Article | Received 5 March 2025; Accepted 9 May 2025; Published 20 May 2025
https://doi.org/10.55092/blockchain20250010

BMDS-Shard: a parallel and atomicity-guaranteed cross-shard blockchain protocol

Haitao Chen[1,2], Jian Zheng[3], Xiaofei Luo[3], Zhili Liu[2], Qingni Shen[1,4,*] and Huawei Huang[3]

[1] School of Software and Microelectronics, Peking University, Beijing, China
[2] Zhongbao Union Technology Co., Ltd, Shanghai, China
[3] School of Software Engineering, Sun Yat-sen University, Zhuhai, China
[4] National Engineering Research Center for Software Engineering, Peking University, Beijing, China

* Correspondence author; E-mail: qingnishen@ss.pku.edu.cn.

Highlights:

- Extends sharding technology research from transaction to data sharing scenarios.
- Improving transaction efficiency through parallel cross-shard processing.
- Ensuring atomicity of cross-shard transactions through snapshot synchronization.
- Optimizing cross-shard transaction efficiency via priority scheduling.

Abstract: Sharding is a key technology to improve the scalability of blockchain, and cross-shard transaction protocols are the core of sharding to ensure transaction atomicity and consistency. Many scholars have conducted related research, including two-phase commit (2PC), transaction splitting, relay transactions, and deterministic ordering. However, there remains a challenge in balancing the efficiency and atomicity of cross-shard transactions (CTXs). Thus, this paper proposes a parallel and atomicity-guaranteed cross-shard transaction protocol, BMDS-Shard. Firstly, by decoupling cross-shard transactions into synchronized transactions in the source and target shards through beacon nodes, and automatically identifying cross-shard processing flows, we assign higher processing priority within shards, thereby solving the inefficiency of relay transactions. Secondly, based on a multi-beacon node data snapshot mechanism, we ensure instant and reliable state data feedback to the source shard, addressing the insufficient atomicity of transaction splitting. Finally, we have implemented the BMDS-Shard protocol, conducted theoretical analysis, and performed experimental validation on the BlockEmulator platform. Both theoretical and experimental results demonstrate that while guaranteeing transaction atomicity, our protocol reduces cross-shard transaction confirmation latency by 65.82% and 65.06% compared to existing relay transaction protocols Monoxide and BrokerChain, respectively.

Keywords: blockchain; sharding; cross-shard transaction; snapshot synchronization; priority scheduling; medical data sharing

1. Introduction

As blockchain technology continues to evolve, an increasing number of industries are beginning to utilize this technology to address practical issues, including cryptocurrency transactions, data sharing transactions and *etc.* Let us make the medical data sharing blockchain [1] as an example, by connecting nodes from medical institutions, commercial insurance institutions, judicial sectors, and other industry bodies (see Figure 1), enables functionalities such as cross-hospital diagnosis and treatment, commercial insurance claims, and judicial evidence collection. However, when blockchain systems come to broader regions and more institutions, there are challenges related to insufficient scalability, including low throughput, high latency, and significant storage pressure. Sharding technology is recognized as one of the most promising solutions to significantly enhance the scalability of blockchain systems [2–4]. In a sharded blockchain system, transactions are inherently random, and the sender and receiver of a transaction are often mapped to multiple different shards. The system requires cross-shard transaction protocols to ensure the atomicity and consistency of such transactions [5].

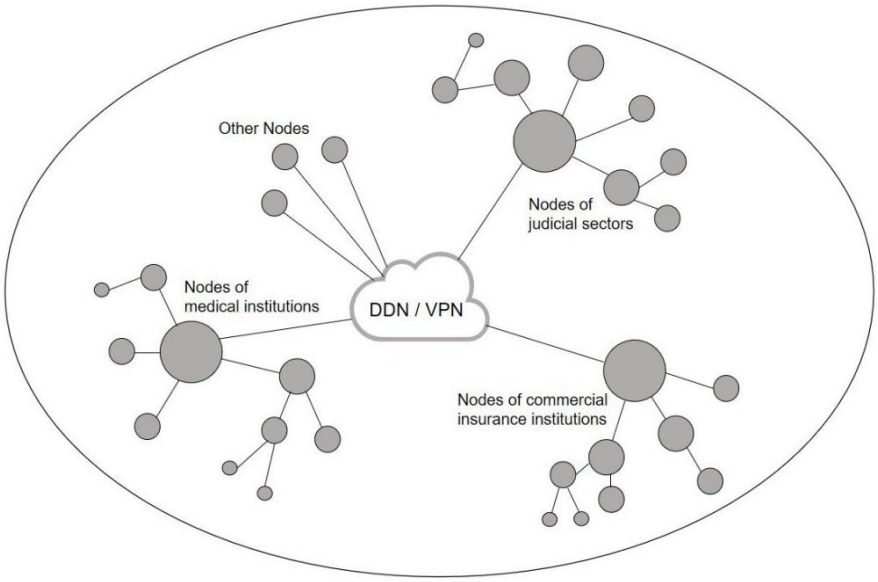

Figure 1. Schematic diagram of a shard network architecture.

Currently, research on cross-shard transaction protocols can be categorized into four approaches: 2PC, transaction splitting, relay transactions, and deterministic ordering. The most commonly used cross-shard transaction protocol is the 2PC [6,7], which is based on a pessimistic concurrency control (PCC) strategy and is mainly applied in the Unspent Transaction Output (UTXO) model. When handling cross-shard transactions, the system selects a coordinator that waits for other shards to be ready, requiring multiple rounds of consensus. This increases transaction confirmation latency, and the single point of failure issue of the coordinator also needs attention. The deterministic ordering protocol [6,8] eliminates the coordinator by pre-executing to obtain prerequisite information for cross-shard transactions and determining the execution order, or by using a sequencing lock mechanism to ensure atomicity and serializability of cross-shard transactions. This effectively reduces transaction conflicts, but lacks parallelism and is prone to transaction congestion in high-conflict scenarios. The transaction splitting

protocol, proposed by RapidChain [9], is based on an optimistic concurrency control (OCC) strategy, splitting cross-shard transactions into two or more independent intra-shard transactions (ITXs), primarily applied in the UTXO model. While this protocol simplifies transaction structures, it introduces more processing steps [10], complicates exception rollback handling, and struggles to guarantee atomicity. The relay transactions protocol is typically applied in the account model. After completing the intra-shard transaction in the shard where the transferring account resides, relay transactions are initiated to send cross-shard transaction information to the receiving shard, which then complete the intra-shard transaction, ensuring atomicity [11]. However, this protocol requires additional relay transactions, leading to higher transaction confirmation latency [10]. Moreover, since cross-shard transactions are executed serially, the lack of parallelism further impacts transaction confirmation latency.

Therefore, existing cross-shard transaction protocols struggle to balance the efficiency of cross-shard transactions with the necessity of atomicity. In response to the aforementioned challenges, this paper proposes a parallel and atomicity-guaranteed cross-shard transaction protocol, BMDS-Shard, and makes the following key contributions:

(1) A parallel and efficient cross-shard transaction mechanism is introduced. By setting up beacon nodes with special cross-shard functionalities, cross-shard transactions are decoupled into source and target shard synchronous transactions through these beacon nodes. The mechanism automatically identifies the cross-shard processing flow and assigns a higher processing priority within the shard, addressing the inefficiency of relay transactions.

(2) An atomicity-guaranteed cross-shard transaction mechanism is proposed. Based on a data snapshot mechanism of multiple beacon nodes, state data is reliably and instantly fed back to the source shard, solving the issue of insufficient atomicity in split transactions.

(3) We have implemented the BMDS-Shard protocol, conducted theoretical analysis, and performed experimental validation on the BlockEmulator [12]. Both theoretical and experimental results demonstrate that while guaranteeing transaction atomicity, our protocol reduces cross-shard transaction confirmation latency by 65.82% and 65.06% compared to existing relay transaction protocols Monoxide and BrokerChain, respectively.

2. Background and motivation

At present, numerous scholars both domestically and internationally have embarked on research aimed at enhancing cross-shard transaction protocols. Although the primary focus of these studies remains predominantly on transfer scenarios, they offer a certain degree of inspiration for the optimization of cross-shard transaction protocols in evidence storage scenarios.

2.1. Two-phase commit(2PC)

In the 2PC protocol, there is a coordinator responsible for collecting transfer information and making decisions. The coordinator first requests other nodes to inquire if they are ready. If and only if all nodes are ready, the coordinator sends a message to synchronize all nodes and execute the transaction; otherwise, it sends a message to all nodes to cancel the transaction [5]. Most existing blockchain sharding schemes (for example, Omniledger [13], Chainspace [14], Pyramid [15], *etc.*) use 2PC to handle cross-shard

transactions. From the perspective of the different roles undertaken by the coordination task, they are mainly divided into three methods: client coordination, input shard coordination, and output shard coordination. OmniLedger assigns the coordinator role to the client, which collects agree or disagree messages from all input shards through the Atomix protocol to make decisions. This scheme requires the client to broadcast transactions across the entire network, and input shards generate asset proofs and submit them, resulting in significant communication overhead. It does not meet the needs of light client scenarios, and the client is vulnerable to denial of service (DoS) attacks by malicious users. Chainspace assigns the coordinator role to the input shard, achieving inter-shard consensus through the S-BAC protocol. This scheme requires frequent communication between input shards to achieve consensus, and the transaction abort rate increases in the case of resource conflicts, affecting transaction efficiency. Pyramid assigns the coordinator role to the b shard responsible for bridging each i shard as the output shard, which needs to wait for all i shards to agree before synchronously executing the transaction. Although this scheme reduces communication overhead, resource conflicts in any i shard will affect the overall transaction efficiency. Overall, the 2PC adopts a relatively rigorous synchronization strategy. Although transaction atomicity and consistency are effectively guaranteed, the transaction efficiency is low and cannot meet the needs of some low-latency scenarios. Moreover, the coordinator is prone to becoming a bottleneck.

2.2. Transaction splitting

The transaction splitting protocol refers to the process of dividing cross-shard transactions with multiple inputs and outputs into several single-input and single-output transactions, primarily applied in the UTXO scenario. Rapidchain proposed a method of splitting transactions to address the shortcomings of OmniLedger: clients no longer need to request asset proofs from each input committee or understand the entire network. Instead, they only need to send transactions to any committee, which then locates the output and input committees and sends transaction information through the Kademlia [16] routing mechanism and improved gossip protocol. This scheme is based on an optimistic concurrency control strategy. Although it simplifies the transaction structure, it results in more processing steps and does not provide a solution for partial shard transaction failures. In some resource competition conflicts, it cannot guarantee transaction atomicity. To solve the issues of transaction conflicts and insufficient atomicity, X-shard [7] adds a conflict detection step before cross-shard transactions, sets up a gateway account specifically for handling cross-shard transaction verification, and ensures atomicity by rolling back failed cross-shard transactions. However, the exception rollback process is complex, affecting transaction efficiency and only achieving weak consistency.

2.3. Relay transactions

The relay transactions protocol is commonly used in account-model sharded blockchains, such as Eth2.0 [17] and RChain [18], which adopt relay transaction-based solutions to handle cross-shard transactions. In such schemes, the shard where the transferring account resides constructs a relay transaction and directly sends the cross-shard transaction information to the receiving shard. The receiving shard caches the received cross-shard information and, when building a new block, selects transactions from the received cross-shard transactions, verifies them, and includes them in the block. However, relay transactions may

lead to shard congestion, delayed transaction processing, and longer transaction confirmation times. Additionally, under Monoxide's sharding strategy [11], a large number of cross-shard transactions may occur, and when the number of shards is high, almost all transactions become cross-shard. To address Monoxide's hot shard problem and reduce cross-shard transactions, BrokerChain [19] employs broker accounts to handle cross-shard transactions, utilizing fine-grained state partitioning and account segmentation to significantly reduce cross-shard transactions and the formation of hot shards, achieving strict atomicity. However, its cross-shard transaction process is a serial relay, where source shard transactions and destination shard transactions cannot be processed in parallel. Moreover, BrokerChain is specifically designed for account-based blockchains and can only process cross-shard transactions one at a time in sequence, unable to handle cross-shard transactions with multiple inputs and outputs. Furthermore, BrokerChain relies on brokers who have accounts in multiple shards to handle cross-shard transactions, and the communication and token capabilities of these brokers can easily become bottlenecks. Overall, relay transactions are suitable for account-based blockchains, but the serial execution and construction of relay transactions between shards lead to increased transaction confirmation delays.

2.4. Deterministic ordering

According to the evaluation by Prophet, a significant number of cross-shard transactions are aborted or rolled back due to resource competition conflicts, primarily because of frequent read-write conflicts during the execution of smart contracts [8]. To address the issues of read-write conflicts and coordinator single-point failures, Prophet designed a conflict-free cross-shard transaction protocol. This protocol pre-executes cross-shard transactions through a self-organized coalition of nodes from different shards to obtain prerequisite information for ordering. It then determines a trusted global order based on stateless ordering and post-validation of pre-execution results, thereby achieving conflict-free cross-shard transactions. CoCoS [6] also proposed a similar cross-shard transaction method, which relies on an ordering lock mechanism. It uses locked concurrency control to ensure transaction atomic and serializability, and ensures transaction isolation by enforcing a specific transaction locking order. The deterministic ordering protocol eliminate the need for a coordinator, effectively reducing read-write conflicts in cross-shard transactions and improving transaction efficiency. However, since state data changes with dependencies in cross-shard transactions still require serial execution, the parallelism is insufficient, especially in high-conflict scenarios where transaction congestion is prone to occur.

3. BMDS-Shard protocol design

Based on transactions splitting and relay transactions, we propose a parallel and atomically guaranteed cross-shard transaction protocol, BMDS-Shard. Firstly, the protocol decouples cross-shard transactions into synchronous transactions for both the original and target shards through beacon nodes, automatically identifying the cross-shard processing flow and assigning them higher processing priority within the shards, thereby addressing the inefficiency of relay transactions. Secondly, based on a snapshot synchronization mechanism [20] of multiple beacon nodes, the protocol ensures that state data is reliably and immediately fed back to the original shard, solving the issue of insufficient atomicity in split transactions. The specific details of the protocol will be elaborated in subsequent chapters.

3.1. Node role setting

To achieve a high-performance sharded blockchain system, we will elect and partition nodes into multiple roles based on reputation-driven mechanism [21] and the functional requirements of the system: ordinary nodes, committee nodes, block-producing nodes, signature-collecting nodes, and beacon nodes, among others. As beacon nodes play a pivotal role in the cross-shard transaction protocol, they will shoulder additional responsibilities. Hence, this section primarily focuses on the functional design of beacon nodes. In our system, a certain number of nodes within each shard are selected to serve as beacon nodes, tasked with critical cross-shard communication and coordination duties. Their main functions include:

(1) Maintaining the block data in each individual shard;

(2) In cross-shard transactions, beacon nodes synchronize the state updates of each shard across the entire network, enabling all shards to stay informed about the latest state changes in other shards;

(3) Providing verification services externally to ensure the correctness and security of cross-shard transactions and state synchronization, thereby safeguarding the overall consistency and collaborative operation of the system.

3.2. Splitting and parallelization for cross-shard transactions

As shown in Figure 2, when a user sends a cross-shard transaction θ_{raw} to a beacon node in the blockchain network through the client, the beacon node decomposes the transaction based on the shards to which the sender and receiver belong. Specifically, the beacon node splits this cross-shard transaction into two independent intra-shard transactions, θ_1 and θ_2, corresponding to the sender's shard (source shard) Shard A and the receiver's shard (target shard) Shard B, respectively. Taking the scenario of medical data sharing by insurance institutions (data users) as an example: after a customer purchases commercial health insurance and seeks medical treatment at a healthcare provider (data provider), an automatic claims process is triggered, initiating a cross-shard data-sharing transaction. The initial transaction takes the form shown in (1):

$$\theta_{raw} = \langle \text{data provider ID, data user ID, transactions data, transaction initiator's signature} \rangle \quad (1)$$

Upon receiving the transaction, if the healthcare provider's node and the insurance institution's node reside in different shards, the beacon node will decompose the transaction into the following two sub-transactions θ_1 and θ_2, and route them to the source shard and target shard respectively. The decomposed transactions are structured as follows:

$$\theta_1 = \langle \text{Type\_data provider, beacon node's signature, } \theta_{raw} \rangle \quad (2)$$

$$\theta_2 = \langle \text{Type\_data user, beacon node's signature, } \theta_{raw} \rangle \quad (3)$$

The first decomposed transaction (θ_1) will be processed in the source shard to update or synchronize the state of the sender's account. Concurrently, the second transaction (θ_2) is routed to the target shard, where it executes corresponding state updates on the recipient's account.

The two generated intra-shard transactions will then enter their respective shard's transaction queues, awaiting processing. Within each shard, transactions are ordered and processed according to the blockchain's rules. Due to the parallel processing nature of sharded blockchains, these two transactions

can independently queue in their respective shard's transaction pools and be processed concurrently, without needing to wait for the outcome of the other transaction. This mechanism significantly enhances the processing efficiency of cross-shard transactions.

Once processing is complete, the shards ensure the global state consistency of the transactions through beacon nodes or other cross-shard communication mechanisms, guaranteeing the accuracy and integrity of cross-shard transaction execution. Ultimately, the user's cross-shard transaction is considered successfully completed, with both the sender's and receiver's states being synchronously updated, ensuring the global state consistency of the blockchain.

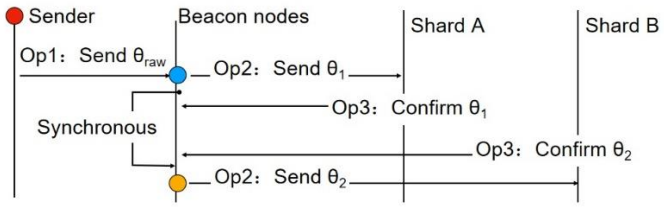

Figure 2. Splitting and parallelization for cross-shard transaction.

3.3. Priority scheduling for cross-shard transactions

3.3.1. Simple priority scheduling for cross-shard transactions

Compared to intra-shard transactions, cross-shard transactions involve more execution steps and a more complex process, which can easily lead to increased transaction confirmation delays. Therefore, we introduce transaction priority scheduling to address this issue.

If a transaction is a cross-shard transaction, it is assigned a priority at the time of creation. Priorities can be divided into several levels, with different cross-shard transactions for different services matching different priority levels. When transactions enter the transaction pool of a consensus node, the node needs to package transactions when producing a block. During the transaction packaging process, the node uses a priority sorting algorithm to sort the transactions in the pool by priority, and transactions with higher priority are packaged first. Through priority scheduling, the execution of high-priority cross-shard transactions can be accelerated.

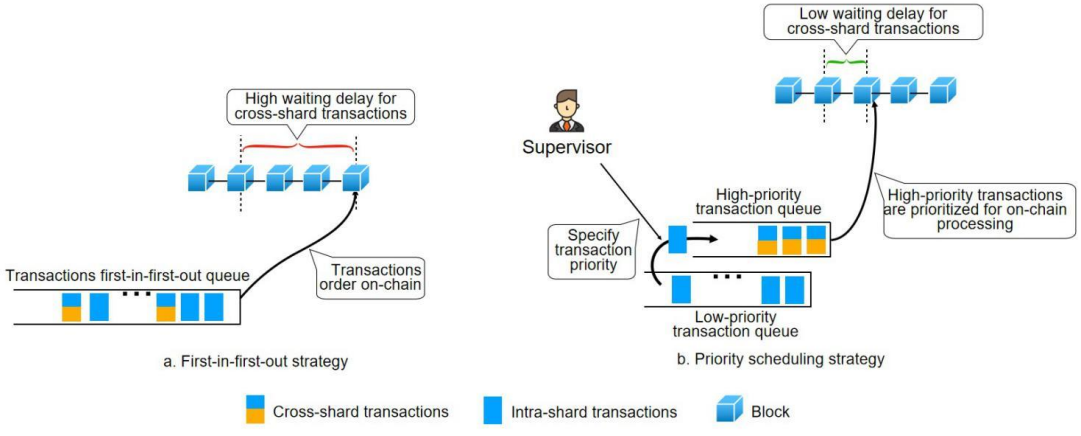

Figure 3. Priority scheduling for cross-shard transaction.

Figure 3 illustrates two methods of cross-shard transaction priority scheduling: Figure 3a uses a first-in-first-out strategy, which can result in high latency for cross-shard transactions; Figure 3b introduces a priority scheduling strategy, effectively reducing the latency for cross-shard transactions.

3.3.2. Hybrid priority scheduling for cross-shard transactions

After introducing priority scheduling, cross-shard transactions are given higher priority. To ensure that the latency of intra-shard transactions remains at a low level and to prevent the "starvation" of intra-shard transactions, a balanced strategy is needed to coordinate both cross-shard and intra-shard transactions. During the transaction packaging process, we propose a hybrid priority sorting algorithm, which not only considers the priority of transactions but also integrates factors such as transaction type and the waiting time of transactions in the transaction pool for comprehensive evaluation.

First, for cross-shard transactions, an initial sorting is performed based on their priority to ensure that high-priority cross-shard transactions are processed first. However, to prevent intra-shard transactions from being neglected for extended periods due to the high priority of cross-shard transactions, the algorithm sets an administrator-specified threshold to ensure that intra-shard transactions also occupy a certain portion of the packaging space in each block.

Algorithm 1 Hybrid Priority Scheduling Algorithm

procedure HYBRIDSCHEDULING($transactions$, β, θ)
  Input: A set of $transactions$ (TXs), where each TX t has attributes $t.type$ (either "cross_shard" or "intra_shard"), $t.priority$ (for cross-shard transactions (CTXs)), and $t.urgency$ and $t.wait\_time$ (for intra-shard TXs (ITXs)). An integer β and a float θ, which are defined by the administrator to control the maximum number of TXs per block and the threshold for the proportion of CTXs in a block, respectively.
  Output: A set P of packed TXs, prioritized according to the hybrid scheduling algorithm.
  $C \leftarrow \emptyset$ ▷ a set of CTXs
  $I \leftarrow \emptyset$ ▷ a set of ITXs
  for $t \in transactions$ **do**
    if $t.type == "cross\_shard"$ **then**
      $C \leftarrow C \cup \{t\}$
    else
      $I \leftarrow I \cup \{t\}$
    end if
  end for
  $C.sort(key = lambda\ \ x : -x.priority)$
  $I.sort(key = lambda\ \ x : (-x.urgency, -x.wait\_time))$
  $P \leftarrow \emptyset$ ▷ a set of Packed TXs
  $\alpha \leftarrow 0$ ▷ Number of CTXs in P
  while $(C \neq \emptyset$ or $I \neq \emptyset)$ and $|P| < \beta$ **do**
    if $\frac{\alpha}{\beta} < \theta$ **then**
      $c \leftarrow$ selectCross(C)
      $P \leftarrow P \cup \{c\}$
      $C \leftarrow C - \{c\}$
      $\alpha \leftarrow \alpha + 1$
    else
      $i \leftarrow$ selectIntra(I)
      $P \leftarrow P \cup \{i\}$
      $I \leftarrow I - \{i\}$
    end if
  end while
  return P
end procedure
function SELECTINTRA(I)
  return $arg\,max_{t \in I}(t.urgency \cdot t.wait\_time)$
end function
function SELECTCROSS(C)
  return $arg\,max_{t \in C}(t.priority)$
end function

Second, for intra-shard transactions, sorting can be based on factors such as urgency (e.g., reflected by transaction fees) and waiting time (transactions that have waited longer in the transaction pool should have a higher chance of being packaged to reduce latency). Even in the presence of a large number of

high-priority cross-shard transactions, intra-shard transactions can still be processed according to reasonable rules, avoiding the occurrence of "starvation." Under normal circumstances, intra-shard transactions have the lowest initial priority; if a certain latency threshold is reached, their priority is increased; if the latency is about to exceed the threshold, their priority is raised to a higher level. In contrast, cross-shard transactions initially have a higher priority.

Finally, consensus nodes can dynamically adjust these sorting parameters to adapt to changes in network load and business requirements. For example, during network congestion, the weight of transaction fees can be appropriately increased to encourage users to pay higher fees for faster confirmation. Conversely, during periods of low network activity, more consideration can be given to the waiting time of transactions to optimize overall transaction processing efficiency.

Through the above hybrid priority sorting strategy, it is possible to ensure the rapid confirmation of cross-shard transactions while maintaining a reasonable processing speed for intra-shard transactions, achieving a balanced priority between cross-shard and intra-shard transaction flows.

3.4. Snapshot synchronization mechanism

In each shard, dedicated beacon nodes are deployed, and these nodes periodically synchronize the latest block snapshots of their respective shards. This approach ensures that the state consistency is maintained across all shards in the network, preventing data inconsistencies caused by shard isolation. The data snapshot mechanism is illustrated in Figure 4.

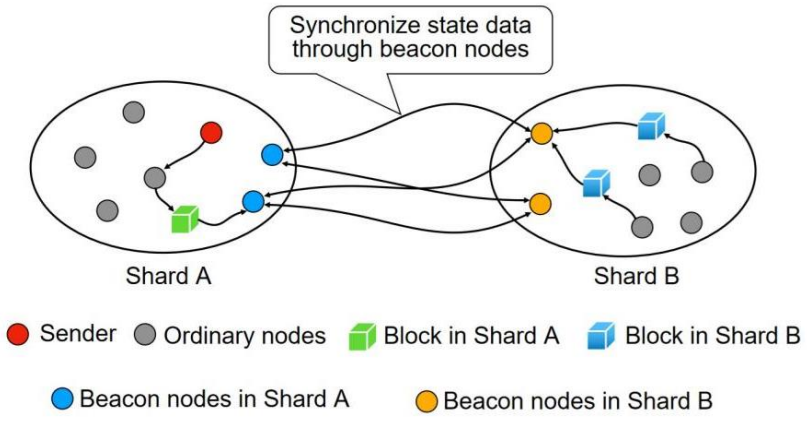

Figure 4. Schematic diagram of snapshot synchronization.

The primary responsibilities of beacon nodes include not only maintaining the block data of their own shard but also handling cross-shard data synchronization tasks. When a transaction or state update occurs in a shard, the beacon node synchronizes these changes to the beacon nodes in other shards. when cross-shard transactions or smart contract executions are involved, other shards can promptly obtain the latest state data required, ensuring the correctness and traceability of cross-shard operations.

The beacon node will send snapshots of transactions involving other shards to the corresponding beacon nodes in those shards. For example, the beacon node in Shard A will generate a snapshot of transactions involving Shard B and then send this snapshot to the beacon node in Shard B. Each time a block is generated in a shard, the beacon node sends a snapshot. Under normal circumstances, the data snapshots generated by multiple beacon nodes from the same block are identical. If the beacon node in

Shard B detects discrepancies in the data snapshots received from multiple beacon nodes in Shard A, it can extract the block headers from the snapshots and perform VRF (Verifiable Random Function) verification, thereby ensuring data consistency.

Additionally, beacon nodes are responsible for providing external verification services. They can verify transactions and state changes in other shards, ensuring the legality and consistency of cross-shard transactions. This verification service is an indispensable part of the sharded blockchain system, as it provides a trust foundation for inter-shard communication and data sharing. Through the cross-shard synchronization and verification mechanisms of beacon nodes, the entire sharded blockchain system can achieve efficient and reliable execution of cross-shard transactions while maintaining global state consistency.

4. Theoretical analysis

4.1. Atomicity analysis

Beacon nodes and committee nodes work collaboratively to provide atomicity guarantees for cross-shard transactions. As shown in Figure 2, when a beacon node receives a cross-shard transaction θ_{raw}, it splits it into two intra-shard transactions, θ_1 and θ_2. It sends θ_1 to the source shard, Shard A, while concurrently sending θ_2 to the target shard, Shard B. After Shard A executes the intra-shard transaction θ_1 and adds it to the blockchain, it sends a Confirmθ_1 message to the beacon node, indicating that θ_1 has been successfully recorded on the chain. Similarly, after Shard B executes the intra-shard transaction θ_2 and adds it to the blockchain, it sends a Confirmθ_2 message to the beacon node, indicating that θ_2 has been successfully recorded on the chain. Once the beacon node collects both Confirmθ_1 and Confirmθ_2 message, it confirms that the cross-shard transaction θ_{raw} has been successfully processed, ensuring the atomicity of the cross-shard transaction.

If the beacon node fails to collect the Confirmθ_1 message within the specified time limit, it indicates that the transmission of θ_1 may have failed due to network fluctuations. In this case, instead of rolling back, the beacon node immediately resends the corresponding intra-shard transaction θ_1 to Shard A. This ensures that Shard A will eventually receive θ_1 and improves transaction efficiency. The same process applies if the beacon node fails to collect the Confirmθ_2 message within the specified time limit.

Due to the introduction of the retry mechanism, when Shard A and Shard B receive intra-shard transactions from the beacon node, they need to verify whether the transaction has already been processed based on its hash or Nonce value. This ensures the atomicity of transaction processing and prevents duplicate execution. If the transaction has not been processed, the flow proceeds normally: the transaction is executed, and the corresponding Confirm message is sent to the beacon node. If the transaction has already been processed, it indicates an anomaly in the flow, and the previously sent Confirm message may have timed out due to network fluctuations. In this case, the transaction is not executed again, but the corresponding Confirm message is resent to the beacon node to ensure that the beacon node eventually receives confirmations for both intra-shard transactions.

For extreme situations, if the beacon node resends the transaction more than the system-defined maximum retry threshold and still does not receive a confirmation message, the system will activate the final consistency guarantee mechanism. Specifically, the beacon node will broadcast an invalidation command to both Shard A and Shard B, marking the original transaction as invalid to ensure global consistency.

If a transaction involves multiple target shards, it can be treated as multiple parallel one-to-one cross-shard transactions. The processing is similar to the transaction process between Shard A and Shard B, with no mutual interference.

4.2. Performance analysis

In BMDS-Shard, a transaction is first sent from the client to the consensus node, taking a duration of T_{Req}. The consensus node then caches it in the local transaction pool, waiting for T_{PW} to gain the right to package the transaction into a new block. Once the consensus node obtains the right to generate a new block, it initiates a new round of consensus after T_{Cons}. Finally, the consensus node spends T_{Ack} to notify the client that the transaction has been committed. Among these four time intervals, T_{Req} and T_{Ack} have relatively fixed values, which may depend solely on the network between the client and the consensus node, making them bandwidth-dependent. T_{Cons} refers to the time required for consensus nodes to reach an agreement, *i.e.*, the intra-shard consensus time overhead, and thus T_{Cons} is bounded.

For cross-shard transactions, they need to wait in the transaction pools of two shards and undergo consensus. Let the waiting time in the first shard's transaction pool be denoted as T_{PW1}, the consensus time in the first shard as T_{Cons1}, the waiting time in the second shard's transaction pool as T_{PW2}, and the consensus time in the second shard as T_{Cons2}. The confirmation delay for cross-shard transactions, L_{CST}, is expressed by the following equation (4):

$$L_{CST} = T_{Req} + \max(T_{PW1} + T_{Cons1}, T_{PW2} + T_{Cons2}) + T_{Ack} \tag{4}$$

Due to the higher priority of cross-shard transactions, T_{PW1}, T_{PW2} are smaller, while T_{Cons1} and T_{Cons2} have smaller mathematical expectations. Therefore, L_{CST} is smaller for cross-shard transactions.

For intra-shard transactions, the confirmation delay L_{ST} is shown in equation (5) below.

$$L_{ST} = T_{Req} + T_{PW} + T_{Cons} + T_{Ack} \tag{5}$$

Although the priority of intra-shard transactions is lower, the priority scheduling mechanism coordinates both cross-shard and intra-shard transactions, preventing the "starvation" phenomenon of intra-shard transactions. As a result, the T_{PW} of intra-shard transactions is bounded. Additionally, since T_{Cons} has a smaller mathematical expectation, the L_{ST} for intra-shard transactions is relatively small.

5. Experimental evaluation

In this section, we present the implementation of the BMDS-Shard prototype and separately demonstrate the results of two sets of comparative experiments, highlighting the advanced performance of BMDS-Shard.

5.1. Implementation

In order to assess the performance of BMDS-Shard, we carried out simulation experiments on the BlockEmulator platform. The transactions utilized in the simulation were sourced from historical Ethereum transaction data.

We conducted two sets of comparative experiments to comprehensively evaluate the performance of the BMDS-Shard solution. The first set of experiments demonstrates the system's real-time processing capability for cross-shard transactions by presenting confirmation latency across different shards, showing that BMDS-Shard significantly reduces transaction confirmation time between shards. Then,

by comparing the distribution density of confirmation latency for all transactions, cross-shard transactions, and intra-shard transactions, we visually demonstrate BMDS-Shard's smaller latency variation across different transaction types, with Monoxide and BrokerChain protocols included as control groups in this experimental setup. The second set of experiments categorizes transactions by priority and compares the confirmation latencies of all transactions, cross-shard transactions, and intra-shard transactions before and after the implementation of priority scheduling, demonstrating the feasibility of the priority scheduling solution.

Both sets of experiments were conducted on a local single machine, equipped with an INTEL 13900K processor (32 cores, turbo frequency 3.2GHz), 32GB of DDR5 memory, and a physical machine with a system environment of Windows 11 23H2, using a 1TB SSD hard drive. Specific experimental settings will be introduced in subsequent sections.

5.2. Performance of parallel processing of split transactions

In this experiment, a total of 300,000 transactions were executed, with a block size of 10,000 transactions per block, an injection rate of 5000 transactions per second, and a block generation interval of 0.5 seconds. The system was divided into 4 shards, each containing 4 committee nodes.

5.2.1. Time gap of cross-shard transactions confirm latency

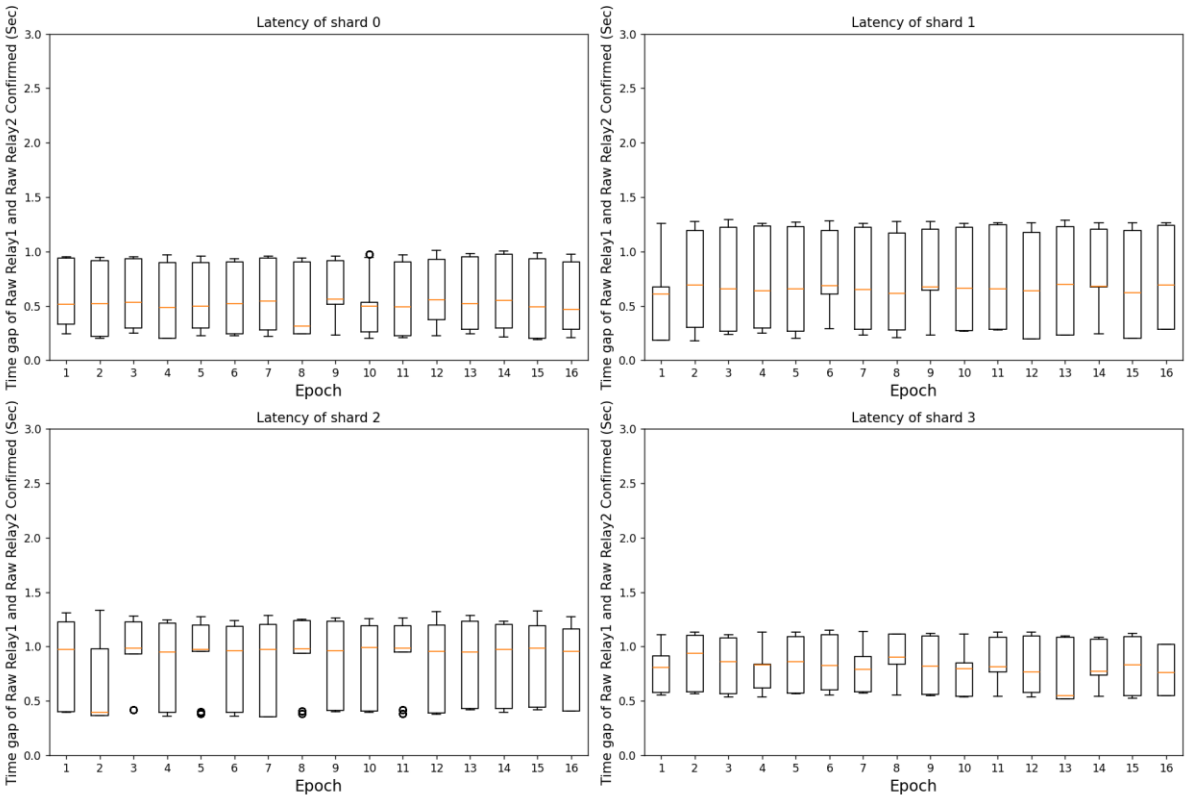

Figure 5. Time gap of cross-shard transactions confirm latency in Monoxide.

Time gap of cross-shard transactions confirm latency refers to the time difference between the completion of intra-shard consensus in the source shard and the completion of intra-shard consensus in the target shard. For example, if a cross-shard transaction takes 0.8 seconds to complete intra-shard

consensus in the source shard and 1.2 seconds to confirm the completion of intra-shard consensus in the target shard, then the time gap of cross-shard transactions confirm latency is 0.4 seconds. This is a key metric for improving the efficiency of cross-shard transactions; the smaller time gap, the higher the efficiency of cross-shard transactions.

As shown in Figures 5, 6, and 7, splitting cross-shard transactions into two parallel intra-shard transactions significantly improves processing speed. This parallel approach enables independent and simultaneous processing by different shards, eliminating sequential execution dependency. The BMDS-Shard scheme achieves superior performance with an average latency of 611.93ms, compared to 744.21ms (Monoxide) and 751.62ms (BrokerChain).

Furthermore, the system incorporates a snapshot synchronization mechanism that provides real-time monitoring of cross-shard transaction states. This mechanism maintains continuous state consistency across shards during transaction processing by enabling instantaneous status updates between participating shards. Such synchronous state tracking is critical for: (1) ensuring transaction transparency, (2) enabling precise progress monitoring, and (3) guaranteeing atomicity in distributed transaction execution.

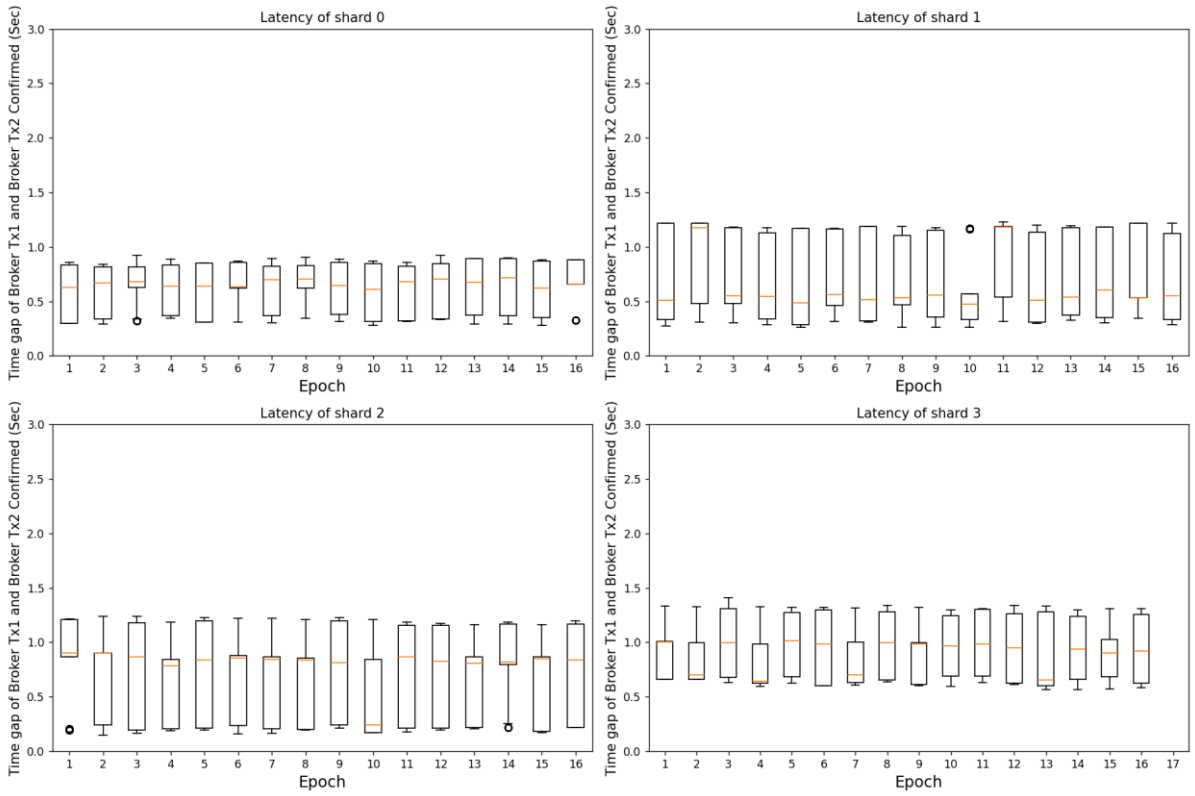

Figure 6. Time gap of cross-shard transactions confirm latency in BrokerChain.

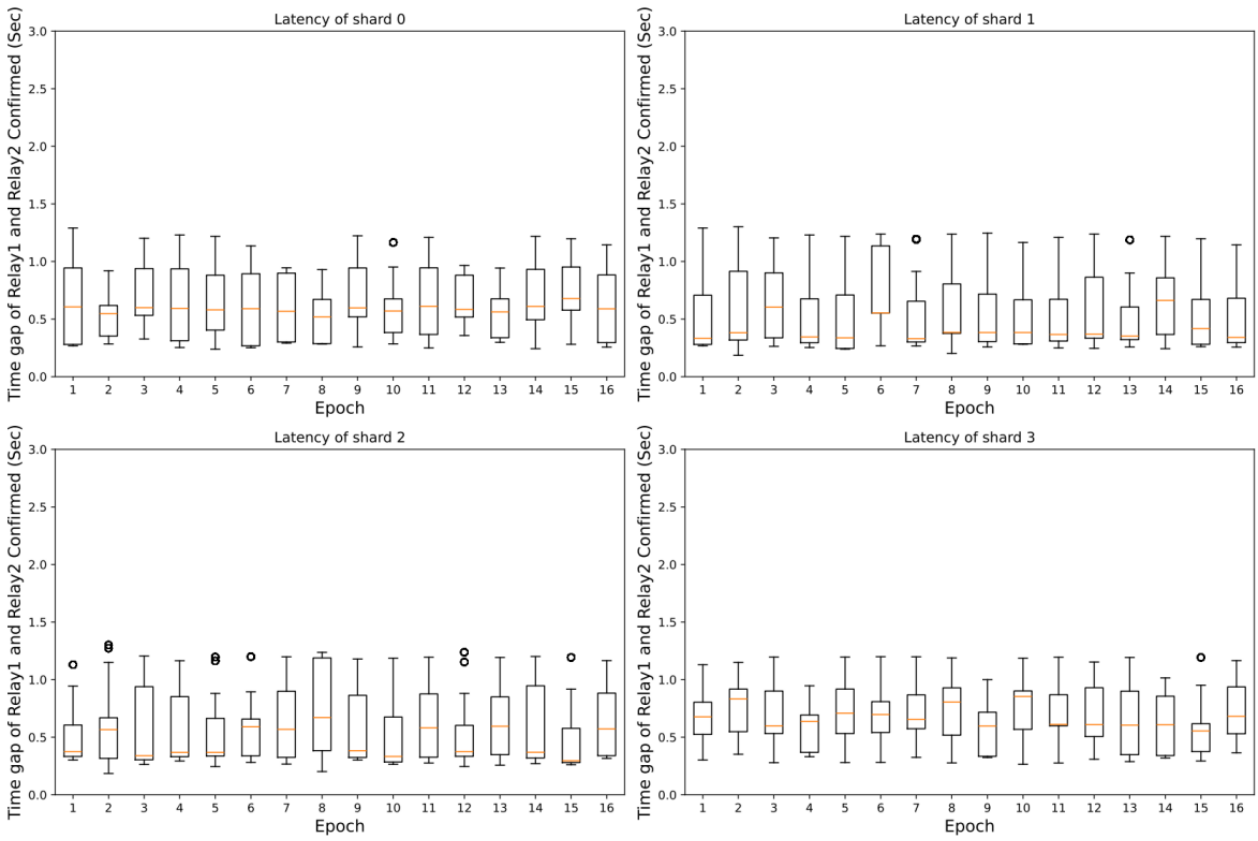

Figure 7. Time gap of cross-shard transactions confirm latency in BMDS-Shard.

5.2.2. Distribution density of confirmation latency for different types of transactions

The distribution density of confirmation latency for different types of transactions refers to the probability distribution of transaction confirmation latencies for intra-shard transactions, cross-shard transactions, and all transactions type. By comparing the distribution density of confirmation latency under different cross-shard transaction protocols, one can intuitively discern the relative merits of these protocols in terms of confirmation latency.

As shown in Figures 8, 9, and 10, significant differences are observed in the confirmation latency distributions of different transaction types across the three cross-shard transaction protocols. Experimental analysis of the latency distribution characteristics reveals that the BMDS-Shard solution achieves substantial improvements in average cross-shard transaction confirmation latency, demonstrating 65.82% and 65.06% reductions compared to the mainstream relay transaction protocols Monoxide and BrokerChain, respectively. Importantly, the confirmation latencies for both cross-shard and intra-shard transactions maintain comparable levels under BMDS-Shard, with the majority of transactions (including both types) being processed within 1.5 seconds. These results confirm that the parallel processing approach effectively reduces transaction confirmation times while ensuring high efficiency and real-time performance for all transaction types, thereby meeting the system's demanding low-latency requirements.

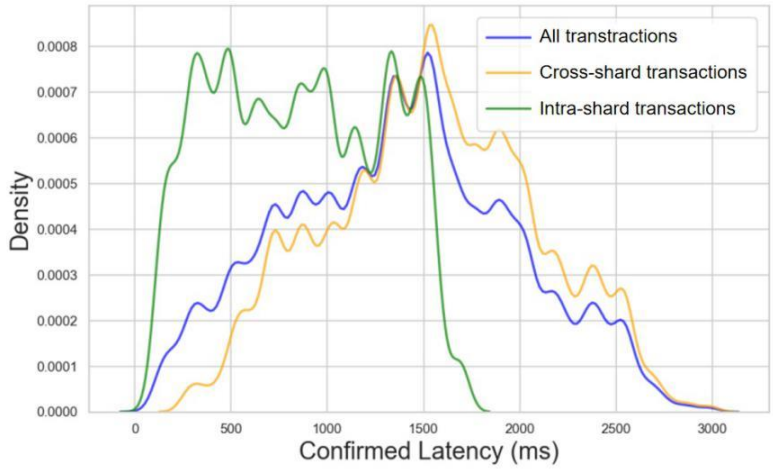

Figure 8. Monoxide confirmation latency distribution density.

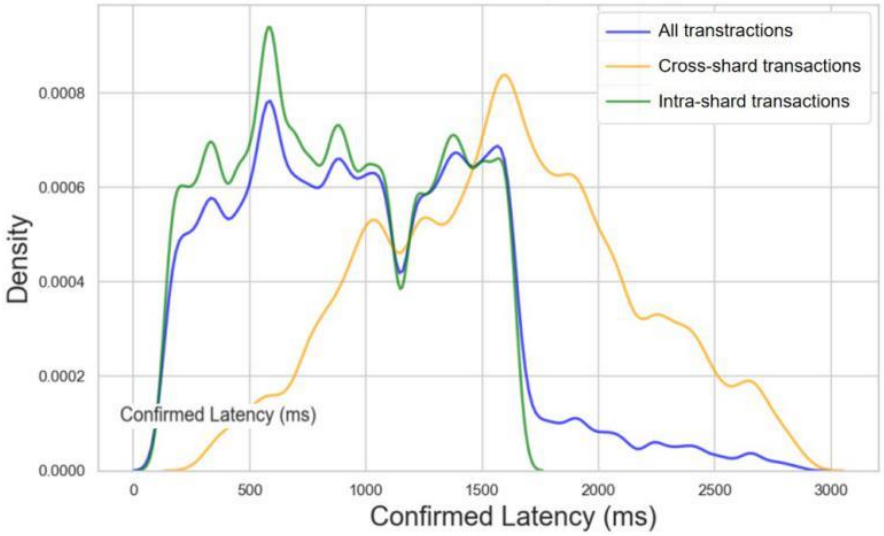

Figure 9. Brokerchain confirmation latency distribution density.

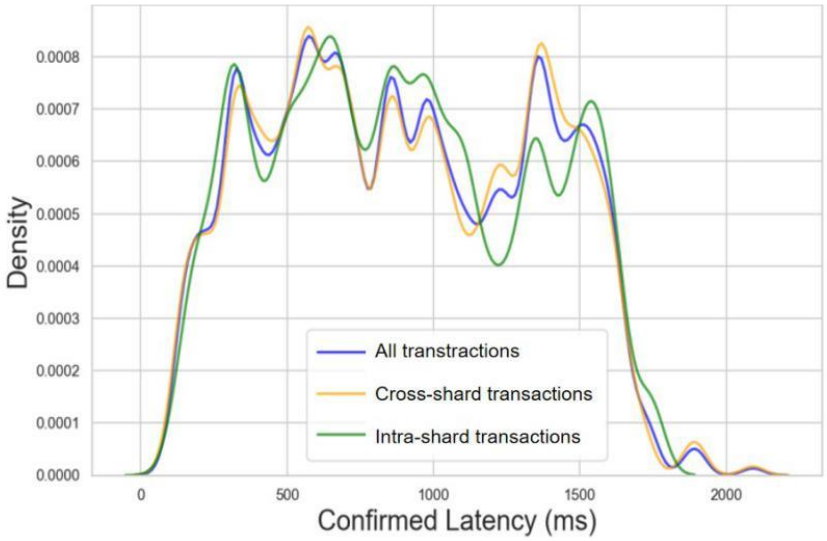

Figure 10. BMDS-Shard confirmation latency distribution density.

5.3. Performance of cross-shard transaction scheduling optimization

This experiment is designed to achieve two key objectives: first, to empirically demonstrate that cross-shard transaction scheduling can effectively reduce confirmation latency by granting prioritized processing to cross-shard transactions that would otherwise incur longer delays; second, to verify that while prioritizing cross-shard transactions, the scheduling mechanism maintains fair processing for lower-priority intra-shard transactions without indefinite postponement, thereby keeping intra-shard transaction latency within acceptable thresholds.

Furthermore, we recognize that block size, block generation interval, transaction processing speed, and cross-shard transaction ratio constitute critical parameters for conducting performance experiments on cross-shard transaction scheduling optimization schemes. We conducted tests in two simulated environments: one where the network and hardware conditions were relatively favorable, named as environment I, allowing the system to support large blocks, low block intervals, and a high proportion of cross-shard transactions; and another where the network and hardware conditions were less ideal, named as environment II, resulting in weaker system support capabilities and a lower proportion of cross-shard transactions. In these two environments, the blockchain's processing capability demonstrates that the network processing capacity in Environment I is approximately 17 times that of Environment II.

5.3.1. Performance of Environment I

In this experiment, a total of 600,000 transactions were executed, with a block size of 10,000 transactions per block, an injection rate of 2000 transactions per second, and a block interval of 1.5 seconds. The system was divided into 4 shards, each containing 4 committee nodes.

Within the priority scheduling scheme, transaction priorities were categorized into levels 0, 1, 2, and 3, with a higher number indicating a higher priority (priority level 0 denotes the lowest). There were 420,000 intra-shard transactions at priority level 0, accounting for 70% of the total. Cross-shard transactions at priority levels 1, 2, and 3 numbered 60,000 each, collectively making up 30% of the total, simulating an application scenario where medical resources serve a significant number of non-local residents. This setup was compared against a no-priority scheduling scheme (No priority).

Figure 11 compares the intra-shard transaction latency with the priority scheduling scheme (Priority = 0) and the intra-shard transaction latency without the priority scheduling scheme (No priority). Since cross-shard transactions account for 30% in the experiment, the latency of intra-shard transactions with the priority scheduling scheme (Priority = 0) is slightly higher than that of intra-shard transactions without the priority scheduling scheme (No priority), but it still remains at a relatively low level. This is because the priority scheduling scheme coordinates the priority of both cross-shard and intra-shard transaction processes. When there are too many cross-shard transactions, the algorithm still considers packaging a portion of intra-shard transactions, allowing intra-shard transactions to be processed according to reasonable rules and avoiding the occurrence of "starvation".

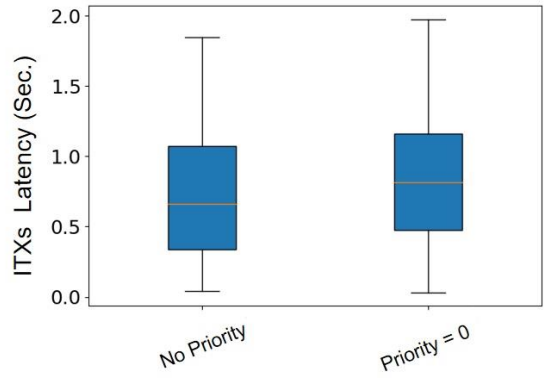

Figure 11. Transaction confirmation latency for no-priority and lowest-priority.

Figure 12 compares the latency of cross-shard transactions with priority scheduling (Priority=1, 2, 3) and those without priority scheduling (No priority). As shown in Figure 5.8, the average latency for non-prioritized transactions (No priority) is 808.98 ms, while the average latencies for prioritized transactions (Priority=1, 2, 3) are 781.55 ms, 697.56 ms, and 607.24 ms, respectively. The results clearly demonstrate that cross-shard transactions with priority scheduling (Priority=1, 2, 3) exhibit lower latency than those without priority scheduling (No priority), with higher priority levels corresponding to progressively lower latency. This performance improvement can be attributed to the priority scheduling mechanism's ability to preferentially process higher-priority cross-shard transactions.

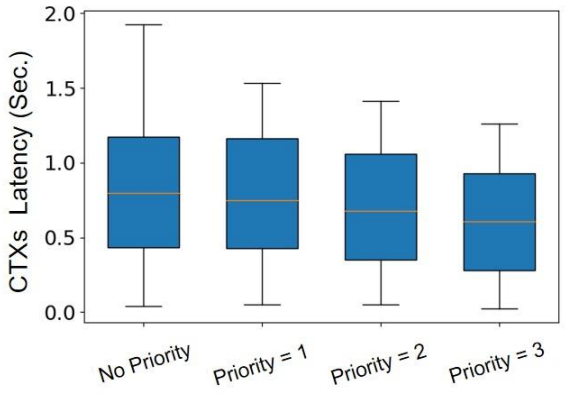

Figure 12. Transaction confirmation latency for no-priority and high-priority.

5.3.2. Performance of Environment II

We simulated scenarios with low network bandwidth and low hardware configuration by varying the block size, transaction injection rate, and block interval. In this experiment, we set the total number of executed transactions to 600,000, with a block size of 2000 transactions per block, an injection rate of 2000 transactions per second, and a block interval of 5 seconds. The system was divided into 4 shards, each containing 4 committee nodes.

In the priority scheduling scheme, transaction priorities were categorized into levels 0, 1, 2, and 3, with a higher number indicating a higher priority. Priority level 0 was assigned to intra-shard transactions, with 540,000 such transactions in this experiment, accounting for 90% of the total. Cross-shard transactions at priority levels 1, 2, and 3 numbered 20,000 each, collectively making up 10% of the total,

simulating a scenario where medical resources primarily serve local residents. This setup was compared against a no-priority scheduling scheme (No priority) for performance evaluation.

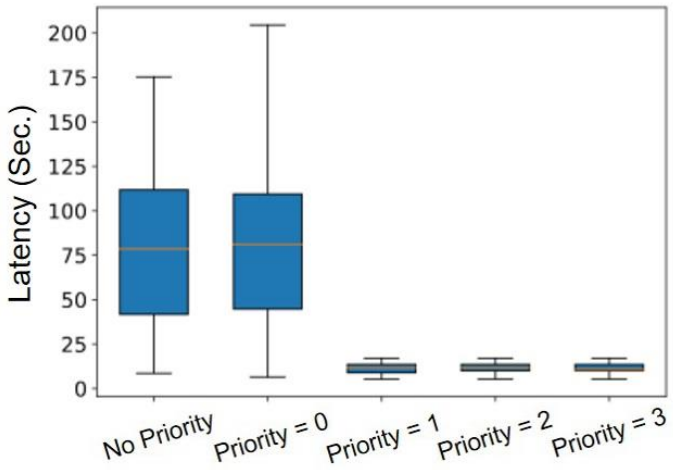

Figure 13. Transaction confirmation latency for no-priority and priority.

As clearly demonstrated in Figure 13, under low-performance hardware conditions, the average transaction latency without priority scheduling (No priority) reaches 79.53 seconds. In contrast, transactions with priority scheduling (Priority = 0, 1, 2, 3) show markedly different performance characteristics: while Priority = 0 transactions exhibit comparable latency (82.09s), higher priority levels achieve remarkable improvements, with Priority = 1 at 11.35s, Priority = 2 at 11.39s, and Priority = 3 at 11.34s — representing an 85.7% reduction in latency compared to the non-prioritized scheme.

The experimental results in Sections 5.3.1 and 5.3.2 show that the priority scheduling scheme reduces cross-shard transaction confirmation latency. By comparing the experimental data between Environment I and Environment II, it is found that the performance of hardware network environment has a certain impact on the optimization effect of cross-shard transactions. In Environment II with poorer hardware conditions, the reduction in cross-shard transaction confirmation latency is more significant. In Environment I where transaction processing speed is faster and queuing latency accounts for a smaller proportion of total confirmation latency, although the improvement in confirmation latency is less pronounced than in Environment II, the proposed method still achieves a reduction in cross-shard transaction latency from 808.98ms before optimization to 607.24ms, with a maximum reduction of 33.22%. Furthermore, the reduction will increase as the hardware network environment deteriorates, which supports significant performance improvement for some poorly performing cross-shard transactions through priority elevation. Meanwhile, the intra-shard transaction latency increases from 702.9ms to 815.2ms, a 15.98% increase that remains within acceptable limits, and this increase will further decrease as the hardware network environment worsens. The experimental results demonstrate that this method can effectively improve cross-shard transaction processing efficiency under different hardware network environments, with particularly outstanding advantages in low-performance hardware network environments closer to practical business scenarios, thus verifying the engineering applicability of the solution.

6. Conclusion

Current research on cross-shard transactions face challenges in balancing efficiency and atomicity. The BMDS-Shard protocol propsed a parallel and atomicity-guaranteed cross-shard transaction protocol based on relay and split transactions. By utilizing beacon nodes and a snapshot synchronization mechanism, it achieves atomicity and consistency in cross-shard transactions, effectively enhancing their efficiency. Moreover, the cross-shard transaction scheduling optimization scheme further reduces the latency of cross-shard transactions while also accommodating intra-shard transactions and other types. We will integrate BMDS-Shard with real-world blockchain networks for medical data sharing and evaluate its performance in practical deployments.

Acknowledgments

This work was supported by the Science and Technology Development Program of Two Districts in Xinjiang Province, China under Grant No. 2024LQ03004, and the National Key R&D Program of China under Grant No. 2022YFB2703301.

Authors' contribution

Conceptualization, C.H.T., Z.J.; methodology, C.H.T., Z.J., L.X.F., L.Z.L.; software, Z.J., L.X.F., L.Z.L.; validation, C.H.T., Z.J., L.X.F., L.Z.L.; formal analysis, Z.J., L.X.F., L.Z.L.; investigation, Z.J., L.X.F., L.Z.L.; resources, Z.J., L.Z.L.; data curation, Z.J., L.Z.L.; writing—original draft preparation, C.H.T., Z.J., L.X.F., L.Z.L.; writing—review and editing, S.Q.N., H.H.W.; visualization, Z.J., L.X.F., L.Z.L.; supervision, S.Q.N., H.H.W.; project administration, C.H.T., S.Q.N., H.H.W.; funding acquisition, S.Q.N., H.H.W. All authors have read and agreed to the published version of the manuscript.

Conflicts of interests

The authors declared that they have no conflict of interest.

References

[1] Zhang R, Xue R, Liu L. Security and privacy for healthcare blockchains. *IEEE Trans. Serv. Comput.* 2021, 15(6):3668–3686.

[2] Huang H, Yue Z, Peng X, He L, Chen W, *et al.* Elastic resource allocation against imbalanced transaction assignments in sharding-based pennissioned blockchains. *IEEE Trans. Parallel Distrib. Syst.* 2022, 33(10):2372–2385.

[3] Xie J, Yu FR. Huang T, Xie R, Liu J, *et al.* A survey on the scalability of blockchain systems. *IEEE Network* 2019, 33(5): 166–173.

[4] Cai X, Geng S, Zhang J, *et al.* A sharding scheme-based many-objective optimization algorithm for enhancing security in blockchain-enabled industrial internet of things. *IEEE Trans. Ind. Inf.* 2021, 17(11):7650–7658.

[5] Shi J, Zhang A, Bai XY, Cai HQ, Liu XZ. Survey on performance optimization technologies of distributed ledger system (In Chinese). *J. Softw.* 2023, 34(10): 4607–4635.

[6] Que Q, Chen Z, Zhang Z, Yang Y, Zhou A. A coordinator-free cross-shard transaction processing for sharded permissioned blockchain (In Chinese). *J. Comput. Res. Dev.* 2023, 60(11):2469–2488.

[7] Xu J, Ming Y, Wu Z, Wang C, Jia X. X-shard: optimistic cross-shard transaction processing for sharding-based blockchains. *IEEE Trans. Parallel Distrib. Syst.* 35(4):548–559.

[8] Hong Z, Guo S, Zhou E, Zhang J, Chen W, *et al*. Prophet: conflict-free sharding blockchain via byzantine-tolerant deterministic ordering. In *IEEE INFOCOM 2023 - IEEE Conference on Computer Communications*, New York City, USA, May 17–20, 2023, pp. 1–10.

[9] Zamani M, Movahedi M, Raykova M. RapidChain: scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, Torond, Canada, October 15–19, 2018, pp. 931–948.

[10] Liu A, Liu Y, Wu Q, Zhao B, Li D, *et al.* CHERUBIM: A secure and highly parallel cross-shard consensus using quadruple pipelined two-phase commit for sharding blockchains. *IEEE Trans. Inf. Forensics Secur.* 2024, 19:3178–3193.

[11] Wang JP, Wang H. Monoxide: Scale out blockchains with asynchronous consensus zones. In *16th USENIX Symposium on Networked Systems Design and Implementation. Boston: USENIX Associatio*n, Boston, USA, February 26–28, 2019, pp. 95–112.

[12] Huang H, Ye G, Yang Q, Chen Q, Yin Z, *et al*. BlockEmulator: an emulator enabling to test blockchain sharding protocols. *IEEE Trans. Serv. Comput.* 2025, 18(2):690–703.

[13] Kokoris-Kogias E, Jovanovic P, Gasser L, Gailly N, Ford B, *et al*. Omniledger: a secure, scale-out, decentralized ledger via sharding. In *Proceedings of the 2018 IEEE Symposium on Security and Privacy (SP)*, San Francisco, USA, May 20–24, 2018, pp. 583−598

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

[15] Hong Z, Guo S, Li P, Chen W. Pyramid: a layered sharding blockchain system. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, May 10–13, 2021, pp. 1–10.

[16] Maymounkov P, Mazières D. Kademlia: a peer-to-peer information system based on the XOR Metric. In *Peer-to-Peer Systems*, Cambridge, USA, 2002, pp. 53–65.

[17] Shard chains. Danksharding. Available: https://ethereum.org/en/eth2/shard-chains/ (accessed on 16 November 2024).

[18] Eykholt E, Meredith LG, Denman J. RChain architecture documentation. Available: https://architecture-docs-zh-cn.readthedocs.io/_/downloads/zh-cn/stable/pdf/ (accessed on 16 November 2024).

[19] Huang H, Peng X, Zhan J, Zhang S, Lin Y, *et al*. BrokerChain: a cross-shard blockchain protocol for account/balance-based state sharding. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, May 2–5, 2022, pp. 1968–1977.

[20] Qi X. S-Store: a scalable data store towards permissioned blockchain sharding. In *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, London, UK, May 2–5, 2022, pp. 1978–1987.

[21] Jiang N, Bai F, Huang L, An Z, Shen T. Reputation-driven dynamic node consensus and reliability sharding model in IoT blockchain. *Algorithms* 2022, 15(2): 28−50.