Presto: Optimizing Cross-Shard Transactions in Sharded Blockchain Architecture

Qiuyu Ding*, Rongkai Zhang*, Shenglin Yin*, PengZe Li*, Shengjie Guan*, Zhen Xiao*, Jieyi Long†
*School of Computer Science, Peking University
†Theta Labs, Inc.
{dingqiuyu, rkzhang, yinsl, guanshengjie}@stu.pku.edu.cn, {lipengze, xiaozhen}@pku.edu.cn, jieyi@thetalabs.org
Corresponding author: Zhen Xiao

***Abstract*—Blockchain sharding technology has been used to enhance the scalability of blockchain systems. As the number of shards increases, the high latency inherent in cross-shard transactions gradually becomes a bottleneck, hindering improvements in overall system efficiency. Therefore, reducing the latency of cross-shard transactions is significantly important. However, existing mechanisms for handling cross-shard transactions fail to minimize the latency of cross-shard transactions and have not fully used the bandwidth available within shards. In this paper, we introduce Presto, a protocol designed for the account-state-based blockchain, which reduces the latency of handling cross-shard transactions. Presto leverages the concept of optimistic pre-execution along with pending tree to optimize cross-shard transaction processing. Presto also employs predistribution of cross-shard transactions with Erasure Coding to efficiently utilize bandwidth resources. We have developed an prototype and conducted extensive experiments on a cloud platform. The evaluation results indicate that Presto surpasses existing solutions in terms of system throughput, transaction confirmation latency, and mempool queue size, demonstrating Presto's potential to significantly improve blockchain scalability and user experience.**

***Index Terms*—Blockchain, Sharding Consensus, Byzantine Fault Tolerance**

I. Introduction

As blockchain technology gains widespread adoption across various industries, the quality of service (QoS) it provides becomes increasingly critical, particularly for applications that are highly sensitive to delays. Transactions that fail to complete within expected time not only compromise the user experience but also hinder critical business processes, potentially disrupting normal operations. When the incoming transaction volume exceeds the processing capacity of the blockchain, the accumulation of transactions in a queue causes longer processing times and increased latency. Consequently, addressing scalability and enhancing transaction processing speed have emerged as focal points in the evolution of blockchain technology to improve service quality and meet the high-performance demands of various sectors.

Sharding technology has emerged as a promising solution for scaling blockchain systems. It allows for parallel transaction processing across shards to reduce storage and computational burden. Transactions executed within a single shard, referred to as intra-shard transactions, benefit from efficient processing and low latency. However, cross-shard transactions, which depend on states across multiple shards, introduce significant challenges. These challenges manifest as increased latency and degraded system throughput compared to intra-shard transactions, hindering scalability and overall performance. Statistics [1] indicate that over 96% of transactions in sharding blockchain systems are cross-shard, which can significantly impact overall system performance. This high proportion of cross-shard transactions has prompted research efforts to focus on two main aspects: reducing the ratio of cross-shard transactions [2] and optimizing the cost of processing cross-shard transactions. Our work focuses on reducing the cost of cross-shard transaction processing and optimizing latency.

Existing approaches to address cross-shard transactions can be broadly categorized into relay and two-phase commit (2PC) methods. Relay-based solutions, such as Monoxide [3], execute transactions sequentially: the relay transactions are first processed in the source shard and then sent to the target shard to complete the processing. This sequential process requires both shards to wait for two rounds of consensus, resulting in higher latency and lower throughput, potentially negating the scalability benefits of sharding. BrokerChain [4] proposes using third-party clients as relays for cross-shard transactions to alleviate the load on leader nodes. However, this approach lacks sufficient decentralization, introducing reliability issues and susceptibility to censorship attacks, with varying service quality affecting user experience. Alternatively, two-phase commit-based mechanisms needs many rounds of communication between involved shards, leading to long processing time and potential unavailability of certain states during the consensus rounds. The two-phase commit (2PC) protocol such as Pyramid [5] comprises a prepare phase and a commit phase, each involving Byzantine Fault Tolerant (BFT) consensus [6] among participating shards. The prepare phase guarantees input availability, while the commit phase ensures transaction validity. Upon commitment of input availability in the prepare phase, access to the input is restricted until the commit phase determines transaction validity. This process leads to exceptionally high latency, which is intolerable for users.

In this paper, we present Presto, a novel mechanism aims at improving the latency of cross-shard transactions. The fundamental idea is to handle cross-shard transactions proactively and optimistically, while distributing the bandwidth load across nodes in shard to minimize the leader node's burden. It is worth noting that numerous challenges exist in the design process. First, proactive execution poses security risks. Overly optimistic processing may lead to the execution of unauthenticated transactions on the target shard, enabling malicious actors to blindly consume resources. Ensuring the correctness of cross-shard transactions during execution is a significant challenge. We discover that transactions that pass at least the first round of PBFT [6] consensus can be guaranteed to be correct. Therefore, the target shard can pre-excute cross-shard transactions after receiving the first-round consensus result of the source shard to shorten the consensus process and improve the latency of cross-shard transactions. Second, ensuring reliable message transmission between shards during the transfer process while efficiently utilizing idle bandwidth across nodes in each shard is crucial for guaranteeing the atomicity of cross-shard transactions. This poses a significant challenge that requires designing a new cross-transaction relay mechanism to achieve both reliability and bandwidth efficiency.

The main contribution of this paper are summarized as follows:

- **Low-Latency scheme for cross-shard transaction processing:** We provide a detailed process analysis for cross-shard transactions and introduce a new insight to reduce the cross-shard transaction processing latency. We decouple the consensus process and design a pending tree data structure to increase transaction concurrency.
- **Efficient cross-shard transactions transmission mechanism:** Based on our analysis of bandwidth for shard transaction transmission, we pre-distribute transaction transmission using erasure coding (EC) [7] and design a succinct and reliable transmission protocol.
- **Theoretical analysis:** We conduct a theoretical analysis of the proposed cross-shard transaction processing mechanism, focusing on security and transaction atomicity. Several potential malicious behaviors are discussed, and corresponding defense mechanisms are explained to demonstrate the robustness of our approach.
- **System implementation:** Finally, we have implemented an emulator version of Presto based on BlockEmulator [8], a popular sharding blockchain prototype. Results indicate that Presto can improve latency by 66%, 54% and the TPS by 63%, 48% compared to BrokerChain and Monoxide in a given environment.

II. Related Work

A. Sharding

Sharding has emerged as a promising solution to address the scalability and performance limitations of traditional blockchain systems [9] [10]. By partitioning transactions across multiple shards, which operate independently and process transactions in parallel, sharding significantly enhances throughput of transactions [11]. Traditional blockchain sharding mechanisms can be categorized based on their account models, such as the account balance model and the Unspent Transaction Output (UTXO) model [12]. Elastico [13], OmniLedger [14], and RapidChain [1] are notable implementations of sharding using the UTXO model, demonstrating the feasibility of distributing the ledger's state across multiple shards while maintaining network integrity and security. OptChain [15] further evolves sharding by implementing a new method for account state partitioning, potentially reducing the overhead associated with cross-shard transactions and simplifying state management. Brokerchain [4] protocol employs Metis algorithm into fine-grained state partitioning and account segmentation to balance workloads across different shards, mitigating the 'hot shard' problem.

B. Cross-Shard Transaction Processing

Several approaches have been proposed to handle cross-shard transactions. Monoxide [3] introduces an account balance-based sharding system, addressing the complexities of cross-shard transactions through a relay mechanism that ensures efficient processing and maintains consistency and atomicity. Another relay-based method [16] [17], Brokerchain [4] uses a third-party broker as an intermediary for token transfers. The 2PC-based methodology, exemplified by the Pyramid [5] system, allows shards to overlap, enabling some nodes to belong to more than one shard and facilitating the direct processing of cross-shard transactions. Pyramid consolidates cross-shard transactions into a single block, processed by nodes situated in the intersection of the involved shards, enhancing overall system performance. CHERUBIM [18], which is also based on the two-phase commit (2PC) protocol, employs pipeline technology to further enhance the throughput of cross-shard transactions. However, it does not effectively address the issue of prolonged latency. The Benzene [19] architecture presents a cooperation-based mechanism that separates transaction recording and consensus execution. In this dual-chain structure, consensus on proposer blocks is achieved through the concerted efforts of multiple shards, involving transaction recording, verification in a Trusted Execution Environment (TEE) [20], vote generation, and block confirmation. However, the requirement for nodes to operate within a TEE (Trusted Execution Environment) imposes limitations on Benzene.

While each approach has unique attributes, they collectively contribute to addressing the burden of cross-shard transactions and represent the diverse strategies being explored to enhance scalability, efficiency, and security in sharding systems. However, these approaches fail to fundamentally reduce the processing requirements of cross-shard transactions, leaving the issue of increased transaction latency largely unaddressed. In contrast, Presto can expedite cross-shard transaction processing, leading to a significant reduction in latency.

III. System Model

Presto adopts an account/balance model. Each shard maintains a set of account state, ensuring efficient parallel processing and scalability. Each set of state is completely separate across the shards and all of them form a unified global ledger state.

Presto employs a partially synchronous peer-to-peer network model [21] for inter-node communication. In this model, nodes are connected through a network that allows messages to be relayed from one node to another within the network's boundaries. There exists an upper bound Δ on the message transmission delay. This bound is not known a priori but is guaranteed to exist and hold after some unknown time GST (Global Stabilization Time) [21]. The partially synchronous model captures the realistic behavior of networks, where message delays may vary and be unpredictable, but are eventually bounded.

We hypothesize that the system comprises N full nodes, with a maximum of f nodes being malicious. The number of nodes required for consensus is denoted as $N >= 3f + 1$ as other byzanting resistance blockchain systems [22]. The Presto protocol utilizes PBFT for intra-shard consensus, ensuring agreement within each shard. Presto facilitates compatibility with alternative consensus algorithms like HotStuff [23] [24], allowing for adaptability to different blockchain ecosystems. In Presto, malicious full nodes are unable to forge signatures but they retain the ability to execute a range of attacks. These potential attacks include delaying or omitting messages and sending fake messages. To resist an Eclipse attack [25], a node must connect to more than f other nodes, where f represents the maximum number of malicious nodes in the network. Malicious can also deploy adaptive attacks [26] targeting individual shards, potentially compromising the single shard's security requirements. To resist such attacks, it is imperative to implement a strategy that includes random shard assignment and a robust reconfiguration scheme. Several established methods, such as those based on Proof-of-Work (PoW) puzzles, referenced in works like [1] and [14], alongside the cuckoo rule [27], have been proven effective in protecting sharding systems against adaptive attacks. So we can assume that every shard is honest. It is important to note that the research on this assignment mechanism is orthogonal to the main contribution of this paper.

IV. System Overview

Presto solves the long latency problem of handling cross-shard transaction by optimizing consensus process and expediting cross-shard transaction dissemination through a series of innovative mechanism. Presto's workflow consists of four main phases: Source Execution Phase, Reliable Relay Phase, Target Execution Phase and Final Redemption Phase. In each shard of Presto, a randomly elected leader initiates consensus epochs by proposing blocks containing both cross-shard and intra-shard transactions. Leader in Presto leverages erasure coding to split block into small pieces. Then leader distributes the pieces to nodes in shard to optimize network bandwidth utilization during proposing phase. During proposing phase of consensus in the source shard, the nodes in the source shard also transmit the received pieces of cross-shard transactions to target shard. After cross-shard transactions are validated on the source shard, the nodes in source shard send confirm proof to the target shard. Upon collecting enough pieces of cross-shard transactions with confirm proof, target shard nodes reconstruct the complete transactions and add them to their mempool.

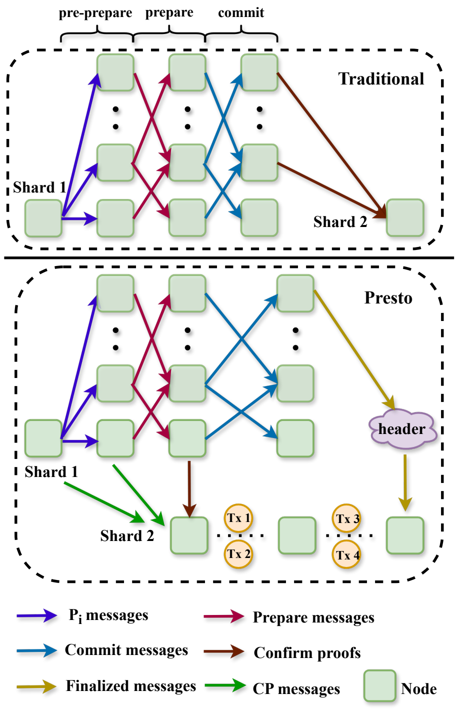

Figure 1. Optimization of consensus layer in processing cross-shard transactions.

Deviating from traditional approaches that necessitate all responding shards' finalization before token is available to be spent on target shard, Presto introduces the innovative **Pending Tree** mechanism to pre-utilize the unfinalized token in target shard. The pending tree is a buffering sketch that temporarily holds unfinalized transactions, enabling immediate token transfers with temporary account state. The nodes of each shard maintain block headers from other shards to validate and finalize cross-shard transactions.

V. Design of Presto

The Presto protocol deconstructs the process of cross-shard transactions into four phases, including Source Execution Phase, Reliable Relay Phase, Target Execution Phase and Final Redemption Phase. The four phases of the Presto protocol, while started sequentially, exhibit a degree of parallelism in their execution. However, the intra-shard transactions only need to be executed in Source Execution Phase. In the following sections, we will describe the details of each phase, providing a comprehensive understanding of the Presto protocol's cross-shard transaction mechanism.

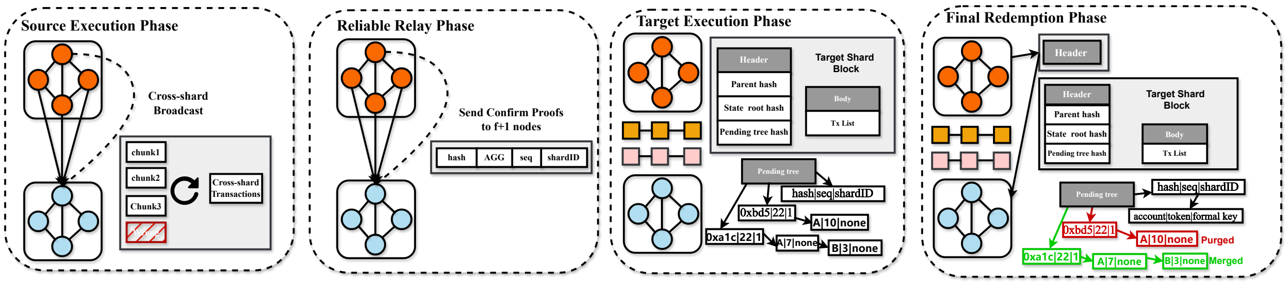

Figure 2. Overview of Presto.

A. *Source Execution Phase*

During the Source Execution Phase, the leader node of the source shard packages transactions and starts PBFT consensus, ultimately leading to their on-chain commitment. For intra-shard transactions, this commitment within the Source Execution Phase marks their completion. However, cross-shard transactions necessitate an additional submission and finalization step within the target shard before they are considered finalized.

At the beginning of the Source Execution Phase, the leader node of each shard starts the propose operation of PBFT consensus. The cross-shard transactions and intra-shard transactions belonged to one shard are packed into a block by the leader node of the shard. Then the leader node distributes the block to the normal nodes of the shard, as depicted in Line 6, Algorithm 1.

In Presto, we introduce an efficient distribution scheme for block propagation with EC technique in the leader node's proposal operation. The block is splited into N pieces, where N is the number of the nodes in the shard. And the structure of each piece is outlined as follows:

$$\begin{aligned} P_i = \{&\text{view}, \text{hash}, B_{\text{intra}}(i), \sigma_{id}, \text{seq}, \text{Proofs}, \text{shardID} \\ &\{\ B^1_{\text{cross}}(i), B^2_{\text{cross}}(i), \ldots, B^K_{\text{cross}}(i)\ \}\} \end{aligned}$$

Here, i denotes the node id within the shard. The node id can be generated by arranging the nodes within the shard in descending order based on their public addresses. The σ_{id} denotes the signature signed by the leader node. The K denotes the total number of the target shard. The seq refers to the block number, which identifies the height of a block in the blockchain. $hash$ refers to the hash of the proposed block. view refers to the current view of this round consensus. Proofs represents the list of proof genrated by other shards, which will be utilized to prove correctness of some transactions in this shard, as discussed in Section V-C. shardID represents ID of the source shard. The leader node employs Erasure Coding (EC) to partition batched intra-shard transactions into m segments, with $B_{\text{intra}}(i)$ representing the segment intended for the i-th node. To tolerate the potential loss of segments, the erasure code should be (j, m), which means that at least j out of m segments are required to recover the original message. Here, j should satisfy the condition $j \leq m - \lceil \frac{f \cdot m}{N} \rceil$, where f represents the maximum number of faulty nodes tolerated by the system [28].

Regarding cross-shard transactions, the leader node arranges them into distinct batches based on the target shards. Subsequently, EC coding is applied to segment each batch into m parts, denoted as $B^k_{\text{cross}}(i)$, where k represents the target shard id and i is the segment index.

Each node in source shard is responsible for pre-dissemination the cross-shard transactions data to target shards. Upon receiving pre-prepare message P_i, normal node i extracts $B^k_{\text{cross}}(i)$ and constructs Cross-Shard Packages $CP_{k,i}$ to send them to nodes in the corresponding shard. $CP_{k,i} = \{\text{hash}, B^k_{\text{cross}}(i), \text{Height}\}$, where hash is from P_i, indicating the original block. Height is extracted from P_i to ensure consistency. Then normal node i sends each $CP_{k,i}$ to the target shard k. Enough $CP_{k,i}$ can be sent to nodes of target shard k and be recovered to complete batch of cross-shard transactions B_{cross}. Besides, normal node i still needs to broadcast P_i to the nodes in its shard. If nodes within the source shard have collected the enough P_i, then they reconstruct the original block, validate the transactions' legitimacy and broadcast the prepare vote to all nodes in the source shard, as depicted in Line 9-16, Algorithm 1. Then upon receiving at least $2f + 1$ prepare votes, each node in source shard broadcasts commit vote. If the nodes receive at least $2f + 1$ commit votes, the nodes commit the transactions into the ledger, as depicted in Line 17-20, Algorithm 1 .

The above design provides a dual advantage during the initial consensus phase: nodes within the source shard not only engage in intra-shard consensus but also leverage the collective bandwidth of the shard to relay cross-shard transactions proactively. Compared with traditional approaches that delay transaction relay until after the execution phase in the source shard, our approach adopts a pre-dissemination strategy, allowing transactions to be pre-distributed to the relevant shards during prepare operation of PBFT consensus. As a result, the source shard nodes only need to broadcast light-weight confirm proofs that represent these cross-shard transactions has been committed. This innovation significantly reduces transaction latency. The integration of EC within the sharded architecture optimizes the use of intra-shard bandwidth and markedly improves the network's throughput and latency.

Algorithm 1 Replica Algorithm in Source Shard

```
1: **Consensus Round Begin:**
2: if node is leader then
3:     package transactions from the $mempool$ into $block$.
4:     $Pieces$ := SpliteBlock($block$)
5:     for $P_i$ in $Pieces$ do
6:         SendMSG $\langle$pre-prepare, $P_i\rangle$ to $node_i$.
7:     end for
8: end if
9: Upon receiving pre-prepare message $P_i$:
10: $B_{intra}$, $B_{corss}$ := extractData($P_i$)
11: broadcastMSG $\langle$txs-piece, $B_{\text{intra}}\rangle$ within its shard.
12: for $B^k_{cross}$ in $B_{corss}$ do
13:     broadcastMSG $\langle$presend-piece, $B^k_{cross}\rangle$ within $shard_k$.
14: end for
15: wait for enough txs-piece messages to recover $block$.
16: broadcastMSG$\langle$prepare, $v, hash, seq, \sigma_{id}\rangle$within its shard.
17: Upon receiving $\geq 2f+1$ prepare votes:
18: Broadcast $\langle$commit, $v, hash, seq, \sigma_{id}\rangle$.
19: Aggregate the $\sigma_{id}$ and pack into confirm proof $M$.
20: Broadcast $M$ to at least $f+1$ nodes in the target shard.
```

B. Reliable Relay Phase

The Reliable Relay Phase starts concurrently with the prepare operation of the PBFT consensus within the Source Execution Phase, allowing for partial overlap of these two phases. In this phase, if nodes in source shard collects at least $2f+1$ prepare votes, the node aggregates the votes into confirm proof and sends the proof to at least $f+1$ nodes in the target shard [29]. Besides, the nodes in source shard also need to broadcast its commit vote to other nodes within its shard. The confirm proof M consists of the following:

$$M = \{hash, AGG, seq, shardID\} \quad (1)$$

Here, $hash$ means the hash of Proposed block. AGG means the aggregated signature of prepare votes. To ensure the atomicity of transactions, reliable broadcasting during relaying is imperative. Therefore, it is necessary to ensure that at least $f+1$ nodes can obtain a quorum vote from the source shard's nodes.

Next, we will discuss why we set the beginning of relay proof after collecting enough prepare votes but not before the nodes collect at least $2f+1$ prepare votes. This design choice enables nodes in the target shard to acquire and preemptively integrate relevant cross-shard transactions into their ledger during the block proposal stage in the source shard. However, it introduces a security risk. Specifically, a malicious leader could exploit the target shard's lack of knowledge regarding the source shard's account information, producing many transactions that will not be committed on the source shard to the target shard to commit.

The first round of consensus, which is indicated by a node receiving at least $2f+1$ prepare votes, represents that the cross-shard transactions have been verified as correct by the majority of nodes in the source shard. We observe that following the first round, the transactions have a high likelihood of final commitment. The only situation where they might not commit is if the leader fails or if in the network less than $2f+1$ nodes have received the $2f+1$ prepare votes. The section VI will provide an analysis of the probability of failure in the commit phase of the PBFT. Therefore, a cross-shard transaction can only be transmitted to the target shard for preemptive inclusion if it has received correctness confirm proof from source shard. Given that during the Source Execution Phase, we have already relayed EC-encoded chunks to the nodes in the target shard, the relay process now only requires confirm proof of the corresponding hashes in set P_i. By maintaining a distinct separation between the transactions forwarding and validation stages, Presto mitigates the risk of such adversarial exploits, ensuring that relay processes are both secure and reliable against potential attacks.

Algorithm 2 Replica Algorithm in Target Shard

```
1: if node is leader then
2:     keep monitoring headers of other shards
3:     if New headers in other shards exist in $Pending\ Tree$ then
4:         add $TxFinal$ into $block$.
5:     end if
6: end if
7: Upon receiving $CP$ message:
8: store $CP_{\text{k,i}}$.
9: broadcastMSG $\langle$presend-piece, $CP_{\text{k,i}}\rangle$ within its shard.
10: if $CP_{\text{k,i}}$ can recover then
11:     recover Transactions from $CPs$.
12: end if
13: Upon receiving confirm proof $M$:
14: if $M.hash$ related Transactions recieved then
15:     package transactions associated with $M.hash$ into $txes$.
16:     add $txes$ into $Mempool$.
17:     broadcastMSG $\langle$send-M, $M\rangle$ within its shard.
18: else
19:     request $txes$ from peer nodes.
20: end if
21: Upon receiving $\geq 2f+1$ commit votes:
22: for $tx$ in $receivedTxs$ do
23:     if $tx$ to be executed in $Pending\ Tree$ then
24:         execute $tx$ in $Pending\ Tree$.
25:     end if
26:     if $tx$ is finalized transaction then
27:         merge $Pending\ Tree\ node$ into $state\ tree$.
28:     end if
29: end for
```

C. Target Execution Phase

The Target Execution Phase follows the Reliable Relay Phase, with a degree of overlap with the source shard execution phase. In this phase, upon receiving confirm proof M and verifying the signatures of at least $2f+1$ nodes from the source shard, the target shard leader node packs transactions associated with the $hash$ of CP from mempool into block transaction list, as depicted in Algorithm 2. Besides, the target shard nodes also attach the confirm proof M into $Proofs$ of P_i.

As the target shard normal nodes receive P_i containing the confirm proof M that represents CP transaction, they recover the block and then continue to consensus process. After block is committed, they store the execution results of the CP's transactions within the pending tree. The pending tree acts as a staging ground. The pending tree mechanism allows for the utilization of tokens involved in unfinalized cross-shard transactions by other transactions until these tokens finalized in state tree.

The pending tree serves as a data structure alongside the receipt tree, state tree, and transaction tree, introduced to enhance the availability of cross-shard transfers during execution. It holds the provisional final states of transactions likely to be committed to the chain. Structurally, the pending tree is organized as a Merkle tree, where each key corresponds to the hashes and their respective heights derived from checkpoint states in specific shard. This represents that if a hash at a given height within the source shard matches, then the node's state is supposed to be ultimately merged into the state tree in target shard. The pending tree stores values that have multiple attributes including $hash$, $token$ and $formal\ key$, which are illustrated in Figure 2. The $hash$ and $token$ denote the amount that is to be transferred to a specific address. For cross-shard transactions utilizing the token in pending tree from the source shard, $formal\ key$ is required to record the formal key of the pending tree within the source shard. When users initiate cross-shard transactions that utilize the token in pending tree, these transactions are accompanied by the keys of the pending tree in source shard. Consequently, nodes responsible for verifying the validity of these transactions also verify the corresponding hashes and heights. If the hash at the corresponding height of the obtained header does not match, the cross-shard transaction will be rejected. Once an unfinalized cross-shard transaction enters the pending tree, it creates a temporal balance in pending tree. For instance, after a transaction from source shard transfers tokens to address A in the target shard, A can engage in new transactions leveraging the temporal balance of 10 in the pending tree before the cross-shard transaction is finalized. As shown in the Figure 2, when A utilizes temporal balance, A generates a new transaction Tx. This action triggers the creation of a series of child nodes under the node from Address A.

Through this design, system's liveness is significantly improved, enabling temporal balance of unfinalized transactions to be previously utilized. These execution results generated by cross-shard transactions in pending Tree are ultimately settled during the fourth phase, known as the Final Redemption Phase which will be described in subsection V-D. In summary, the addition of the pending tree enhances the overall transactional accessibilty by allowing for the interim use of cross-shard transactions prior to final commitment of cross-shard transactions on the blockchain.

D. Final Redemption Phase

Consensus nodes continuously monitor for hash values corresponding to keys in the pending tree that appear within their locally maintained list of block headers of different shards. When a leader node detects such a seq in the key of the pending tree and the seq of $formal\ key$ exist in other shards, it proposes a special transaction. This special transaction $Txfinal$ includes $nonce$ which is the key of the pending tree. This special transaction structure does not involve state dependency relations since, by design, the targets in transfer transactions inherently execute as receipt actions. Therefore, it is only necessary to execute the add operation to the formal state. During the process of integrating specific nodes from the pending tree into the state tree, if a token originates from the pending tree of a different shard, the system will verify whether the corresponding hashes and $formal\ key$ exist in other shard. If this verification fail, the merge of this particular token will be abandoned.

After receipt execution of the node in the pending tree, other forked nodes of pending Tree that have the same height but not be committed into source shard are purged. Once the seq of the $nonce$ in $Txfinal$ surpasses the height value of an unused pending transaction, consensus nodes also perform a cleanup operation. As depicted in Figure 2, our design enables us to purge failed transactions from the pending tree based on their nonce values.

VI. Analysis

In this section, we first analyze the performance of Presto's cross-shard transaction processing protocol and provide the corresponding theoretical modeling. Subsequently, we examine the protocol's consistency, liveness, and atomicity from a security perspective. Finally, we investigate potential censorship attacks.

A. Performance Analysis

According to the design proposed in Section V, a certain degree of parallelization can intuitively reduce transaction latency. In this section, we will mathematically analyze the extent to which this design achieves optimization in the consensus process. Here, B denotes the size of the propose message. b represents the bandwidth available to the nodes within the shard. Firstly, we analyze the latency improvement when the proposed block is normally committed. The time for the leader to distribute blocks in the traditional way is $\frac{B}{b} \cdot N$.

Let $T_{collect}$ denote the time taken for nodes to broadcast to each other after receiving the proposed block and to collect $2f+1$ votes. Once a node receives at least $2f+1$ votes, it starts a new round of broadcasting and collecting votes until the block is finalized. The time for this process is represented by T_{commit}. In the source shard, we divide the transaction list of size B into m parts. Therefore, the time for the leader in Presto to distribute the transaction list to all nodes and recover the transaction list is the sum of two forwarding time and the decoding time.

The time consumption in the Presto protocol can be expressed as

$$\frac{B \cdot N}{m \cdot b} + B \cdot \left(\frac{N - \frac{N}{m}}{b \cdot m} \right) + T_{\text{collect}} + T_{\text{lite-relay}} + T_{\text{target-consensus-lite}} \quad (2)$$

In contrast, the time consumption in the relay-based method is given by

$$\frac{B \cdot N}{b} + T_{\text{collect}} + T_{\text{commit}} + \frac{(f+1) \cdot B}{b} + T_{\text{target-consensus-full}} \quad (3)$$

Here, $T_{\text{target-consensus-lite}}$ is different from $T_{\text{target-consensus-full}}$ in that the full cross-shard transaction procedure is reduced to confirm proofs denoted by M. Moreover, $T_{\text{lite-relay}}$ represents the time taken to relay content of constant size, which compared to the term $\frac{(f+1)\cdot B}{b}$, highlighting its efficiency.

Probability Calculation for Passing the First Round but Not the Second Round: To calculate the likelihood of a block progressing through the first round (Prepare) but failing to reach commitment, we utilize the cumulative distribution function to measure probabilities under specific circumstances. Any individual node in PBFT systems can be divided into two categories, including consensus reached and not reached node. The Faulty Probability Determined (FPD) model is employed when the probability of each node's failure is constant. The FPD model is particularly suited for assessing the performance of larger systems where node failures are probabilistically estimated. Let us denote the probability of node failure as P_f and the probability of a block passing the first round but not committed as P_p. To elucidate the correlation between the success rate P_p and P_f, it is imperative to establish two pivotal conditions requisite for achieving consensus:

- No more than $\lfloor \frac{N}{3} \rfloor$ faulty nodes in the first round (Event A).
- In the commit round, failures occur in more than $\lfloor \frac{N}{3} \rfloor$ nodes (Event B).

Here, we have $P_p = P(A) \times P(B|A) \times P_f$. This is because if more than one-thirds of the nodes fail to collect enough commit votes, the new leader may not utilize the previous block, and the probability is P_f, resulting in the block not being committed. Assuming the presence of i faulty nodes in the prepare round, where $0 \leq i \leq \lfloor \frac{N}{3} \rfloor$, the cumulative distribution function allows us to deduce:

$$P(A) = \sum_{i=0}^{\lfloor \frac{N}{3} \rfloor} \binom{N}{i} P_f^i (1 - P_f)^{N-i} \quad (4)$$

The probability $P(B|A)$ is contingent upon $P(A)$. Equation (4) postulates the existence of i faulty nodes in the pre-prepare round. Assuming j nodes fail to achieve consensus, where $0 \leq j \leq \lfloor \frac{N}{3} \rfloor$, we obtain:

$$P(B|A) = \sum_{j=\lceil \frac{N}{3} \rceil}^{N} \binom{N}{j} P_f^j (1 - P_f)^{N-j} \quad (5)$$

For example, suppose $N = 50$ and $P_f = 0.1$. In that case, is approximately:

$$P_p \approx 1.75 \times 10^{-6} \quad (6)$$

It is worth noting that the above derivation process is based on certain assumptions and approximations, and the actual situation may differ. However, this approach helps us estimate the performance and security of the PBFT protocol in the presence of malicious nodes.

B. Atomicity of Cross-shard Transactions

Cross-shard transactions need four phases, so the atomicity analysis involves the reliability analysis of these four phases. First, the cross-shard transaction is executed in the source shard. If it fails to be committed, the only reason is that the consensus process proposed by the leader node in source shard is not finally completed. In this case, if the nodes in the target shard do not collect $2f + 1$ votes from the source shard, the transaction will not exist in the block of the target shard. If the target shard collects at least $2f + 1$ votes, the transaction will exist in the pending tree of the target shard. Due to the uniqueness of the height of the source shard blockchain, when another new block with the same height proposed by the source shard is committed, the block with a different hash that forks out in the pending tree will be deleted during the finalization phase of the target shard. Therefore, the execution of this transaction that is not committed in the source shard will not be finalized in the target shard.

Next, we investigate whether the relay stage is reliable if the transaction is successfully committed in the source shard. Since it has been committed, it means that at least $2f + 1$ nodes have broadcast their votes. In the relay stage, the nodes in the source shard need to pass the voting message to no less than $f + 1$ nodes in the target shard, so at least one normal node receives the vote and broadcasts it, ensuring that the cross-chain transaction is committed in target shard.

C. Security Analysis

Theorem 1 (Consistency of Transactions)**.** *In this protocol, there is no round r in which there are two honest party ledger states $\log_1$ and $\log_2$ with transactions tx_1 and tx_2, respectively, such that $tx_1 \neq tx_2$ and $tx_1.I \cap tx_2.I \neq \emptyset$.*

Proof. The ledgers of the source shard and target shard themselves are guaranteed by the PBFT algorithm. Between the two ledgers, suppose tx_1 is in block A and tx_2 is in block B, where block A is generated before block B but at the same height. At this time, block B replaces block A. Initially, the pending tree will record the cross-shard transaction record of block A, which will not be finalized. At this time, block B will exist in the pending tree at the same height until the final finalized state determines the final tx input state. The consistency at this moment is also guaranteed by the consensus algorithm of the target shard. □

Theorem 2 (Liveness of Transactions)**.** *If a cross-shard transaction request is submitted to the system, after R rounds of* *node communication (intra-shard or cross-shard), it is either accepted and committed by the system or comprehensively rejected by the system, where R is the liveness parameter.*

Proof. In Presto, honest nodes acting as proposers can affect the liveness of transactions. Therefore, we discuss two cases based on the honesty of the proposer [30]:

1) The proposer is malicious. In this case, if the proposer proposes an empty block or a block containing false transactions, even if malicious nodes vote, the consensus in intra-shard requires at least $2f+1$ votes to proceed to the next round. The consensus in this round will be discarded. The transaction will be packaged by other nodes. Since the relay mechanism ensures that the confirm proof M will appear in at least one non-Byzantine node, the liveness is guaranteed by the consensus mechanism of the target shard, and the situation and principle are the same as in the source shard. Therefore, a round of consensus involving cross-shard transactions will eventually be terminated.
2) The proposer is honest. If there are non-leader nodes that act maliciously by not voting or participating in forwarding relay transactions, their number will not exceed $N/3$. The redundancy protection of the EC code ensures that even if there are f malicious nodes, it will not affect the liveness of transactions in the consensus stage. During the Reliable Relay Phase, when the nodes in the target shard receive the confirm proofs, even if they are unable to collect the complete transaction due to time reasons, they can query the honest nodes in the source shard to recover the complete cross-shard transaction. An additional round of communication ensures that the transaction can be committed. Therefore, a round of consensus involving cross-shard transactions will be terminated.

□

In summary, the Presto protocol can ensure termination and start the next round within a limited number of rounds.

D. Potential Attack Analysis

1) Double Spending Attacks: Double spending attacks occur when a malicious actor attempts to use the same set of tokens or assets for more than one transaction. In the context of cross-shard transactions, these attacks can occur due to the asynchrony between shards and the lack of instantaneous global visibility. In each shard of Presto, the account nonce and the PBFT consensus in both the source and target shards can prevent double-spending attacks for intra-shard transactions. When it comes to cross-shard transactions, the nonce in pending tree combines height of the blockchain in the source shard block, which must also be verified by the target shard's consensus. The above design can avoid double spending attacks.

2) Censorship Attacks: In a censorship attack [31], a malicious node or group of nodes intentionally ignores or delays processing certain transactions, effectively censoring them from the network. During the initial phase of processing cross-shard transactions, the threat of attacks is not considered pertinent, as this stage is safeguarded by the underlying consensus algorithm. Consequently, concerns regarding censorship attacks are primarily confined to the Reliable Relay Phase, which involves the relay of cross-shard transaction data. The implementation of Erasure Coding can ensure that the recovery of data with $2f+1$ segments is feasible, thereby permitting the system to tolerate attacks from up to f nodes. However, this requires a reliable broadcast protocol to ensure data integrity. In the Reliable Relay Phase, the procedure to relay the confirmation proof can ensure the phase reliablity.Therefore, when nodes in the target shard receive the confirm proof but haven't recovered the transaction from the source shard, they will directly ask other nodes in the source shard to obtain the cross-shard transactions. This mechanism ensures that no single node or small group of nodes can unilaterally block or delay a transaction, thereby preventing censorship.

In conclusion, while cross-shard transactions are inherently susceptible to various adversarial attacks, the Presto protocol effectively mitigates these risks through a combination of unique mechanisms and design choices.

VII. Evaluation

A. Experimental Setup

1) Prototype Implementation: We have implemented the Presto using a widely recognized emulator platform, namely the BlockEmulator [8]. The BlockEmulator is a sharding chain emulator that implements all blockchain processes, including consensus, execution, and broadcasting. The BlockEmulator utilizes PBFT consensus within a single shard.

The experimental environment was meticulously architected using Alibaba Cloud's virtual machines, each equipped with 4 CPU cores (Intel Xeon, 2.5/3.2GHz) and 8GB of RAM. The network bandwidth was standardized at 50Mbps to reflect common operational constraints. The interval for generating blocks is set to 3 seconds, and the block size parameter is configured to be 10,000 txes. Within our experimental framework, each shard was composed of four nodes, with each virtual machine dutifully hosting a single node to ensure isolation and accurate performance measurement.

2) Dataset: To ensure the veracity of our experiments, we employed a dataset of genuine transactions extracted from the Ethereum blockchain. We collected transactions in the most recent one million transactions up to March 2024 [32] for evaluations. Clients were programmed to dispatch transactions at a predetermined rate, emulating the temporal distribution of transactions on the live Ethereum network.

3) Baselines: For comparative purposes, we also implemented two schemes, BrokerChain and Monoxide, within the block emulator. To ensure a fair comparison, we maintained the same PBFT consensus for intra-shard transactions and kept the execution layer unchanged. The primary difference between the two prototypes lies in the processing of cross-shard transactions. The fundamental concepts of these schemes are discussed in Section II. Monoxide's architecture, centered around a relay mechanism for cross-shard transaction forwarding. BrokerChain's unique client-as-relay design also offered an alternative perspective on cross-shard transaction handling. In BrokerChain, each shard was configured with a single broker to handle cross-shard transactions. Pyramid was deliberately excluded from our baseline comparison due to its complexity, which entails a cumbersome three-round consensus process, leading to higher bandwidth consumption and latency in transaction processing compared to both Monoxide and BrokerChain.

To evaluate the individual contributions of Presto's design components, we conducted an ablation study. We also implemented two variants in addition to Presto: Foresee and Presend.

Foresee: This implementation removes the Erasure Coding (EC) based pre-distribution of transactions and relies solely on early relaying during the Source Execution Phase. Upon receiving at least $2f + 1$ prepare votes, the relay message containing all cross-shard transactions is disseminated.

Presend: This variant incorporates EC-based pre-distribution into the relay-based scheme. Before the source shard enters the commit phase, transactions are pre-distributed to target shard using EC. Then, the confiirm proof is sent only after the cross-shard transactions committed on source shard.

4) Metrics: Our experiments delved into two pivotal performance metrics: throughput and latency of cross-shard transaction processing. We measured the throughput to assess the system's capacity to handle a high volume of transactions. This latency is particularly crucial for cross-shard transactions, where delays can be exacerbated by the complexity of cross-shard communication. Our analysis was further enriched by examining the transaction pool within each shard, quantifying the backlog of transactions awaiting packaging. This measure served as an indirect indicator of the system's efficiency under duress, reflecting Presto's superior ability to sustain minimal queuing, even when subjected to the rigors of high transactional loads, thus ensuring a reduction in overall latency.

B. Throughput Measurement

First, We fixed the number of shards at 8 and adjusted the transaction rate to measure the TPS and latency of different systems. Figure 3 shows that Presto notably excels beyond the established baseline. Additional ablation experiments reveal considerable enhancements over Monoxide and BrokerChain. With increased injection speeds, Presto's improvements are distinctly more substantial when compared with Monoxide and BrokerChain. As shown in the Figure 3, the Presto method demonstrated the greatest improvement compared to Monoxide, with a 63% increase in TPS and up to a 66% reduction in latency. Compared to BrokerChain, Presto achieved a 54% reduction in latency and a 48% decrease in TPS. Presto showcased notable performance enhancements, especially in bandwidth-constrained scenarios, due to its innovative approach to reducing bandwidth costs during the relay process.

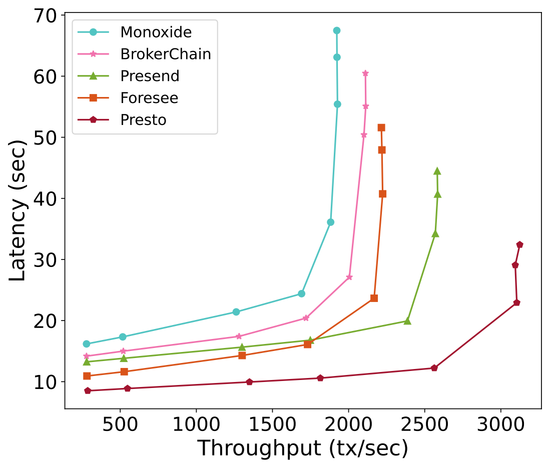

Figure 3. Latency vs. Throughput.

We also analyzed and compared the Presend and Foresee methods. The Presend method showed a 34% improvement in TPS and a 44% improvement in latency. Presend's pre-distribution technique accelerated the relaying of cross-shard transactions, and the use of EC coding reduced the bandwidth pressure on the primary nodes. The Foresee method also demonstrated improvements in latency, with a 34% reduction, leading to a 16% increase in TPS. Foresee's pending tree mechanism achieved concurrency reduction, which decreased latency. Additionally, the Foresee mechanisms, to some extent, reused the idle bandwidth during the consensus process.

Next, we set the injection speed at a constant 3000 transactions per second and increase the number of shards in the systems. Figure 4 illustrates that the system's TPS initially increased proportionally with the number of shards, demonstrating efficient scaling. However, the performance bottleneck disappears when the number of shards increases beyond 12.

C. Latency Analysis

We then shifted our focus to the latency aspect, varying the number of shards to measure the confirmation delay in an expanding scenario. As the shard count rose, the transaction confirmation latency correspondingly decreased, which is depicted in Figure 5. This trend was attributed to the increased number of nodes available to process cross-shard transactions and a reduction in the average number of cross-shard transactions spanning two shards, thus alleviating congestion. Presto exhibited the least latency, as it parallelized the execution across two shards, reducing latency by a substantial 66% compared to Monoxide. Presend and Foresee achieved approximately 46% and 33% lower latency compared to Monoxide.

D. Transaction Lifecycle Analysis

Under constant transaction rate and shard quantity, we repeatedly sampled multiple transaction data to obtain the time consumed in each stage of cross-shard transaction processing. The baseline stages were divided into three: T1, the time phase in Source Execution Phase; T2, the time phase in Reliable

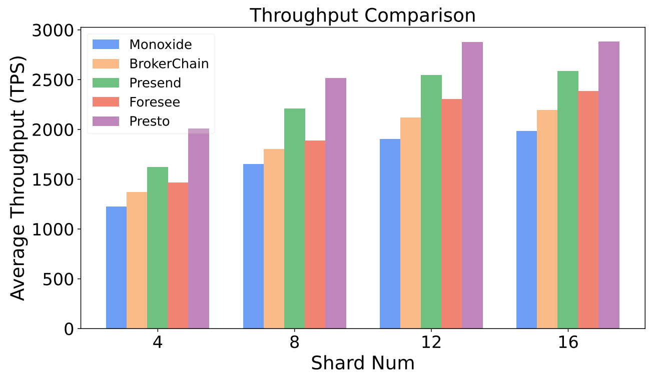

Figure 4. Transaction throughput with varying shard number S.

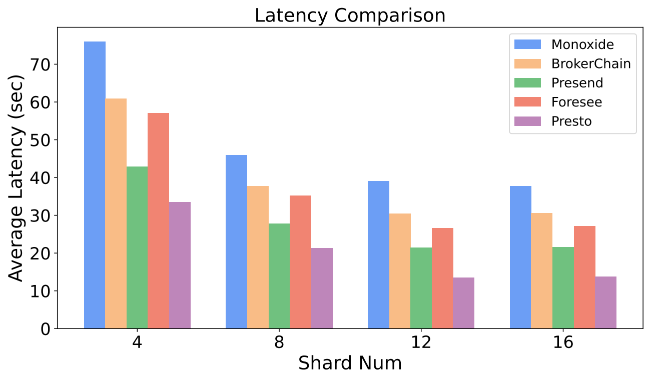

Figure 5. Transaction latency with varying shard number S.

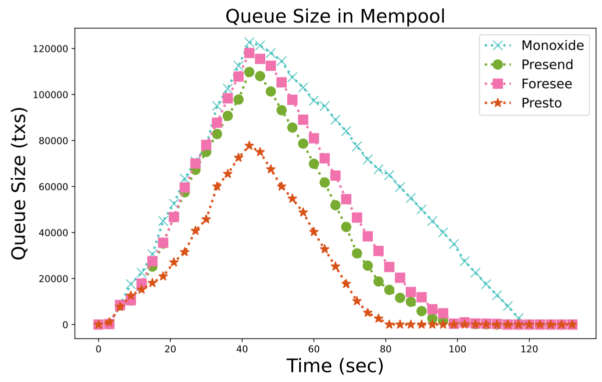

Figure 6. Queue size of the TX pool.

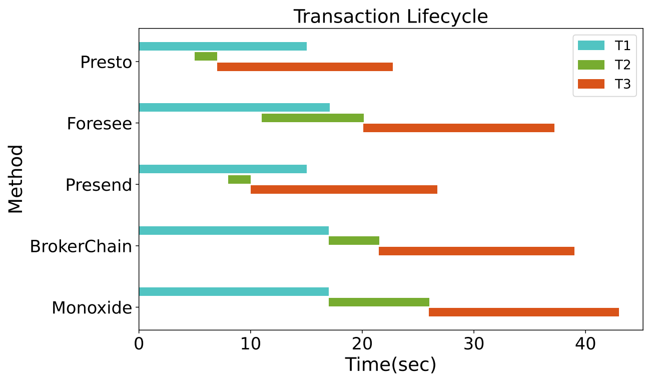

Figure 7. The lifecycle of cross-shard transaction.

Relay Phase; T3, the time phase in Target Execution Phase. The results, as shown in the Figure 7, demonstrate that Presto significantly reduces most in latency of all phases compared to other methods. Presend ranks second in performance, with substantial reductions in T1, T2, and T3 due to early distribution and the use of EC codes. The Foresee mechanism advances the beginning phase of the T2 stage by preemptively packaging transactions in the Target Shard and slightly reduces the duration of T2 by capitalizing on idle bandwidth ahead of time.

E. Queue Size Analysis

Our subsequent investigations delved into the queue length during transaction injection as Figure 6. With four shards and an injection rate of 4000 transactions per second sustained over 45 seconds, we observed the dynamics of the transaction pool. As shown, the number of pending transactions within the Presto transaction pool was significantly lower compared to the other two baselines. This reduction not only reflects Presto's high throughput capabilities but also contributes to the reduction in latency, further enhancing the overall system performance and user experience.

VIII. Conclusion

In this paper, we have presented Presto, an innovative blockchain system that leverages sharding to effectively mitigate the latency encountered in cross-shard transaction processing. At the heart of Presto's design is a dual-faceted strategy: on the one hand, it proactively executes cross-shard transactions in an optimistic manner, thereby reducing the overall processing time; on the other hand, it harnesses pre-dissemination and Erasure Coding to maximize the use of available bandwidth within each shard for the transmission of cross-shard transactions. Notably, the versatility of Presto's approach is such that it can be applied to the majority of sharding blockchain architectures. Comparative analysis with recent protocols for cross-shard transaction processing reveals that Presto enhances system throughput by 63% and decreases transaction confirmation latency by up to 66%. These improvements underscore the potential of Presto to serve as a robust solution for the next generation of sharding blockchain systems, offering a scalable and efficient framework for managing cross-shard transactions.

IX. Acknowledgement

The authors would like to thank the anonymous reviewers for their comments. This work was supported by the National Key R&D Program of China under Grant 2023YFB2703800 and the Beijing Natural Science Foundation under Funding No. IS23055. The contact author is Zhen Xiao.

References

[1] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:52220000

[2] P. Li, M. Song, M. Xing, Z. Xiao, Q. Ding, S. Guan, and J. Long, "Spring: Improving the throughput of sharding blockchain via deep reinforcement learning based state placement," in *Proceedings of the ACM on Web Conference 2024*, ser. WWW '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 2836–2846. [Online]. Available: https://doi.org/10.1145/3589334.3645386

[3] J. Wang and H. Wang, "Monoxide: Scale out blockchains with asynchronous consensus zones," in *Symposium on Networked Systems Design and Implementation*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:73724931

[4] 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," *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, pp. 1968–1977, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:249901334

[5] Z. Hong, S. Guo, P. Li, and W. Chen, "Pyramid: A layered sharding blockchain system," *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pp. 1–10, 2021. [Online]. Available: https://api.semanticscholar.org/CorpusID:236479859

[6] M. Castro, "Practical byzantine fault tolerance," in *USENIX Symposium on Operating Systems Design and Implementation*, 1999. [Online]. Available: https://api.semanticscholar.org/CorpusID:221599614

[7] L. Rizzo, "Effective erasure codes for reliable computer communication protocols," *ACM SIGCOMM computer communication review*, vol. 27, no. 2, pp. 24–36, 1997.

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

[9] H. Team, "Byzantine fault." 2018.

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

[11] G. Wang, Z. J. Shi, M. Nixon, and S. Han, "Sok: Sharding on blockchain," *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:204749727

[12] S. Delgado-Segura, C. Pérez-Solá, G. Navarro-Arribas, and J. Herrera-Joancomartí, "Analysis of the bitcoin utxo set," *IACR Cryptol. ePrint Arch.*, vol. 2017, p. 1095, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:11760706

[13] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016. [Online]. Available: https://api.semanticscholar.org/CorpusID:3351908

[14] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 583–598, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:29885138

[15] L. N. Nguyen, T. D. T. Nguyen, T. N. Dinh, and M. T. Thai, "Optchain: Optimal transactions placement for scalable blockchain sharding," *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pp. 525–535, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:207757825

[16] S. Das, V. Krishnan, and L. Ren, "Efficient cross-shard transaction execution in sharded blockchains," *arXiv preprint arXiv:2007.14521*, 2020.

[17] 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.

[18] 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. [Online]. Available: https://api.semanticscholar.org/CorpusID:267285085

[19] Z. Cai, J. Liang, W. Chen, Z. Hong, H. Dai, J. Zhang, and Z. Zheng, "Benzene: Scaling blockchain with cooperation-based sharding," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, pp. 639–654, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:254670264

[20] C. Liu, H. Guo, M. Xu, S. Wang, D. Yu, J. Yu, and X. Cheng, "Extending on-chain trust to off-chain–trustworthy blockchain data collection using trusted execution environment (tee)," *IEEE Transactions on Computers*, vol. 71, no. 12, pp. 3268–3280, 2022.

[21] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM (JACM)*, vol. 35, no. 2, pp. 288–323, 1988.

[22] 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*, 2024.

[23] M. Yin, D. Malkhi, M. K. Reiter, G. Golan-Gueta, and I. Abraham, "Hotstuff: Bft consensus with linearity and responsiveness," *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019. [Online]. Available: https://api.semanticscholar.org/CorpusID:197644531

[24] R. Guerraoui, P. Kuznetsov, M. Monti, M. Pavlovič, and D.-A. Seredinschi, "The consensus number of a cryptocurrency," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 307–316.

[25] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on {Bitcoin's}{peer-to-peer} network," in *24th USENIX security symposium (USENIX security 15)*, 2015, pp. 129–144.

[26] R. Rana, S. Kannan, D. Tse, and P. Viswanath, "Free2shard: Adaptive-adversary-resistant sharding via dynamic self allocation," *arXiv preprint arXiv:2005.09610*, 2020.

[27] J. Tian, C. Jing, and J. Tian, "Cuckchain: A cuckoo rule based secure, high incentive and low latency blockchain system via sharding," in *2023 IEEE Symposium on Computers and Communications (ISCC)*. IEEE, 2023, pp. 1228–1234.

[28] D. Xiang, R. Zhou, L. Ma, Q. Zeng, X. Fu, Q. Wang, B. Chen, and Y. Jia, "Decoupling consensus and storage in consortium blockchains by erasure codes," *2022 4th International Conference on Data Intelligence and Security (ICDIS)*, pp. 194–199, 2022. [Online]. Available: https://api.semanticscholar.org/CorpusID:254999497

[29] K. Qiao, H. Tang, W. You, and Y. Zhao, "Blockchain privacy protection scheme based on aggregate signature," in *2019 IEEE 4th International Conference on Cloud Computing and Big Data Analysis (ICCCBDA)*. IEEE, 2019, pp. 492–497.

[30] L. Zhang, B. Zhang, and C. Li, "An efficient and reliable byzantine fault tolerant blockchain consensus protocol for single-hop wireless networks," *IEEE Transactions on Wireless Communications*, 2023.

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

[32] P. Zheng, Z. Zheng, J. Wu, and H.-N. Dai, "Xblock-eth: Extracting and exploring blockchain data from ethereum," *IEEE Open Journal of the Computer Society*, vol. 1, pp. 95–106, 2020.