB. Since COPRAS traverses the entire transaction network $G(V, E)$ to update each user's belonging coefficient, the scheme meets the requirement of GL but incurs a storage overhead of $O(y^2)$. However, COPRAS does not adapt to network load fluctuations, so AP is ignored. After processing 47 million one-to-one Bitcoin transactions, when the number of shards is incrementally increased from 2 to 64, the CTR remains generally below 6.4%.

2) Polling-based Schemes: Krol *et al.* [73] introduced Shard Scheduler, a mechanism for object placement and migration scheduling aimed at improving overall system throughput. Shard Scheduler supports complex multi-account transactions arising from smart contracts by determining the optimal placement of accounts while minimizing migration costs. It maintains two key data structures: an alignment vector, which represents the total transaction cost associated with an account in a specific shard, and a load metric for each shard. To allocate new accounts, Shard Scheduler selects a main shard based on the criterion of choosing the least-loaded shard and assigns the new account accordingly. Simultaneously, for pre-existing accounts involved in the transaction, Shard Scheduler calculates the migration cost of each account based on their alignment vectors and determines whether migration to the main shard is beneficial. Thus, Shard Scheduler adapts to network load fluctuations, satisfying AP. Additionally, it incorporates an economic incentive mechanism to encourage miners to execute migrations that enhance overall system throughput rather than prioritizing the interests of a single shard. The Shard Scheduler scheme satisfies WB while maintaining an $O(y)$ storage overhead, where y represents the number of

active accounts. However, since historical account interaction data is not considered, Shard Scheduler does not satisfy GL. When the number of shards is 16 and the migration cost is 10, the CTR is approximately 60%.

Li *et al.* [74] introduced LB-Chain, a smart account allocation and migration scheme designed to balance transaction loads across shards while minimizing account migrations to reduce overhead. The account allocation process consists of two components: transaction prediction and an account allocation algorithm. In the prediction phase, LB-Chain employs Long Short-Term Memory (LSTM) [80] technology to periodically forecast the future transaction volume of "hot accounts" based on historical transactions and pending transactions generated by these accounts. However, as it does not take account interaction into consideration, LB-Chain cannot effectively address the cross-shard transaction optimization problem and therefore does not meet the requirements of GL. Based on the transaction prediction results, LB-Chain employs a heuristic approach to move "hot accounts" from overloaded shards to the least-loaded shard while keeping the shard assignments of "non-hot accounts" unchanged. This redistribution reduces transaction imbalances among shards, thereby ensuring WB. Once the allocation scheme is finalized, the system migrates the account state and its pending transactions from the source shard to the target shard. Additionally, the system incorporates measures such as ensuring intra-shard consensus, prioritizing transaction execution, handling transaction queue migration, and postponing the verification of new transactions to maintain migration safety and enhance migration efficiency. Overall, LB-Chain effectively adapts to network load fluctuations, meeting the requirements of AP. Since LB-Chain is primarily optimized for workload balance and does not provide details on cross-shard transaction handling, its CTR cannot be evaluated.

3) Transaction Characteristics-based Schemes: To minimize cross-shard communication overhead, Tao *et al.* [18] proposed a sharding approach based on smart contracts. The method groups users interacting with a single smart contract into a shard, enabling independent verification and confirmation of transactions within each shard. For users participating in multiple smart contracts, their transactions are placed in a designated shard, MaxShard. MaxShard records all transactions in the system, and miners maintain a local call graph [81] between smart contracts and users, eliminating the need for remote access to the entire transaction history. Consequently, the approach does not satisfy GL, but it incurs only $O(1)$ storage overhead, indicating that storage requirements remain constant regardless of transaction volume. To prevent empty block generation and enhance throughput, Tao *et al.* proposed a dynamic inter-shard merging algorithm and an intra-shard transaction selection algorithm. The former encourages small shards to merge into larger shards based on cooperative game theory [82] to optimize computational resources, while the latter improves the efficiency of larger shards by allowing miners to choose distinct transaction sets. Through evolutionary stable strategy [83], Tao *et al.* further demonstrate that the proposed inter-shard merging and intra-shard transaction selection algorithms converge to Nash equilibrium. Since the scheme can adapt shard sizes dynamically in response to transaction state changes, it satisfies AP and WB. Since the proposed scheme is designed exclusively for smart contracts in which users participate, the CTR is 0%. Nevertheless, its applicability remains limited to specific smart contract interactions.

Tao *et al.* [75] proposed a transaction distribution mechanism aiming to enhance the throughput of the relay chain [84]. The core idea of Sliver is to assign cross-chain transactions with dependencies to a single shard, thereby entirely eliminating the need for cross-shard interactions. Specifically, Sliver assigns the same identifier to the registration, recording, and completion of a transaction, ensuring that miners within a shard do not need to verify cross-chain transactions from other shards. To address the load imbalance of the relay chain, Sliver formulates relay transaction allocation as an integer optimization problem. The objective is to incrementally minimize the maximum shard workload while satisfying all cross-chain transaction dependencies. After transforming the objective function into a separable convex function with respect to the allocation variables, Sliver then reformulates the integer optimization problem into an equivalent linear programming problem using λ-techniques. Finally, this problem is solved efficiently by a linear programming solver [85], [86]. Therefore, Sliver satisfies the requirement of WB. As Sliver exclusively records cross-chain transaction data, it does not satisfy GL, and its storage overhead is $O(t)$. Additionally, Sliver does not account for short-term fluctuations in shard workload, thereby disregarding AP. Since the proposed scheme is solely for cross-chain transactions, the CTR is 0% for this specific scenario, yet its applicability remains restricted to cross-chain transactions.

B. Review on Cross-shard Transaction Processing Mechanisms

In this subsection, we review 19 articles related to cross-shard transaction processing mechanisms according to their categories. In addition, we evaluated them using general evaluation metrics, as shown in Table V. Notably, UTXO-based transaction processing mechanisms support ME, but do not support CE. Unless otherwise specified, we assume that the BFT consensus protocol can tolerate up to 1/3 Byzantine faults, while the PoW consensus protocol can tolerate up to 1/2 faulty nodes.

1) 2PC Protocol: To handle UTXO-based cross-shard transactions, OmniLedger [11] proposes Atomix, a client-driven 2PC protocol that ensures transactions are atomically committed or eventually aborted. Each input shard tolerates Byzantine faults up to 1/3 of its nodes, generating a proof of acceptance or proof of rejection via the ByzCoinX [92] consensus protocol while locking the funds. The client then gathers sufficient proof to either commit or abort the transaction and subsequently reclaims the locked funds. Therefore, the protocol can satisfy AM. However, if the client crashes indefinitely, the locked funds may become permanently inaccessible, thereby violating the requirement of LN. Furthermore, OmniLedger enforces serialized transaction execution

TABLE V
SUMMARY AND COMPARISON OF CROSS-SHARD TRANSACTION PROCESSING MECHANISMS

Category	Ref	TM	AM	LN	IL	FAR	RAR	CAR	IN	CE	ME	PL	FT	CO
2PCP	OmniLedger [11]	UTXO	●	○	●	●	○	○	○	○	●	○	1/3	$O(m+n)$
	SGX-Shard [12]	General	●	●	●	●	○	○	○	●	●	○	1/2	$O(m^2+n)$
	Instachain [87]	UTXO	○	●	○	●	●	●	○	○	●	○	1/3	$O(m+n)$
	Chainspace [88]	A/B	●	●	●	●	○	●	○	●	●	○	1/3	$O(m^2+n^2)$
	Byzcuit [58]	A/B	●	●	●	●	●	○	○	●	●	○	1/3	$O(m^2+n)$
	SSHC [20]	UTXO	●	●	○	●	●	○	○	○	●	●	1/3	$O(m+n)$
	CHERUBIM [62]	General	●	●	○	●	○	○	○	○	●	●	1/3	$O(m+n)$
CSBFTM	Sharper [66]	A/B	●	●	●	●	●	●	○	○	●	○	1/3	$O(m^2n^2)$
	FS [21]	UTXO	●	●	●	●	●	●	○	○	●	●	1/3	$O(m+n)$
	Cochain [89]	A/B	●	●	●	●	●	●	○	○	●	●	1/3	$O(m^2+n^2)$
CSTSM	RapidChain [14]	UTXO	●	●	○	●	○	○	○	○	●	●	1/2	$O(m^2+m\log n)$
RTM	Monoxide [15]	A/B	●	●	○	○	○	●	●	●	○	○	1/2	$O(m+n)$
	BrokerChain [16], [17]	A/B	●	●	○	●	●	●	●	○	○	○	1/3	$O(m^2+n)$
	X-shard [53]	A/B	●	●	●	●	●	●	○	○	●	●	1/3	$O(m^2+n)$
DM	Benzene [67]	A/B	●	●	○	●	●	○	○	○	○	○	1/2	$O(m+n)$
	Jenga [90]	A/B	●	●	○	●	●	●	●	●	○	○	1/3	$O(m^2+1)$
MLSM	Pyramid [91]	A/B	●	●	●	●	●	○	○	●	●	●	1/3	$O(m+n)$
	Prophet [54]	A/B	●	●	●	●	●	●	●	●	○	●	1/3	$O(m+n^2)$
	DC_chain [68]	A/B	●	●	●	●	●	●	○	●	●	○	1/3	$O(m+n)$

●: Meet the criterion; ○: Do not meet; '-': Not Available; UTXO: UTXO model; A/B: Account/Balance model; m: The size of each shard; n: The number of shards involved in the transaction. 2PCP: Two-phase Commit Protocol; RTM: Relay Transaction mechanism; CSBFTM: Cross-Shard BFT Mechanism; CSTSM: Cross-shard Transaction Splitting Mechanism; DM: Decoupling Mechanism; MLSM: Multi-layer Sharding Mechanism; Ref: Reference; TM: Transaction Model; AM: Atomicity; LN: Liveness; IL: Isolation; FAR: Fork Attack Resistance; RAR: Replay Attack Resistance; CAR: Censorship Attack Resistance; CE: Contract Enablement; ME: MIMO Enablement; IN: Incentive; FT: Fault Tolerance; PL: Parallelism; CO: Communication Overhead. FAR, RAR, and CAR indicate whether a scheme provides countermeasures against FA, RA, and CA, respectively.

by constructing a block DAG, satisfying IL, but it does not support PL. Since OmniLedger lacks an incentive mechanism for participating entities, IN is overlooked. OmniLedger employs Collective Signatures (CoSi) [93] to enhance transaction security, serving as a defensive countermeasure against FA, thereby satisfying the requirement of FAR. However, OmniLedger is vulnerable to CA and RA due to inadequate supervision of the leader's behavior within shards and the potential for attackers to replay response messages from a shard to previous transactions within a protocol instance. Consequently, RAR and CAR remain unfulfilled. In an experimental setup comprising 1,800 nodes and 25 shards, with a malicious node proportion of 12.5%, OmniLedger achieves a throughput of 13,000 TPS and a latency as low as 8.04 seconds. With the introduction of CoSi signatures and a centralized coordinator, when a transaction involves n shards with a shard size of m, the cross-shard communication overhead is only $O(m+n)$.

SGX-Shard [12] introduces a cross-shard consensus protocol that ensures the correct execution of transactions even in the presence of malicious transaction coordinators. The consensus protocol designates a reference shard as a centralized coordinator, which executes BFT internally to implement a 2PC state machine and mandates the involved shards to send response messages. Funds are unlocked for submission only after collecting a sufficient number of responses from the participating shards. Therefore, the protocol satisfies AM. Since the reference committee is designed with high security, ensuring continuous processing of client requests without the risk of indefinite blocking, LN is guaranteed. Moreover, SGX-Shard employs 2PL for concurrency control, which may constrain concurrency efficiency under high workloads. As a result, SGX-Shard satisfies IL but not PL. From a security perspective, the shards use TEE (e.g., Intel SGX [94]) to enhance the Practical Byzantine Fault Tolerance (PBFT) protocol [42], [43] to achieve intra-shard consensus with no more than 1/2 Byzantine fault tolerance, so it can support FAR. However, the lack of a verification mechanism for shard response messages and cross-shard supervision of malicious behavior prevents SGX-Shard from mitigating RA and CA. Consequently, RAR and CAR remain unfulfilled. In terms of applicability, SGX-Shard supports CE and ME due to its emphasis on accommodating general blockchain workloads. Additionally, SGX-Shard does not support IN due to insufficient consideration of its design and implementation. In an experimental deployment on the Google Cloud Platform with 972 nodes and 36 shards, under a 12.5% adversary model, SGX-Shard achieves a throughput of 3,000 TPS with an average latency of 80 seconds. When a transaction involves n shards with each

shard size of m, the cross-shard communication overhead is $O(m^2+n)$.

Instachain [87] adopts a stateless blockchain model within shards and integrates a novel cross-shard verification technique, enabling efficient processing of UTXO-based transactions. Instachain employs a client-driven 2PC protocol for cross-shard transaction verification. Each shard executes synchronous HotStuff [95] with 1/3 Byzantine fault tolerance to process cross-shard transactions sequentially, generating a Proof-of-Inclusion (POI) that confirms the removal of funds from the UTXO pool of the input shard. The client subsequently collects the POI to coordinate cross-shard transactions, incurring a cross-shard communication overhead of $O(m+n)$. If an input asset is invalid, the protocol permits the submission of other valid input assets. As a result, Instachain fails to satisfy AM. To prevent transaction blocking, each node employs a secure shard detector to monitor shard status. When a shard loses liveness, the detector initiates a shutdown request to the reference shard. The reference shard reorganizes nodes and transactions upon receiving a sufficient number of shutdown requests, ensuring transaction transfer to an active shard. Thus, Instachain satisfies LN and CAR. However, Instachain fails to satisfy IL and PL due to the absence of parallel execution and concurrency control mechanisms. Moreover, Instachain maintains a POI accumulator in each shard and mandates clients to provide non-membership proofs for cross-shard transactions, which consequently disregards FAR and RAR. Since Instachain lacks a mechanism to incentivize nodes for cross-shard transactions or ensure its security, IN is not satisfied. As Instachain lacks specific experimental details, its throughput and latency remain unevaluated.

Chainspace [88] employs S-BAC, a distributed 2PC protocol integrating Byzantine agreement and atomic commit. In this protocol, all input shards act as coordinators, each generating an availability certificate via a 1/3 Byzantine fault-tolerant protocol and coordinating cross-shard transactions through certificate exchange. A transaction is submitted only upon the receipt of availability certificates from all participating shards. Since each shard executes the BFT protocol for consensus, this approach mitigates single points of failure and prevents transaction blocking. Thus, S-BAC satisfies the requirements of AM and LN. Since S-BAC effectively implements a variant of OCC, it resolves conflicts by processing a transaction and aborting others. As a result, S-BAC satisfies IL but not PL. Furthermore, Chainspace is auditable, allowing users to track transactions and identify malicious nodes. Thus, S-BAC supports FAR and CAR but does not support RAR, as it remains vulnerable to the replay of response messages. Although Chainspace incentivizes nodes via CSCoin contracts, this incentive does not apply to cross-shard transactions and thus fails to support IN. In an experiment with 15 shards, each containing 4 nodes, Chainspace achieves a throughput of 350 TPS and a latency of 210 ms. Input shards exchange availability certificates to coordinate cross-shard transactions, leading to a communication overhead of $O(m^2+n^2)$. Regarding applicability, Chainspace is tailored for smart contracts and extends to multi-object transactions, satisfying CE and ME.

Byzcuit [58] is a fork of Chainspace that assigns a shard as the cross-shard transaction manager. This centralized coordination reduces cross-shard communication overhead to $O(m^2+n)$ among n shards, each containing m nodes. Since its experimental setup is based on Chainspace, it inherits the same satisfaction of AM, LN, and IL. However, as it does not enhance parallelism in Chainspace, it remains incompatible with PL. Byzcuit is also expected to satisfy CE and ME in terms of applicability, similar to Chainspace. Furthermore, the protocol mandates that the output shard implicitly consume the virtual input to transition into an input shard, requiring each shard to participate in the first phase of 2PC and execute BFT consensus with 1/3 Byzantine fault tolerance. This mechanism satisfies FAR and RAR, though it lacks a mechanism to accommodate CAR. Byzcuit achieves a throughput of 1,550 TPS in an experimental setup with 96 nodes and 10 shards.

SSHC [20] implements a responsive cross-shard transaction batch-processing scheme to improve the efficiency of cross-shard transactions in the UTXO model. Each input shard aggregates multiple transaction inputs and applies Pipelined Byzantine Fault Tolerance (PLBFT) to generate an availability certificate, tolerating up to 1/3 Byzantine faults. The input shard then aggregates the availability certificates into a Merkle tree [96] and commits to the root using a threshold signature. Shards validate each other by verifying the availability certificates received from other input shards. Once all input shards have issued their availability certificates, the output shard can directly verify the state of each input using the Merkle tree without invoking the BFT protocol again. By analyzing the cross-shard processing mechanism of SSHC, it is actually a distributed 2PC protocol and therefore can satisfy AM and LN. Due to the incorporation of batch processing, SSHC satisfies the requirement of PL. From a security perspective, each input shard employs PLBFT consensus and binds its input state to the corresponding transaction ID when transmitting availability certificates, thereby supporting FAR and RAR. However, as individual shards do not maintain a global perspective of cross-shard transactions, SSHC remains vulnerable to CA. Consequently, CAR remains unaddressed. Additionally, SSHC does not support IN due to the absence of a well-defined incentive mechanism. As SSHC lacks specific experimental details, its throughput and latency remain unevaluated. Compared to other distributed 2PC protocols, SSHC incurs a cross-shard communication overhead of $O(m+n)$.

Existing 2PC protocols process cross-shard transactions sequentially, resulting in significant system overhead and reduced throughput. CHERUBIM [62] introduces a general pipeline 2PC framework, P-2PC, to optimize the 2PC protocol through the reuse of cross-shard certificates. With the integration of intra-shard BFT, CHERUBIM extends P-2PC to 4P-2PC, enabling pipelined processing both within and across shards. The output shard first sorts transactions and forms a batch for the input shard. The input shard then verifies the availability of the transaction inputs in the current batch and the validity of the previous batch using PLBFT. Upon receiving the input shard's availability certificate, the output shard aggregates it using the Aggregated Multi-signature (A-Msign) protocol and returns it to the input shard for final confirmation. Thus, it satisfies the requirement of AM. Ad-

ditionally, each shard functions as a coordinator by executing the BFT consensus protocol, eliminating the risk of a single point of failure. Consequently, it satisfies LN. An analysis of the cross-shard consensus process reveals that 4P-2PC can process four transaction batches simultaneously within a single voting round, thus satisfying PL. However, due to the lack of concurrency control, it does not satisfy IL. From a security perspective, as each shard utilizes BFT consensus, it satisfies FAR. However, it lacks countermeasures to mitigate RAR and CAR, as it fails to incorporate a global perspective and response binding mechanisms. This vulnerability arises from insufficient oversight of the leader's behavior within the shard and the risk of an attacker replaying the response message in a different protocol instance. Additionally, IN remains unexplored. Regarding applicability, CHERUBIM supports two trading models but does not specify support for smart contracts, thereby satisfying ME but not CE. The integration of pipelining and multi-signature protocols significantly reduces communication overhead, resulting in a cross-shard communication complexity of $O(m + n)$. With 20 shards, each containing 60 nodes, the system achieves a throughput of 49,700 TPS and a transaction confirmation latency of 3.45 seconds.

2) Cross-Shard BFT Mechanisms: Sharper [66] employs DAG techniques to preserve a global transaction sequence and introduces a flat consensus protocol to coordinate cross-shard transaction ordering. The cross-shard consensus process in the presence of Byzantine nodes comprises three stages: propose, accept, and commit. As illustrated in Fig. 9, a client first submits a cross-shard transaction request to any node in the relevant shard. The receiving node assumes the role of the primary node, initiating the consensus protocol across all relevant shards. The primary node broadcasts a proposal to all nodes, and each node then collects at least $2f + 1$ matching messages from each shard before advancing to the next phase. Once a sufficient number of commit messages are collected, the node executes and finalizes the cross-shard transaction, satisfying AM. An analysis of this consensus process reveals that Sharper integrates the Byzantine consensus protocol into cross-shard consensus, triggering a view change upon node failure or timeout. Additionally, Sharper enforces a total order among transactions accessing the same data. Consequently, it satisfies the requirements of LN and IL. However, PL is not addressed. From a security perspective, the consensus mechanism of Sharper functions as a Byzantine consensus protocol encompassing all participating nodes, providing resilience against three potential attacks. Consequently, FAR, RAR and CAR are satisfied. Since Sharper lacks incentives for node participation in cross-shard transactions, it does not fulfill the requirement of IN. With a setup of 4 shards and 4 nodes, where all transactions are cross-shard, Sharper processes 7,500 transactions with a latency of 700 ms. The cross-shard communication complexity is $O(m^2n^2)$, where m denotes the shard size and n denotes the number of nodes. As Sharper exclusively supports the account model and lacks references to smart contract compatibility, it does not satisfy CE or ME.

FS [21] proposes the parallel Cross-Shard BFT protocol, CSBFT, enabling each node to function as both a leader and a backup across multiple concurrent CSBFT instances. During both the cross-preparation and cross-commitment phases, each shard leader initiates proposals and gathers at least $2f+1$ votes within its shard before forwarding them to the coordinator shard leader. The coordinator shard leader aggregates the received messages into a confirmation message and disseminates it to all shard leaders. Subsequently, the shard leader broadcasts the confirm message within the shard. Once honest nodes verify it, the transaction is deemed committed. CSBFT employs a two-layer HotStuff protocol, ensuring consistency and mitigating single points of failure, thereby satisfying AM, LN, and IL. Furthermore, since a node can participate in multiple consensus rounds concurrently, CSBFT functions as a variant of the pipeline mechanism, satisfying PL. If a shard leader exhibits malicious behavior, other shards swiftly generate and transmit cross-shard view change messages to the target shard, ensuring the cross-shard view mechanism effectively mitigates FA and CA. Thus, it satisfies FAR and CAR. However, CSBFT does not address IN due to the absence of incentives. Under an experimental setup of 128 nodes and 36 shards, with an average of 2 input shards, FS achieves a throughput of approximately 32,000 TPS and a latency of 2.2 seconds. The leader aggregates and distributes messages twice, yielding a cross-shard communication complexity of $O(m + n)$.

In sharded blockchain systems, smaller shards are susceptible to corruption, compromising overall system security. As a result, existing studies favor larger shards, which substantially limit transaction concurrency in large-scale blockchain sharding systems. Cochain [89] introduces the Consensus on Consumes (CoC) protocol, enabling multiple shards to form a CoC group that oversees smaller member shards. If a member shard is found to be corrupted, CoC swiftly replaces it with another, ensuring each shard maintains a 2/3 fault tolerance rate. To validate intra-shard consensus results, each member shard executes transactions via the PFBT protocol and transmits the results to other shards using Cross-Shard BFT. A cross-shard transaction is relayed to the target shard only after the source shard's block is confirmed by the CoC protocol, thereby satisfying AM. The CoC protocol ensures isolation to prevent transaction conflicts, thereby supporting IL. Furthermore, Cochain improves processing efficiency through a pipeline mechanism, enabling shards to optimistically generate new blocks while awaiting consensus verification, thus satisfying LN and PL. From a security perspective, the CoC group verifies intra-shard consensus results, effectively mitigating three distinct types of potential attacks. Thus, FAR, RAR and CAR are supported. However, CoC does not address IN. Under an experimental setup with 6,100 nodes and 10 shards, where each block accommodates up to 4,096 transactions, the system achieves a throughput of 65,066 TPS and an approximate latency of 40 seconds. The cross-shard communication overhead is $O(m^2 + n^2)$. Cochain supports multi-asset transactions but does not mention smart contract functionality, satisfying ME but not CE.

3) Cross-shard Transaction Splitting Mechanisms: To reduce the overhead introduced by centralized 2PC on clients,

RapidChain [6] employs a transaction-splitting mechanism to handle cross-shard transactions. The output shard splits cross-shard transactions into multiple sub-transactions, generating several UTXOs equivalent in number to those of the input shard designated for expenditure. Upon receiving a sub-transaction, each input shard verifies it using intra-shard BFT protocol [97] with 50% fault tolerance in a synchronous network, and then returns the result to the output shard. If one sub-transaction fails and the others succeed, the sharded blockchain uses the funds in future transactions by notifying the owner of the funds, effectively achieving the same outcome as a transaction rollback. Therefore, Rapidchain satisfies AM. Besides, the output committee uses batching verification requests at each round, thus PL is supported. However, since transactions are split into multiple sub-transactions, if one of these sub-transactions fails while another conflicts with concurrent transactions, the entire transaction may fail simultaneously. Therefore, the mechanism cannot satisfy IL. Regarding transaction activity, the client solely initiates transactions without participating in coordination, thereby preventing funds from being locked indefinitely and satisfying LN. From a security perspective, RapidChain supports FAR. However, malicious nodes can still replay response messages and withhold acknowledgments to launch attacks, rendering RapidChain incapable of satisfying RAR and CAR. Since RapidChain lacks incentives for nodes to participate in cross-shard transactions, it does not support IN. Under an experimental setup with 4,000 nodes and 250 shards, where each transaction block contains 4,096 transactions, the system achieves a throughput of 7,384 TPS with a latency of 8.84 seconds. With the implementation of the Kademlia routing mechanism [98], the cross-shard communication overhead is only $O(m^2 + m \log n)$.

4) Relay Transaction Mechanisms: Monoxide [15] introduces an eventual atomicity scheme to efficiently manage cross-shard transactions, ensuring correctness and robustness in asynchronously operating shards. As illustrated in Fig. 11(a), Monoxide decomposes cross-shard transactions into multiple steps, each involving a single shard, and incentivizes miners to execute these steps via relay transactions. This mechanism guarantees the eventual completion of all operations, achieving the correct end state and thereby satisfying AM. Eventual atomicity enables transactions to interleave asynchronously and without locks, ensuring shard concurrency and full utilization. Thus, Monoxide satisfies LN. However, neither IL nor PL is explored. The Chu-ku-nu mechanism allows miners to generate multiple blocks concurrently across different zones using a single PoW consensus protocol with 1/2 Byzantine fault tolerance. Monoxide ensures transaction security by requiring the source shard's block to confirm the initial transaction and the target shard's first block to confirm the relay transaction. There remains a certain probability that the ledger of the source exchange will be replaced by another ledger. However, unlike immediate sharded blockchains, the eventual sharded blockchain relies on probabilistic block confirmations within each shard. There is a probability that the source shard ledger may be replaced by an alternative ledger. Security analysis indicates that Monoxide fails to support FAR and CAR, but supports RAR. Monoxide introduces fee splitting for cross-shard transactions, incentivizing the relayed step of transaction processing and thereby supporting IN. Furthermore, Monoxide expands its support from asset transactions to smart contracts, thereby satisfying CE but not ME. In a setup with 2,048 shards and 24 nodes per shard, Monoxide achieves a throughput of 11,694 TPS with a latency of 13–21 seconds. When n shards, each containing m nodes are involved, the cross-shard communication overhead is $O(m + n)$.

Advanced blockchain sharding solutions, such as Monoxide, experience issues of load imbalance and an elevated proportion of cross-shard transactions. Brokerchain [16], [17] employs fine-grained status partitioning and account segmentation, integrating them with broker accounts to facilitate cross-shard transactions. As illustrated in Fig. 11(b), user accounts pledge a portion of their funds to become brokers, whose status is partitioned and distributed across multiple shards. During cross-shard transaction consensus, the broker divides the transaction into two parts, executing one on the source shard and the other on the target shard. The broker creates the latter part only after the source shard confirms the former part. Subsequently, the target account validates the broker's transaction, prompting the source shard to release the locked funds. Therefore, Brokerchain can satisfy AM. Moreover, if the transaction remains unconfirmed within the designated token-lock period, the source shard refunds the token to the sender, clearly satisfying LN. The absence of concurrency control and parallelism techniques results in Brokerchain not supporting PL and IL. Regarding security, each shard validates transactions through the PBFT protocol, ensuring consensus with a fault tolerance threshold of 1/3. Additionally, each account in Brokerchain maintains a nonce counter, and cross-shard transactions incorporate a token-lock duration. These mechanisms provide partial protection against three potential attacks. Thus, FAR, RAR and CAR are supported. Brokerchain incorporates an incentive mechanism to encourage users to act as brokers, thereby satisfying IN. The performance of BrokerChain was evaluated in an environment with 112 nodes and 16 shards, featuring an 8-second block interval and a block capacity of 500 transactions. The system maintains a fixed transaction arrival rate of 500 and utilizes 40 broker accounts. BrokerChain attains an average throughput of 352 TPS with an acknowledgment latency of 275.94 seconds. The cross-shard communication complexity is $O(m^2 + n)$, where n denotes the number of shards and m represents the size of each shard. Finally, as Brokerchain solely processes asset transactions under the Account/Balance model, neither CE nor ME is supported.

X-shard [53] adopts an optimistic approach to concurrently process cross-shard transactions as sub-transactions within input shards, thereby improving transaction processing efficiency. It establishes a gateway account in each shard to facilitate transactions with other shards. The input shard decomposes a cross-shard transaction into multiple sub-transactions, jointly signed using a threshold signature scheme. Subsequently, the funds are transferred to the gateway account of the designated output shard. If all sub-transactions achieve intra-shard PBFT consensus, the output shard incorporates the com-

plete cross-shard transaction into the blockchain. Conversely, if any sub-transaction fails validation, a rollback transaction is generated to revert the processed sub-transaction. Thus, X-shard satisfies AM. Furthermore, X-shard implements a timestamp-based verification period mechanism, mandating each shard to complete verification within a predefined time frame upon receiving cross-shard transactions and their corresponding sub-transactions. Hence, LN is satisfied. Analyzing X-shard's handling of concurrent transactions, we find that it satisfies IL, as it employs OCC to process MIMO-type cross-shard transactions. Additionally, the batch signing mechanism of threshold signatures allows X-shard to support PL. Regarding security, X-shard incorporates timestamps and threshold signatures, effectively mitigating three potential attacks. Therefore, FAR, RAR and CAR are satisfied. However, IN remains unexplored. In an experimental setup with 16 shards and 10 nodes per shard, a transaction arrival rate of 3,000 TPS, and a maximum block size of 100 transactions, X-shard attains a peak throughput of 1,200 TPS while maintaining an optimal communication overhead of $O(m^2 + n)$. X-shard exclusively supports asset transactions under the MIMO model, thereby satisfying ME but not CE.

5) Decoupling Mechanisms: Benzene [67] employs a dual-chain architecture to separate transaction recording from consensus execution. The Proposer chain autonomously records transactions within the shard and generates proposal blocks containing local transactions, whereas the Vote chain facilitates cross-shard collaboration and verifies proposal blocks from other shards through voting. For local transaction confirmation, miners aggregate multiple transactions into proposal blocks using PoW consensus and submit them to TEE for verification. Subsequently, miners broadcast the proposed block header and TEE proof to the entire network. Miners in other shards cast votes for valid proposal blocks, generate voting blocks, and disseminate them. Finally, the proposal block with the highest number of votes, determined by both the vote blocks and local proposal blocks, is recorded in the ledger. This voting design allows Benzene to effectively mitigate FA and RA. However, miners can still withhold blocks, and Benzene lacks a countermeasure against CA. Security analysis indicates that Benzene supports FAR and RAR, but fails to address CAR. Cross-shard transactions attain eventual atomicity via a two-phase confirmation process executed asynchronously. The consensus process functions as a TEE-assisted relay transaction mechanism. Thus, Benzene satisfies AM and LN. Furthermore, Benzene enhances the confirmation process for cross-shard transactions by employing batch processing, thereby supporting PL. However, IL and IN remain unexplored. In an experimental setup with 50 shards, Benzene achieves a throughput of 32,370 TPS with a latency of 13 seconds. The cross-shard communication overhead is $O(m + n)$ where n denotes the number of shards and m represents the size of each shard. Regarding applications, Benzene does not support CE and ME, as neither smart contracts nor MIMO are addressed.

Jenga [90] introduces a novel sharding-based approach for efficient smart contract execution, primarily requiring all state shards to retain the contract's execution logic while establishing an orthogonal execution channel for each state shard. Each channel operates orthogonally to all state shards, enabling concurrent transaction execution and interaction with state shards via orthogonal subsets, thereby eliminating cross-shard communication. Building upon these concepts, Jenga introduces a three-phase cross-shard consensus protocol. During cross-shard consensus, the state shard first determines the required transaction state and broadcasts it to the execution channel via the subgroup. Upon receiving the state, the execution channel initiates the counter and executes all relevant contracts. Once the state counter is restored, the execution channel transmits the execution result back to the state shard via the subgroup. Finally, the state shard updates the state and records the transaction on the blockchain. Thus, Jenga satisfies AM. Additionally, Jenga prevents client blocking by requiring a fee pledge, thereby satisfying LN. However IL and PL remain unexplored. Regarding security, Jenga's consensus protocol, integrated with contract state sharing and a multi-channel execution mechanism, mitigates three potential attacks. Therefore, FAR, RAR and CAR are satisfied. Furthermore, to deter malicious client behavior, the system requires clients to pledge fees before engaging in cross-shard transactions. This mechanism serves as a variant of the incentive mechanism, thereby supporting IN. In an experimental setup with 12 shards and 240 nodes per shard, each node verifies up to 4,096 transactions of 512 bytes per consensus round. With 20% of nodes designated as malicious, Jenga achieves a throughput of 4,300 TPS and a latency of 11 seconds. Additionally, as Jenga does not specify support for MIMO-type transactions, it satisfies CE but not ME.

6) Multi-layer Sharding Mechanisms: Pyramid [91] introduces a layered sharding consensus mechanism that leverages collaboration among multiple shards to ensure consistency. The layered sharding mechanism categorizes nodes into internal shards (i-shards) and bridge shards (b-shards). b-shards maintain the complete ledger of their associated i-shards and achieve cross-shard block consensus using a BFT-type protocol combined with CoSi. During each round of cross-shard transaction consensus, the b-shard generates a cross-shard block and transmits it to the relevant i-shards. The block is submitted only after receiving acceptance messages from all relevant i-shards. For cross-shard transactions spanning multiple shards, Pyramid decomposes them into multiple steps and processes them using a relay mechanism to address b-shard overlap constraints. Thus, Pyramid satisfies AM and LN. Furthermore, to prevent transaction conflicts, the i-shard randomly selects one block for acceptance while rejecting the others, thereby satisfying IL. Regarding security, Pyramid incorporates 1/3 Byzantine fault tolerance within each shard, supplemented by hierarchical verification mechanisms to confirm blocks, thereby supporting FAR and RAR. However, due to the limited overlap range of b-shards, it may still face the risk of block withholding and thus does not support CAR. However, as Pyramid lacks an incentive mechanism for nodes involved in cross-shard transactions, it does not support IN. Since Pyramid does not employ pipelining or batching to accelerate consensus, it does not fulfill PL. In an experimental setup with 3,500 nodes and 17 shards, each node verifies up to 4,096 transactions per consensus round. Pyramid attains a throughput

of 110,500 TPS with a latency of 5 seconds. For n shards, each comprising m nodes, the cross-shard communication overhead is $O(m+n)$. Since Pyramid explicitly introduces multi-step transactions covering both money transfers and smart contracts, it supports CE and ME.

To mitigate high abort rates caused by nondeterministic race conditions, PROPHET [54] guarantees conflict-free execution through a two-layer sharding architecture. This architecture operates through the collaboration and supervision of reconnaissance coalitions, a sequence shard, and executing shards. PROPHET employs a cooperative consensus mechanism comprising four phases: pre-execution, sequencing, execution, and verification. Initially, it leverages untrusted, self-organizing node coalitions from distinct shards to pre-execute cross-shard transactions, including smart contracts, to derive prerequisite ordering information. Subsequently, it constructs a trusted global order via stateless ordering and post-verification of pre-execution results. Following this order, shards systematically execute and record transactions conflict-free. At the conclusion of each round, shards broadcast their verification outcomes. Meanwhile, PROPHET employs an asynchronous correction mechanism, enabling shards to process transactions without waiting for proofs from all other shards, thereby alleviating transaction blocking. If subsequent proof deems a transaction invalid, PROPHET rolls back the transaction along with its state changes. Thus, PROPHET satisfies AM and LN. Moreover, deterministic global ordering eliminates transaction conflicts, thereby satisfying IL. Regarding security, the reconnaissance coalition, sequence shard, and execution shards all employ PBFT with 1/3 Byzantine fault tolerance to establish intra-shard consensus. Together with deterministic global ordering and the asynchronous correction mechanism, PROPHET effectively resists three potential attacks. Therefore, PROPHET satisfies the requirements of FAR, RAR, and CAR. During cross-shard consensus, PROPHET incentivizes reconnaissance coalitions by allocating transaction fees for successfully committed pre-executed transactions, thereby satisfying IN. Additionally, due to the absence of batching and pipelining, PL is not supported. In an experimental setup with 64 shards and 50 nodes per shard, the proportion of malicious nodes is fixed at 12.5%. PROPHET attained a throughput of 1,203 TPS with a latency of 2.5 seconds. With BLS signatures implemented within each shard, the cross-shard communication overhead is $O(m+n^2)$. Furthermore, since PROPHET is designed for smart contracts and does not support MIMO-type asset transactions, it satisfies CE but not ME.

DC_chain [68] introduces a Decentralized Coordinator (DC) and a robust two-layer consensus protocol to accelerate cross-shard consensus. The DC consists of primary nodes and maintains a verifiable global state database, preemptively aborting conflicting transactions and thereby satisfying IL. Moreover, the consensus protocol finalizes valid cross-shard transactions via efficient interactions between the DC and participating shards. During cross-shard consensus, the DC initially partitions cross-shard transactions, including smart contracts, into sub-transactions and establishes consensus on the sub-transaction proposals. Subsequently, DC members forward the transaction to the local shard for verification. Local shards process sub-transactions via a variant of the SBFT [99] protocol, with the leader returning the result to the client. Thus, DC_chain satisfies AM. Furthermore, after achieving consensus on a cross-shard transaction, the source and target shards can proceed with new transactions without waiting for mutual confirmation, thereby satisfying LN. However, due to the lack of batching, pipelining, and incentive mechanisms, PL and IN are not supported. By incorporating timers and BLS signatures at each stage of cross-shard consensus, along with a verifiable global database, the DC demonstrates resilience against three potential attacks. Therefore, the requirements of FAR, RAR, and CAR are satisfied. Under experimental conditions with 5 shards, each comprising 4 nodes, and a fixed transaction distribution of 10% cross-shard transactions, DC_chain achieves a throughput of 200 TPS. As DC_Chain employs a two-tier consensus protocol and integrates the linear SBFT protocol within each shard, its cross-shard communication overhead is $O(m+n)$. Regarding applicability, DC_Chain supports both CE and ME, as it accommodates smart contracts and multi-input transactions.

C. Evaluation and Discussion

Table IV presents the evaluation results from ten studies on cross-shard transaction optimization schemes. The primary objective of these schemes is to minimize cross-shard communication overhead while maintaining workload balance. Therefore, they should satisfy three key evaluation criteria: GL, AP and WB, while minimizing the CTR. Based on the literature review, six studies meet the GL, indicating that most solutions consider historical transaction patterns and implement predictive transaction allocation strategies. Regarding WB, nine studies fulfill this criterion, demonstrating a growing research emphasis on improving shard utilization to optimize workload distribution. However, only four studies support AP, with two of them relying on a polling-based mechanism. AP refers to the capacity of a transaction allocation strategy to dynamically adjust to network load fluctuations during the consensus process, thereby allowing the system to dynamically optimize transaction allocation based on real-time network conditions. By integrating the findings on workload balance, it becomes evident that most studies adjust account distribution only during the reconfiguration phase, without integrating a load-aware module into the consensus process. This limitation hinders the system's ability to efficiently handle transaction surges and network fluctuations. Therefore, future research should focus on account migration mechanisms to strengthen the system's adaptability and resilience, ultimately improving the efficiency and stability of cross-shard transaction processing.

Table V summarizes the statistical results of 19 studies on cross-shard transaction processing mechanisms. The core objective of these mechanisms is to improve the efficiency of transaction processing while ensuring transaction reliability. A review of the relevant literature reveals that AM and LN, which are critical properties to maintain consistency and facilitate fault recovery in distributed systems [45], have been implemented in nearly all existing solutions. However, only around

60% of the surveyed mechanisms satisfy the IL requirement, highlighting the absence of a universally effective approach for handling cross-shard transaction conflicts. A detailed case-based analysis reveals that existing approaches primarily rely on concurrency control strategies such as 2PL and OCC to handle transaction conflicts. Although these strategies mitigate conflicts to some extent, high locking rates and rollback frequencies may lead to decline in overall transaction processing efficiency. Additionally, our study reveals that only a few solutions, mainly those based on relay transaction mechanisms, incorporate IN. Regarding SE, the incentive mechanisms for cross-shard transactions require further exploration to ensure fair profit distribution and punish malicious nodes. Notably, only eight of the 19 studies address CE. As a key technology capable of adapting to diverse application requirements, smart contracts play a crucial role in the practical adoption of sharding schemes. Therefore, dedicated research on smart contract transaction processing mechanisms is necessary. Finally, in terms of PL, the support provided by existing solutions remains insufficient. Considering the high CO and the critical role of PL for enhancing transaction processing efficiency, future research should further explore the potential of pipelining and other acceleration technologies in cross-shard transactions to optimize overall transaction execution and communication performance.

VI. Open Issues and Future Directions

Based on the review and comparison of the literature, this section identifies several open issues and suggests future research directions to advance research on CTAOP mechanisms.

A. Efficient and Secure Account Migration

1) Open Issue: Adaptability is overlooked in most existing transaction allocation optimization schemes. To reduce cross-shard transactions while ensuring workload balance, state-of-the-art studies [65], [69], [70] periodically assign frequently interacting accounts to the same shard, but neglect to respond to network load fluctuations within an epoch. Account migration mechanisms satisfy adaptability to a certain extent, but existing mechanisms still face issues regarding security and efficiency. In the context of blockchain sharding, developing efficient and secure account migration mechanisms requires addressing numerous detailed considerations.

2) Future Research Direction: To ensure the robustness of the account migration process, three aspects need to be further studied: efficiency, security; and cost-economy. Firstly, account migration entails managing the state of an account and all its associated transactions, resulting in a substantial increase in transaction confirmation latency. Therefore, improving transaction efficiency during account migration is essential. A potential solution involves fine-tuning locking of accounts to process deposit transactions exclusively in real-time [100]. Second, simplistic account migration schemes are susceptible to various types of attacks. Thus, the migration mechanism must be meticulously designed to ensure both consistency and liveness throughout the migration process while incorporating an effective recovery mechanism for potential failures, such as migration timeout [101]. Finally, when accounts migrate between shards, state data must be synchronized, resulting in significant synchronization overhead and performance reduction in a real-world scenario. Consequently, it is imperative to design cost-effective account migration and state synchronization schemes. For example, LB-Chain and tMPT [102] provide valuable insights. LB-Chain mitigates unnecessary migrations by leveraging LSTM to predict active entities, while tMPT reduces the volume of state synchronization data by pruning inactive entities within the Merkle Patricia Trie (MPT).

B. Deterministic Parallelism

1) Open Issue: Although some cross-shard transaction processing mechanisms introduce concurrency control strategies such as 2PL and OCC to ensure isolation, researchers have observed that these solutions perform much less effectively than expected in practical applications. The poor performance can be attributed to inherent conflicts among cross-shard transactions, as well as the independent and stochastic scheduling of these transactions across different shards [73]. Specifically, each shard schedules local transactions independently and lacks a mechanism to coordinate a globally consistent transaction order. As a result, the execution order of cross-shard transactions may be inconsistent across shards, leading to unexpected and unavoidable conflicts. However, traditional concurrency control cannot guarantee global determinism and ultimately resolves conflicts through rollback or abort, sacrificing throughput and increasing latency.

2) Future Research Direction: The literature calls for the development of deterministic execution schemes to address the challenges of uncertain competition. One promising approach is to establish a transaction execution order before processing pending transactions, thereby reducing the failure rate and its associated negative impacts [73]. This approach aligns with Deterministic Concurrency Control (DCC) algorithms in distributed environments, which eliminate the need for costly commitment protocols by ensuring that different replicas independently produce identical results from the same input transactions [103]. State-of-the-art deterministic databases, such as BOHM [104], Calvin [105], and Aria [106], implement DCC through various methods. Specifically, BOHM constructs a dependency graph from a batch of input transactions by analyzing their read/write sets. Calvin involves acquiring read/write locks before executing transactions, adhering to a predefined order of input transactions. Aria runs an identical batch of transactions for each replica using a consistent database snapshot and resolves conflicts uniformly. These deterministic execution schemes offer valuable insights for addressing uncertain competition problems in cross-shard transaction processing.

C. Appropriate Reward and Punishment

1) Open Issue: We have observed that few mechanisms, such as the relay transaction mechanism, incorporate incentives, while most cross-shard transaction processing solutions overlook this aspect. Additionally, to the best of our knowledge, there is limited research on the detailed design and

theoretical analysis of reward-and-punishment mechanisms for cross-shard transactions, particularly in terms of ensuring transaction fairness and holding malicious nodes accountable. Therefore, designing an appropriate reward and punishment mechanism is crucial to incentivize honest and proactive behavior among participating nodes.

2) Future Research Direction: It is highly recommended to explore reward and punishment mechanisms to encourage participation in cross-shard transaction processing. First, sharded blockchain systems should develop fair incentive mechanisms to motivate miners to engage in cross-shard transactions. Due to the isolation of shard ledgers, most cross-shard transaction processing mechanisms require an additional round of consensus to reach agreement. This imposes high costs and security risks on the nodes involved in cross-shard transactions. For example, in the 2PC protocol, the leader is responsible for gathering, consolidating, and transmitting messages across shards. As a result, the leader incurs greater communication and computation overhead compared to other nodes and should therefore receive higher rewards [107]. Furthermore, accountability mechanisms for malicious nodes must be investigated. For instance, in the 2PC protocol, the leader of a local shard may fail to forward the availability certificate to other related shards, thereby obstructing cross-shard consensus. To address this, a reputation mechanism could be introduced to monitor node behavior. The reputation value would be calculated based on predefined algorithms that evaluate node actions. If a node exhibits untrustworthy behavior and its reputation value falls below a certain threshold, its participation rights could be revoked, or its share of profits should be reduced [108].

D. Efficient Cross-shard Contract Invocation

1) Open Issue: Previous solutions have primarily focused on simple asset transfers and non-nested smart contracts, often overlooking the challenge of efficiently supporting complex contract transactions. Executing a contract may involve accessing states from multiple contracts, making it difficult to decompose the contract into independent operations. For a complex cross-shard contract that requires a series of nested invocations to other cross-shard contracts, named chained calls[2], the confirmation latency increases linearly with the number of participating contracts [109]. Moreover, these complex cross-shard contracts may involve cyclic calls[3] to contracts managed by other shards, potentially leading to deadlocks and negatively impacting smart contract availability [110]. Therefore, designing an efficient cross-shard contract invocation mechanism is essential to address these challenges.

2) Future Research Direction: Given the complexity of smart contracts, designing a flexible and efficient cross-shard contract invocation mechanism is a promising area of research. One potential solution involves refactoring the execution-store architecture of blockchain sharding. For instance, Jenga [90] addresses the isolation among shards by coordinating state storage and execution logic, enabling complex transactions involving multiple contracts to be processed by a single execution shard in a single round. Another promising direction is to enhance the execution speed of smart contracts. Off-chain computing, leveraging the robust capabilities of Trusted Execution Environments (TEEs), offers a way to securely perform complex computations with privacy preservation off-chain while submitting only essential state information and execution results on-chain [111]. However, since TEEs are vulnerable to attacks, establishing a robust remote authentication mechanism is critical to enhancing their security and fostering trusted on-chain and off-chain interactions [112].

[2]Chained calls refer to scenarios where the execution of one smart contract triggers the invocation of another contract, which may, in turn, invoke additional contracts, creating a unidirectional, chain-like dependency.

[3]Cyclic calls, on the other hand, occur when multiple contracts establish a closed-loop interdependency, forming cyclic relationships.

E. Efficient Communication

1) Open Issue: High cross-shard communication overhead continues to hinder sharding performance and limit blockchain scalability. There remains significant potential for further optimizing cross-shard communication. Most cross-shard transaction processing mechanisms require coordination among related shards to reach consensus. As the number of shards increases, communication overhead inevitably becomes excessive. This issue is particularly pronounced in distributed 2PC protocols and Cross-Shard BFT mechanisms. For instance, when the distributed 2PC protocol processes multi-input cross-shard transactions, each shard transmits a unique availability certificate for each input to all other involved shards. In a typical scenario where a cross-shard transaction involves i input shards and j output shards, the message exchange volume reaches $i^2 + i * j$ [113]. To enhance the efficiency of cross-shard transaction processing, communication overhead must be consistently optimized.

2) Future Research Direction: Further research is highly encouraged to optimize communication efficiency among shards. One promising approach is to introduce a batch certification mechanism that proves the availability of multiple inputs within a single certificate. A potential implementation of this approach could leverage the Merkle tree technique to create a single certificate encapsulating multiple input availabilities, which is then shared with relevant shards after achieving BFT consensus within the shard. Additionally, advanced encoding techniques can be explored to reduce the volume of data exchanged between shards. For example, Polyshard [114] and the 2-Dimensional Sharding Scheme [115] propose computational redundancy encoding schemes. These allow nodes to decode and recover original transaction information after receiving a certain number of encoded pieces, eliminating the need for all data to be transmitted. Finally, node allocation strategies should be further refined by considering such factors as geographical distance [116]. By minimizing the number of communication rounds between distant nodes, the overall communication efficiency of the globally sharded blockchain can be significantly improved.

F. Comprehensive and Modular Experimental Platforms

1) Open Issue: Currently, there is a lack of reliable simulation or experimental platforms for validating the CTAOP mechanisms proposed by researchers. The implementation of

sharded blockchain mechanisms is often tailored to the development of specific consensus algorithms. Many researchers modify existing blockchain platforms, such as Bitcoin and Ethereum, to conduct experiments. However, these platforms either lack native support for sharding or have sharding components that are tightly integrated with other system elements, making performance evaluation and comparison with prior research challenging [117].

2) Future Research Direction: As researchers advance CTAOP mechanisms, there is a critical need for a comprehensive and modular sharded blockchain platform to serve as a standardized experimental framework for performance evaluation. This platform should, at minimum, include a simulator with multiple essential components, such as transaction allocation schemes, cross-shard consensus protocols, and ledger management mechanisms [117]. Additionally, it must support diverse performance metrics for blockchain evaluation while providing modular interfaces to facilitate the design and testing of new mechanisms. Existing platforms, such as BlockEmulator [117] and ShardEval [118], offer sharded blockchain simulation frameworks to enable users to evaluate sharding performance. Expanding and refining these platforms could significantly enhance research in this domain.

G. Sustainable Scenario-oriented Research

1) Open Issue: Application-oriented research in blockchain sharding remains insufficient. While blockchain sharding enhances the performance and scalability of traditional blockchains, providing technical support for emerging applications, popular use cases—such as Web3 and Decentralized Finance (DeFi)—are primarily centralized commercial projects deployed on public blockchains [119]. These applications cannot be directly integrated with sharded blockchains, creating compatibility challenges between blockchain sharding mechanisms and real-world scenarios. Addressing these challenges is essential for developing practical sharded blockchain systems.

2) Future Research Direction: To transition CTAOP research into large-scale applications, it is essential to address compatibility challenges and achieve ecosystem integration. For example, integrating decentralized ecosystems like Web3 with blockchain sharding requires designing architectures and models tailored to domain-specific needs. Key research areas include: defining interoperable interaction layers, establishing criteria for client and validator node selection, and designing transaction and ledger formats [120]. Furthermore, establishing a bidirectional feedback loop driven by "technology R&D – real-world application" is imperative. Huang *et al.* pioneered research in this area by developing the Broker2Earn protocol [121] based on a self-developed Brokerchain sharding architecture. This approach addresses inefficiency in fragmented DeFi blockchains, improving processing speed and liquidity utilization while introducing an incentive mechanism. The success of BrokerChain in DeFi not only validates its feasibility but also provides a replicable framework for translating theoretical advancements into industrial applications. Ensuring the sustainability of blockchain research, development, and application becomes essential for promoting ecosystem upgrades and long-term adoption.

VII. Conclusion

This survey provided a thorough review on existing CTAOP mechanisms. We first provided a brief overview of cross-shard transactions based on two transaction models (i.e., the UTXO model and the Account/Balance model) and introduced related basic concepts. Then, we proposed two sets of criteria that should be satisfied by transaction allocation optimization schemes and cross-shard transaction processing mechanisms, respectively. After that, we provided the taxonomies of the two types of CTAOP mechanisms. By employing our proposed criteria, we thoroughly reviewed the existing literature of CTAOP by following the taxonomies. Based on our comprehensive review, we highlighted a number of open issues and proposed a list of future research directions accordingly to guide future research efforts on CTAOP mechanisms.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China under Grants U23A20300; in part by the Key Research Project of Shaanxi Natural Science Foundation under Grant 2023-JC-ZD-35; in part by the Concept Verification Funding of Hangzhou Institute of Technology of Xidian University under Grant GNYZ2024XX007, and in part by the 111 Project under Grant B16037.

References

[1] M. Belotti, N. Božić, G. Pujolle, and S. Secci, "A vademecum on blockchain technologies: When, which, and how," *IEEE Communications Surveys & Tutorials*, vol. 21, no. 4, pp. 3796–3838, 2019.

[2] A. D. Hunt, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[3] A. Gervais, G. O. Karame, V. Capkun, and S. Capkun, "Is bitcoin a decentralized currency?" *IEEE Security & Privacy*, vol. 12, no. 3, pp. 54–60, 2014.

[4] Benefits of accepting Visa. [Online]. Available: https://usa.visa.com/content_library/modal/benefits-accepting-visa.html

[5] J. C. Corbett, J. Dean, M. Epstein, A. Fikes, C. Frost, J. J. Furman, S. Ghemawat, A. Gubarev, C. Heiser, P. Hochschild, W. Hsieh, S. Kanthak, E. Kogan, H. Li, A. Lloyd, S. Melnik, D. Mwaura, D. Nagle, S. Quinlan, R. Rao, L. Rolig, Y. Saito, M. Szymaniak, C. Taylor, R. Wang, and D. Woodford, "Spanner: Google's globally distributed database," *ACM Trans. Comput. Syst.*, vol. 31, no. 3, Aug. 2013.

[6] 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, p. 17–30.

[7] S. Kantesariya and D. Goswami, "Determining optimal shard size in a hierarchical blockchain architecture," in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, 2020, pp. 1–3.

[8] J. R. Douceur, "The sybil attack," in *International workshop on peer-to-peer systems*, 2002, pp. 251–260.

[9] D. Tennakoon and V. Gramoli, "Dynamic blockchain sharding," in *5th International Symposium on Foundations and Applications of Blockchain 2022 (FAB 2022)*, 2022.

[10] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual international cryptology conference*, 2017, pp. 357–388.

[11] 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)*, 2018, pp. 583–598.

[12] 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.

[13] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: What it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1, 2015, pp. 57–64.

[14] 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.

[15] 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.

[16] H. Huang, X. Peng, J. Zhan, S. Zhang, Y. Lin, Z. Zheng, and S. Guo, "Brokerchain: A cross-shard blockchain protocol for account/balance-based state sharding," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022, pp. 1968–1977.

[17] H. Huang, Z. Yin, Q. Chen, J. Zheng, X. Luo, G. Ye, X. Peng, Z. Zheng, and S. Guo, "Brokerchain: A blockchain sharding protocol by exploiting broker accounts," *IEEE Transactions on Networking*, pp. 1–16, 2025.

[18] Y. Tao, B. Li, J. Jiang, H. C. Ng, C. Wang, and B. Li, "On sharding open blockchains with smart contracts," in *2020 IEEE 36th international conference on data engineering (ICDE)*, 2020, pp. 1357–1368.

[19] 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)*, 2019, pp. 525–535.

[20] Y. Liu, J. Liu, Q. Wu, H. Yu, Y. Hei, and Z. Zhou, "SSHC: A secure and scalable hybrid consensus protocol for sharding blockchains with a formal security framework," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 3, pp. 2070–2088, 2020.

[21] Y. Liu, X. Xing, H. Cheng, D. Li, Z. Guan, J. Liu, and Q. Wu, "A flexible sharding blockchain protocol based on cross-shard byzantine fault tolerance," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 2276–2291, 2023.

[22] J. N. Gray, *Notes on data base operating systems*. Berlin, Heidelberg: Springer, 1978, pp. 393–481.

[23] G. Wang, Z. J. Shi, M. Nixon, and S. Han, "Sok: Sharding on blockchain," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, 2019, pp. 41–61.

[24] G. Yu, X. Wang, K. Yu, W. Ni, J. A. Zhang, and R. P. Liu, "Survey: Sharding in blockchains," *IEEE Access*, vol. 8, pp. 14 155–14 181, 2020.

[25] A. Hafid, A. S. Hafid, and M. Samih, "Scaling blockchains: A comprehensive survey," *IEEE access*, vol. 8, pp. 125 244–125 262, 2020.

[26] R. Han, J. Yu, H. Lin, S. Chen, and P. Esteves-Veríssimo, "On the security and performance of blockchain sharding," *Cryptology ePrint Archive*, 2021.

[27] H. Huang, W. Kong, X. Peng, and Z. Zheng, "Survey on blockchain sharding technology," *Computer Engineering*, vol. 48, no. 6, p. 10, 2022.

[28] 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.

[29] X. Liu, H. Xie, Z. Yan, and X. Liang, "A survey on blockchain sharding," *ISA transactions*, vol. 141, pp. 30–43, 2023.

[30] Y. Li, J. Wang, and H. Zhang, "A survey of state-of-the-art sharding blockchains: Models, components, and attack surfaces," *Journal of Network and Computer Applications*, p. 103686, 2023.

[31] L. Zhang, Y. Wang, Y. Ding, H. Liang, C. Yang, and C. Li, "Sharding technologies in blockchain: Basics, state of theart, and challenges," in *Blockchain and Trustworthy Systems*. Singapore: Springer, 2024, pp. 242–255.

[32] Q. Yang, H. Huang, Z. Yin, Y. Lin, Q. Chen, X. Luo, T. Li, X. Liu, and Z. Zheng, "The state-of-the-art and promising future of blockchain sharding," *IEEE Communications Magazine*, vol. 63, no. 4, pp. 192–198, 2025.

[33] V. Buterin. Ethereum: A next-generation smart contracts and decentralized application platform. [Online]. Available: https://ethereum.org

[34] W. Zou, D. Lo, P. S. Kochhar, X.-B. D. Le, X. Xia, Y. Feng, Z. Chen, and B. Xu, "Smart contract development: Challenges and opportunities," *IEEE transactions on software engineering*, vol. 47, no. 10, pp. 2084–2106, 2019.

[35] D. Jin, Z. Yu, P. Jiao, S. Pan, D. He, J. Wu, S. Y. Philip, and W. Zhang, "A survey of community detection approaches: From statistical modeling to deep learning," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 2, pp. 1149–1170, 2021.

[36] A. Buluc, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz, "Recent advances in graph partitioning," 2015. [Online]. Available: https://arxiv.org/abs/1311.3144

[37] Z. Abbas, V. Kalavri, P. Carbone, and V. Vlassov, "Streaming graph partitioning: an experimental study," *Proceedings of the VLDB Endowment*, vol. 11, no. 11, pp. 1590–1603, 2018.

[38] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *Acm Sigact News*, vol. 33, no. 2, pp. 51–59, 2002.

[39] J. Gray and A. Reuter, *Transaction Processing: Concepts and Techniques*. Morgan Kaufmann Publishers Inc., 1992.

[40] C. Cachin, R. Guerraoui, and L. Rodrigues, *Introduction to reliable and secure distributed programming*. Springer Science & Business Media, 2011.

[41] E. Gelenbe and K. C. Sevcik, "Analysis of update synchronization for multiple copy data bases," *IEEE Transactions on Computers*, no. 10, pp. 737–747, 1979.

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

[43] M. Castro and B. Liskov, "Practical byzantine fault tolerance and proactive recovery," *ACM Transactions on Computer Systems (TOCS)*, vol. 20, no. 4, pp. 398–461, 2002.

[44] M. P. Herlihy and J. M. Wing, "Linearizability: A correctness condition for concurrent objects," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 12, no. 3, pp. 463–492, 1990.

[45] M. T. zsu and P. Valduriez, *Principles of distributed database systems*. Springer, 1999, vol. 2.

[46] W. Vogels, "Eventually consistent," *Communications of the ACM*, vol. 52, no. 1, pp. 40–44, 2009.

[47] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser, "Managing update conflicts in bayou, a weakly connected replicated storage system," *ACM SIGOPS Operating Systems Review*, vol. 29, no. 5, pp. 172–182, 1995.

[48] D. Abadi, "Consistency tradeoffs in modern distributed database system design: Cap is only part of the story," *Computer*, vol. 45, no. 2, pp. 37–42, 2012.

[49] D. Reijsbergen and T. T. A. Dinh, "On exploiting transaction concurrency to speed up blockchains," in *2020 IEEE 40th International Conference on Distributed Computing Systems (ICDCS)*, 2020, pp. 1044–1054.

[50] P. A. Bernstein and N. Goodman, "Concurrency control in distributed database systems," *ACM Computing Surveys (CSUR)*, vol. 13, no. 2, pp. 185–221, 1981.

[51] J. N. Gray, R. A. Lorie, G. R. Putzolu, and I. L. Traiger, "Granularity of locks and degrees of consistency in a shared data base," in *IFIP Working Conference on Modelling in Data Base Management Systems*, 1976, pp. 365–394.

[52] H.-T. Kung and J. T. Robinson, "On optimistic methods for concurrency control," *ACM Transactions on Database Systems (TODS)*, vol. 6, no. 2, pp. 213–226, 1981.

[53] J. Xu, Y. Ming, Z. Wu, C. Wang, and X. Jia, "X-shard: Optimistic cross-shard transaction processing for sharding-based blockchains," *IEEE Transactions on Parallel and Distributed Systems*, vol. 35, no. 4, pp. 548–559, 2024.

[54] Z. Hong, S. Guo, E. Zhou, J. Zhang, W. Chen, J. Liang, J. Zhang, and A. Zomaya, "Prophet: Conflict-free sharding blockchain via byzantine-tolerant deterministic ordering," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*, 2023, pp. 1–10.

[55] X. Wang, B. Li, L. Jia, and Y. Sun, "Orbit: A dynamic account allocation mechanism in sharding blockchain system," in *2024 IEEE 44th International Conference on Distributed Computing Systems (ICDCS)*, 2024, pp. 333–344.

[56] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Trans. Program. Lang. Syst.*, vol. 4, no. 3, p. 382–401, Jul. 1982.

[57] M. Saad, J. Spaulding, L. Njilla, C. Kamhoua, S. Shetty, D. Nyang, and D. Mohaisen, "Exploring the attack surface of blockchain: A comprehensive survey," *IEEE Communications Surveys & Tutorials*, vol. 22, no. 3, pp. 1977–2008, 2020.

[58] A. Sonnino, S. Bano, M. Al-Bassam, and G. Danezis, "Replay attacks and defenses against cross-shard consensus in sharded distributed ledgers," in *2020 IEEE European Symposium on Security and Privacy (EuroS&P)*, 2020, pp. 294–308.

[59] R. Han, Z. Yan, X. Liang, and L. T. Yang, "How can incentive mechanisms and blockchain benefit with each other? a survey," *ACM Comput. Surv.*, vol. 55, no. 7, dec 2022.

[60] V. H. Allan, R. B. Jones, R. M. Lee, and S. J. Allan, "Software pipelining," *ACM Computing Surveys (CSUR)*, vol. 27, no. 3, pp. 367–432, 1995.

[61] Y. Liu, J. Liu, J. Yin, G. Li, H. Yu, and Q. Wu, "Cross-shard transaction processing in sharding blockchains," in *Algorithms and Architectures for Parallel Processing*. Cham, Switzerland: Springer, 2020, pp. 324–339.

[62] A. Liu, Y. Liu, Q. Wu, B. Zhao, D. Li, Y. Lu, R. Lu, and W. Susilo, "CHERUBIM: A secure and highly parallel cross-shard consensus using quadruple pipelined two-phase commit for sharding blockchains," *IEEE Transactions on Information Forensics and Security*, vol. 19, pp. 3178–3193, 2024.

[63] N. Szabo, "Formalizing and securing relationships on public networks," *First monday*, 1997.

[64] Z. Zheng, S. Xie, H.-N. Dai, W. Chen, X. Chen, J. Weng, and M. Imran, "An overview on smart contracts: Challenges, advances and platforms," *Future Generation Computer Systems*, vol. 105, pp. 475–491, 2020.

[65] 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)*, 2023, pp. 721–733.

[66] M. J. Amiri, D. Agrawal, and A. El Abbadi, "Sharper: Sharding permissioned blockchains over network clusters," in *Proceedings of the 2021 international conference on management of data*, 2021, pp. 76–88.

[67] Z. Cai, J. Liang, W. Chen, Z. Hong, H.-N. Dai, J. Zhang, and Z. Zheng, "Benzene: Scaling blockchain with cooperation-based sharding," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 2, pp. 639–654, 2022.

[68] K. Zhou, X. Zhang, C. Wang, and H. Cheng, "Accelerating cross-shard blockchain consensus via decentralized coordinators service with verifiable global states," *IEEE Transactions on Services Computing*, vol. 17, no. 4, pp. 1340–1353, 2024.

[69] 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)*, 2018, pp. 128–133.

[70] 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)*, 2022, pp. 284–294.

[71] L. Ren, P. A. Ward, and B. Wong, "Toward reducing cross-shard transaction overhead in sharded blockchains," in *Proceedings of the 16th ACM International Conference on Distributed and Event-Based Systems*, 2022, pp. 43–54.

[72] 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*, vol. 35, no. 3, pp. 405–420, 2024.

[73] M. Król, O. Ascigil, S. Rene, A. Sonnino, M. Al-Bassam, and E. Rivière, "Shard scheduler: object placement and migration in sharded account-based blockchains," in *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, 2021, pp. 43–56.

[74] 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.

[75] Y. Tao, B. Li, and B. Li, "On sharding across heterogeneous blockchains," in *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, 2023, pp. 477–489.

[76] G. Karypis and V. Kumar, "A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices," *University of Minnesota, Department of Computer Science and Engineering, Army HPC Research Center, Minneapolis, MN*, vol. 38, pp. 7–1, 1998.

[77] F. Chung, "A brief survey of pagerank algorithms." *IEEE Trans. Netw. Sci. Eng.*, vol. 1, no. 1, pp. 38–42, 2014.

[78] A. Bundy and L. Wallen, "Breadth-first search," *Catalogue of artificial intelligence tools*, pp. 13–13, 1984.

[79] 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. 10008, pp. 1–12, 2008.

[80] F. A. Gers, J. Schmidhuber, and F. Cummins, "Learning to forget: Continual prediction with lstm," *Neural computation*, vol. 12, no. 10, pp. 2451–2471, 2000.

[81] B. G. Ryder, "Constructing the call graph of a program," *IEEE Transactions on Software Engineering*, no. 3, pp. 216–226, 1979.

[82] R. Branzei, D. Dimitrov, and S. Tijs, *Models in cooperative game theory*. Springer Science & Business Media, 2008, vol. 556.

[83] B. Wang, K. R. Liu, and T. C. Clancy, "Evolutionary cooperative spectrum sensing game: how to collaborate?" *IEEE transactions on communications*, vol. 58, no. 3, pp. 890–900, 2010.

[84] Z. Liu, Y. Xiang, J. Shi, P. Gao, H. Wang, X. Xiao, B. Wen, and Y.-C. Hu, "Hyperservice: Interoperability and programmability across heterogeneous blockchains," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 549–566.

[85] H. Crowder, E. L. Johnson, and M. Padberg, "Solving large-scale zero-one linear programming problems," *Operations Research*, vol. 31, no. 5, pp. 803–834, 1983.

[86] A. H. Land and A. G. Doig, *An automatic method for solving discrete programming problems*. Springer, 2010.

[87] M. S. Ozdayi, Y. Guo, and M. Zamani, "Instachain: Breaking the sharding limits via adjustable quorums," *Cryptology ePrint Archive*, 2022.

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

[89] M. Li, Y. Lin, W. Wang, and J. Zhang, "CoChain: High concurrency blockchain sharding via consensus on consensus," in *IEEE INFOCOM 2023-IEEE Conference on Computer Communications*, 2023, pp. 1–10.

[90] M. Li, Y. Lin, J. Zhang, and W. Wang, "Jenga: Orchestrating smart contracts in sharding-based blockchain for efficient processing," in *2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS)*, 2022, pp. 133–143.

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

[92] 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.

[93] E. Syta, I. Tamas, D. Visher, D. I. Wolinsky, P. Jovanovic, L. Gasser, N. Gailly, I. Khoffi, and B. Ford, "Keeping authorities 'honest or bust' with decentralized witness cosigning," in *2016 IEEE Symposium on Security and Privacy (SP)*, 2016, pp. 526–545.

[94] V. Costan and S. Devadas, "Intel sgx explained," *IACR Cryptol. ePrint Arch.*, vol. 2016, p. 86, 2016.

[95] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "HotStuff: Bft consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 347–356.

[96] M. Szydlo, "Merkle tree traversal in log space and time," in *Advances in Cryptology-EUROCRYPT 2004: International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004. Proceedings 23*, 2004, pp. 541–554.

[97] L. Ren, K. Nayak, I. Abraham, and S. Devadas, "Practical synchronous byzantine consensus," *arXiv preprint arXiv:1704.02397*, 2017.

[98] P. Maymounkov and D. Mazieres, "Kademlia: A peer-to-peer information system based on the xor metric," in *International Workshop on Peer-to-Peer Systems*, 2002, pp. 53–65.

[99] G. G. Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D.-A. Seredinschi, O. Tamir, and A. Tomescu, "SBFT: A scalable and decentralized trust infrastructure," in *2019 49th Annual IEEE/IFIP international conference on dependable systems and networks (DSN)*, 2019, pp. 568–580.

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

[101] V. Medina and J. M. García, "A survey of migration mechanisms of virtual machines," *ACM Computing Surveys (CSUR)*, vol. 46, no. 3, pp. 1–33, 2014.

[102] H. Huang, Y. Zhao, and Z. Zheng, "tMPT: Reconfiguration across blockchain shards via trimmed merkle patricia trie," in *2023 IEEE/ACM 31st International Symposium on Quality of Service (IWQoS)*, 2023, pp. 1–10.

[103] D. J. Abadi and J. M. Faleiro, "An overview of deterministic database systems," *Communications of the ACM*, vol. 61, no. 9, pp. 78–88, 2018.

[104] J. M. Faleiro and D. J. Abadi, "Rethinking serializable multiversion concurrency control," *Proceedings of the VLDB Endowment*, vol. 8, no. 11, 2015.

[105] A. Thomson, T. Diamond, S.-C. Weng, K. Ren, P. Shao, and D. J. Abadi, "Calvin: fast distributed transactions for partitioned database

systems," in *Proceedings of the 2012 ACM SIGMOD international conference on management of data*, 2012, pp. 1–12.
[106] Y. Lu, X. Yu, L. Cao, and S. Madden, "Aria: a fast and practical deterministic oltp database," *Proc. VLDB Endow.*, vol. 13, no. 12, p. 2047–2060, Jul. 2020.
[107] A. Liu, Y. Liu, Z. Pan, Y. Li, J. Liu, and Y. Lu, "Kronos: A robust sharding blockchain consensus with optimal communication overhead," *Cryptology ePrint Archive*, 2024.
[108] C. Huang, Z. Wang, H. Chen, Q. Hu, Q. Zhang, W. Wang, and X. Guan, "Repchain: A reputation-based secure, fast, and high incentive blockchain system via sharding," *IEEE Internet of Things Journal*, vol. 8, no. 6, pp. 4291–4304, 2020.
[109] P. Zheng, Q. Xu, Z. Zheng, Z. Zhou, Y. Yan, and H. Zhang, "Meepo: Sharded consortium blockchain," in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, 2021, pp. 1847–1852.
[110] J. Zhang, W. Chen, Z. Hong, G. Xiao, L. Du, and Z. Zheng, "Efficient execution of arbitrarily complex cross-shard contracts for blockchain sharding," *IEEE Transactions on Computers*, no. 01, pp. 1–14, 2024.
[111] X. Qi and Y. Li, "Lightcross: Sharding with lightweight cross-shard execution for smart contracts," in *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications*, 2024, pp. 1681–1690.
[112] S. Ott, B. Orthen, A. Weidinger, J. Horsch, V. Nayani, and J.-E. Ekberg, "MultiTEE: Distributing trusted execution environments," in *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, 2024, pp. 1617–1629.
[113] J. Tian, J. Tian, and R. Du, "MSLShard: An efficient sharding-based trust management framework for blockchain-empowered iot access control," *Journal of Parallel and Distributed Computing*, vol. 185, p. 104795, 2024.
[114] S. Li, M. Yu, C.-S. Yang, A. S. Avestimehr, S. Kannan, and P. Viswanath, "Polyshard: Coded sharding achieves linearly scaling efficiency and security simultaneously," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 249–261, 2020.
[115] C. Wang and N. Raviv, "Low latency cross-shard transactions in coded blockchain," in *2021 IEEE International Symposium on Information Theory (ISIT)*, 2021, pp. 2678–2683.
[116] C. Mao and W. Golab, "Geochain: a locality-based sharding protocol for permissioned blockchains," in *Proceedings of the 24th International Conference on Distributed Computing and Networking*, 2023, pp. 70–79.
[117] H. Huang, G. Ye, Q. Yang, Q. Chen, Z. Yin, X. Luo, J. Lin, J. Zheng, T. Li, and Z. Zheng, "BlockEmulator: An emulator enabling to test blockchain sharding protocols," *IEEE Transactions on Services Computing*, vol. 18, no. 2, pp. 690–703, 2025.
[118] V. Priyadarshi, S. Goel, and K. Kapoor, "Analysis of optimal number of shards using shardeval, a simulator for sharded blockchains," in *Information Systems Security: 19th International Conference, ICISS 2023, Raipur, India, December 16–20, 2023, Proceedings*, 2023, p. 339–359.
[119] H. Huang, J. Wu, and Z. Zheng, *From Blockchain to Web3 and Metaverse*. Springer, 2023.
[120] Y. Liu, B. Zhao, Z. Zhao, J. Liu, X. Lin, Q. Wu, and W. Susilo, "SS-DID: A secure and scalable web3 decentralized identity utilizing multilayer sharding blockchain," *IEEE Internet of Things Journal*, vol. 11, no. 15, pp. 25 694–25 705, 2024.
[121] Q. Chen, H. Huang, Z. Yin, G. Ye, and Q. Yang, "Broker2Earn: Towards maximizing broker revenue and system liquidity for sharded blockchains," in *IEEE INFOCOM 2024 - IEEE Conference on Computer Communications*, 2024, pp. 251–260.

Zitong Zhang is currently pursuing the M.Eng degree with the School of Cyber Engineering, Xidian University, Xi'an, China. His research interests include blockchain sharding and distributed systems.

Haoxiang Han received the bachelor's degree in network engineering from Fujian Agriculture And Forestry University in 2020. He is currently pursuing the Ph.D degree in cyberspace security at Xidian University. His research interests are in blockchain and sharding technology.

Zheng Yan is currently a Huashan distinguished professor at the Xidian University, China. She earned her PhD degree from the Helsinki University of Technology, Finland. She joined the Nokia Research Center, Helsinki in 2000, working as a senior researcher until 2011. She worked as a visiting professor and a Finnish Academy Fellow at the Aalto University, Finland for over seven years. Her research interests are in cyber trust, security, privacy, and data analytics. She has authored over 400 publications, with 270+ first and corresponding authorships, featured prominently in top-tier venues. She is the sole author of two books on trust management, utilized in teaching for a decade. 150+ patents invented by her have been adopted by industry, a few of them have been incorporated into international standards and widely used in practice. She has delivered more than 40 invited keynote speeches and talks at international conferences and world leading companies. Dr. Yan served and is serving as an EiC/area/associate/guest editor for 60+ reputable journals, a steering committee member of several conferences, a general/program chair for 40+ international conferences. She is a founding steering committee co-chair of IEEE International Conference on Blockchain. Dr. Yan is recognized as a Stanford World Top 2% Scientist and an Elsevier Highly Cited Chinese Researcher. She has earned many accolades, including Distinguished Inventor of Nokia, N²Women Star in Computer Networking and Communications, the IEEE TCSC Award for Excellence in Scalable Computing, the ELEC Impact Award for patent contributions to Finnish society, the Best Journal Paper Award issued by IEEE ComSoc and several other Best Paper awards, 18 times IEEE Distinguished/Outstanding Leadership/Service awards, three EU awards, etc. She is an External Member of Finnish Academy of Science and Letters, and a Fellow of IEEE, IET, AAIA, and AIIA. Her excellence has been covered by the media on numerous occasions.

Erol Gelenbe (Life Fellow, IEEE) received the B.S.E.E. degree from METU, Turkey, the M.S.E.E. and Ph.D. degrees from the Polytechnic Institute of NYU, and the D.Sc. degree in Mathematical Sciences from Sorbonne University. He pioneered system performance evaluation methods and invented the random neural network and G-Networks, while also helping to develop commercial tools, including the queueing network analysis package QNAP and the manufacturing simulator FLEXSIM. He has graduated over 90 Ph.D. students, including 24 women. He is currently a Professor with the Institute of Theoretical and Applied Informatics, Polish Academy of Sciences, a visiting faculty at King's College London, and an associated researcher at CNRS I3S, Université Côte d'Azur (Nice). Previously, he has held chaired professorships at Imperial College, the University of Central Florida, Duke University, Université Paris-Descartes, Paris-Saclay University, and Université de Liège. He was a Principal Investigator of numerous European Union research projects and a Coordinator of FP7 NEMESYS and H2020 SerIoT. He has also been funded in the U.S. by NSF, ONR, ARO and industry, and in the UK by UKRI, MoD and also industry. He is also a fellow of ACM, IFIP, RSS, AAIA and IET. He is an Elected Fellow of the French National Academy of Technologies, the Science Academies of Poland, Turkey, the Royal Academy of Belgium, an Honorary Fellow of the Hungarian Academy of Sciences and the Islamic World Academy of Sciences. He is a Member of Academia Europaea and the National Academy of AI (USA). His prizes include the Parlar Foundation Science Award (Turkey), the Grand Prix France Télécom, the ACM SIGMETRICS Life-Time Achievement Award, the IET Oliver Lodge Medal, and the Mustafa Prize. He was awarded the honors of Chevalier de la Lègion d'Honneur, Chevalier des Palmes Académiques, and Commandeur de l'Ordre du Mérite by France. He was also awarded Commander of the Order of the Crown of Belgium, Commendatore al Merito and Grande Ufficiale dell'Ordine della Stella by Italy, and Officer of the Order of Merit of Poland. He is ranked in the top 7% of the Stanford 2%. ScholarGPS ranks him No.2 in Poland over all fields, and No. 1 in Poland for Engineering and Computer Science.