Scaling Blockchain via Layered Sharding

Zicong Hong, *Graduate Student Member, IEEE*, Song Guo, *Fellow, IEEE*, and Peng Li, *Senior Member, IEEE*

***Abstract*— As a promising solution to blockchain scalability, sharding divides blockchain nodes into small groups called shards, splitting the workload. Existing works for sharding, however, are limited by cross-shard transactions, since they need to split each cross-shard transaction into multiple sub-transactions, each of which costs a consensus round to commit. In this paper, we introduce PYRAMID, a novel sharding system based on the idea of layered sharding. In PYRAMID, the nodes with better hardware are allowed to participate in multiple shards and store the blockchains of these shards thus they can validate and execute the cross-shard transactions without splitting. Next, to commit the cross-shard transactions with consistency among the related shards, we design a cooperative cross-shard consensus based on collective signature-based inter-shard collaboration. Furthermore, we present an optimization framework to compute an optimal layered sharding strategy maximizing the transaction throughput with the constraint of system security and node resource. Finally, we implement a prototype for PYRAMID based on Ethereum and the experimental results reveal the efficiency of PYRAMID in terms of performance and scalability, especially in workloads with a high percentage of cross-shard transactions. PYRAMID improves the throughput by up to 3.2 times compared with the state-of-the-art works and achieves about 3821 transaction per seconds for 20 shards.**

***Index Terms*— Blockchain, scalability, sharding, cross-shard transactions.**

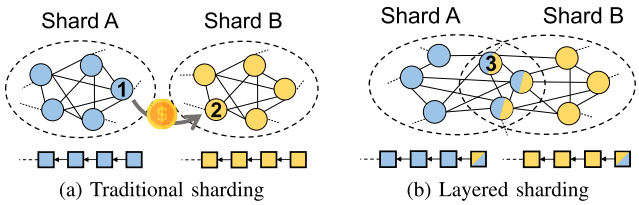
I. INTRODUCTION

BLOCKCHAIN, represented by Bitcoin [1], Ethereum [2] and Hyperledger Fabric [3], has attracted growing attention in the area of supply chains [4], high performance computing [5], and search engine [6] in recent years. It is a distributed ledger technology used to guarantee high security and reliability of historical data in a distributed system involving multiple untrusted participants and without a central authority.

As a prerequisite for broad application of blockchain, *scalability* is an important property [7], [8]. It denotes the ability of a blockchain system to support the increasing load of transactions, as well as the increasing number of nodes in the network. However, most of the existing popular blockchain systems suffer from poor scalability [1], [2] since their consensuses involve all nodes. In other words, every node needs to verify and store all transactions and every consensus message needs to be broadcast in the whole network.

Among the technologies for the blockchain scalability, sharding is one of the most promising and popular ones [9]. Its main idea is to divide nodes into multiple consensus groups called *shards*. Accounts are distributed to the shards, each of which processes the transactions involving their stored accounts. As shown in Fig. 1(a), each shard maintains a blockchain and runs its own consensus independently. Ideally, the throughput scales out linearly with the number of shards. The technology has been paid close attention by the academia for recent years [10], [11], [12], [13], [14], [15], [16], [17], [18]. Moreover, for the industry, many blockchains are currently being upgraded to a sharding architecture. For example, Zilliqa has implemented sharding on its mainnet [19] and Ethereum plans to support sharding in its Eth2 upgrade in 2022 [20].

Although sharding improves the scalability of blockchain, it raises a new challenge to cross-shard transactions. The cross-shard transactions are the transactions involving multiple accounts distributed in different shards. For example, as shown in Fig. 1(a), the transaction sent by Node 1 in Shard A and received by Node 2 in Shard B is a cross-shard one. More seriously, each transaction may involve more accounts in practice (see § VI.) To commit a cross-shard transaction, the existing sharding works [12], [13] divide it into several



(a) Traditional sharding (b) Layered sharding

Fig. 1. Illustration for different blockchain sharding systems.

Manuscript received 10 March 2022; revised 15 June 2022; accepted 30 June 2022. Date of publication 21 October 2022; date of current version 22 November 2022. This work was supported in part by the Key-Area Research and Development Program of Guangdong Province under Grant 2021B0101400003; in part by the Hong Kong Research Grants Council (RGC) Research Impact Fund (RIF) under Project R5060-19; in part by the General Research Fund (GRF) under Project 152221/19E, Project 152203/20E, and Project 152244/21E; in part by the National Natural Science Foundation of China under Grant 61872310; in part by the Shenzhen Science and Technology Innovation Commission under Grant JCYJ20200109142008673; and in part by the Japan Society for the Promotion of Science (JSPS) KAKENHI under Grant 21H03424. An earlier version of this paper was presented at the 2021 IEEE Conference on Computer Communications [DOI: 10.1109/INFOCOM42981.2021.9488747]. *(Corresponding author: Song Guo.)*

Zicong Hong is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China (e-mail: zicong.hong@connect.polyu.hk).

Song Guo is with the Department of Computing, The Hong Kong Polytechnic University, Hong Kong, China, and also with The Hong Kong Polytechnic University Shenzhen Research Institute, Shenzhen 518057, China (e-mail: song.guo@polyu.edu.hk).

Peng Li is with the School of Computer Science and Engineering, The University of Aizu, Aizuwakamatsu 965-8580, Japan (e-mail: pengli@u-aizu.ac.jp).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/JSAC.2022.3213350.

Digital Object Identifier 10.1109/JSAC.2022.3213350

sub-transactions, each of which is handled by the associated shard. It serious degrades the transaction throughput and multiplies the confirmation latency in a sharding system.

A. Basic Idea

This paper proposes a new blockchain sharding schema, named *layered sharding*, to improve the scalability via sharding while processing the cross-shard transactions efficiently. Its main idea is to allow shards to overlap, which means some nodes can locate in more than one shard as shown in Fig. 1(b). These nodes store the blockchains of multiple shards, thus they can verify and execute the cross-shard transactions directly. For example, as shown in Fig. 1(b), Node 3 can verify and execute the cross-shard transaction involving Shard A and B since Node 3 stores the blockchains of both shards. Moreover, the idea is consistent with the fact that the hardware (e.g., storage, computation, network) of blockchain nodes is different in a blockchain system. Thus, the nodes with better hardware can be deployed to more shards, which not only fully utilizes the resource of blockchain systems but also efficiently processes cross-shard transactions.

B. Challenges

However, it is non-trivial to achieve the layered sharding before tackling the following challenges. 1) *How to design a consensus to commit cross-shard transactions in multiple shards.* The traditional sharding works adopt either Nakamoto or Byzantine consensus for block proposal within each shard. However, in layered sharding, nodes can generate blocks composed of cross-shard transactions. These blocks will involve the state of multiple shards, need to be committed in multiple shards, and maybe conflict with other shards' consensus, thus demanding a new design of block structure, consensus procedure, and conflict detection. 2) *How many shards are needed and which nodes should be assigned to which shards.* In the traditional sharding works, all shards have an identical role and the hardware of the nodes is assumed the same. In comparison, the shards in the layered sharding play different roles and the capacity of nodes is different. Some nodes are responsible for internal transactions, while others should handle cross-shard transactions. It is critical to study shard assignment for the layered sharding, which determines system performance and security level.

To this end, this paper presents a layered sharding system for blockchain called PYRAMID. The main **contributions** can be summarized as follows.

- We propose a new blockchain architecture that forms a layered structure among shards. Based on the characteristics of cross-shard transactions, we investigate the verification rules for cross-shard transactions and design a cross-shard structure for blocks.
- We design a cooperative cross-shard consensus to commit cross-shard transactions in multiple shards in one round. The consensus is composed of collective signature-based inter-shard collaboration to commit the cross-shard block.
- We present an optimization framework for layered sharding. We first analyze the transaction structure and node resource based on the observation of blockchain systems. We then formulate a transaction throughput maximization problem with the constraint of security and resource and solve it based on integer programming.
- We implement a prototype for PYRAMID based on Ethereum. The results show that it outperforms the state-of-the-art sharding works in the aspects of throughput and latency. It improves the throughput by up to 3.2 times compared with the existing works and achieves about 3821 transaction per seconds (TPS) for 20 shards.

The rest of this paper is organized as follows. § II provides background. § III introduces the layered sharding model. § IV presents the architecture including the new designs of block and consensus. § V provides a formal security analysis. § VI designs an optimization framework. § VII extends our approach to a general case. § VIII performs experiments and analyzes the evaluation results. § IX introduces some related works for sharding. Finally, § X gives the conclusion.

II. PRELIMINARY

A. Blockchain

A blockchain is a distributed ledger recording historical transactions. The ledger is maintained by a set of untrusted blockchain nodes connected by a peer-to-peer network and each node maintains a full copy of the ledger. The transactions issued by clients are verified by the nodes and then grouped and recorded into the blockchain via the consensus protocol. In the following, we introduce the common transaction models and consensus protocols for the blockchain.

1) Transaction Model: There are two major types of transaction models, i.e., the Unspent Transaction Output (UTXO) model [1] and the account/balance model [2]. In the UTXO model, each block stores many transactions, each of which contains one or more inputs and outputs. Each input of a transaction includes a reference to one output of an existing transaction. Note that the output needs to have not been referenced by any inputs before. In the account/balance model, each block represents a state and stores a list of accounts and transactions. Each account stores a balance, and if it is a smart contract, it will also store the code and internal storage. The transactions record the history of state transitions (e.g., the change of balance or contract storage) in the block.

2) Consensus Protocol: There are two major kinds of consensus protocols, i.e., Byzantine Fault Tolerant (BFT) protocols and Nakamoto consensus protocols. Practical BFT (PBFT) is the most well-known BFT protocol and has been adopted by Hyperledger Fabric [3]. It consists of three successive phases, i.e., pre-prepare, prepare, and commit phase. The transition condition between any two phases is that each node collects a quorum of messages. Proof-of-Work (PoW) is the most well-known instance of Nakamoto consensus and has been used in Bitcoin [1]. Each node must solve a computational puzzle to propose a new block. Although the specific processes of them are different, the aims of them are same in blockchain systems, i.e., ensuring all nodes in the

system agree on some information while facing malicious or faulty nodes.

B. Sharding

Sharding is an idea originating from the database partitioning technique that divides a very large database into much smaller parts named *shards* [21]. In database, by distributing the workload over multiple shards and managing the shards separately, transactions can be processed in parallel. Similarly, in blockchain systems, sharding divides the blockchain nodes and the distributed ledger into shards. Transactions can be distributed and processed in different shards, which enables the computation, communication and storage of nodes scale as the number of shards. The study of the sharding protocol typically focuses on three critical components as follows.

1) Shard Formation: Before joining the system, each node needs to establish an identity via a Sybil attack-resistant method, such as PoW. Then, based on the identity, each node will be assigned to a shard randomly so that each shard is honest with high probability. Besides, to prevent the attack of adversary, the shards need to reconfigure in fixed time periods.

2) Cross-Shard Mechanism: In the sharding system, because the ledger is separated into shards, cross-shard transactions happen frequently. When processing each cross-shard transaction, both of its atomicity (i.e., transactions are committed and aborted atomically) and consistency (i.e., each transaction commit produces a semantically valid state) need to be guaranteed among shards using a cross-shard mechanism.

3) Intra-Shard Consensus: Within each shard, the nodes need to run a Byzantine consensus protocol to agree on a block including a set of transactions proposed in each consensus round. The consensus protocol should achieve *safety* and *liveness*. The former means that honest nodes agree on the same value and the latter means that the valid transactions will eventually be included in the ledger.

The previous blockchain sharding works will be discussed in § IX in detail.

C. Motivation

To illustrate the performance degradation brought by cross-shard transactions to the traditional blockchain sharding, we conduct an initial experiment based on the cross-shard mechanism in Monoxide [13]. It divides each cross-shard transaction related to K shards into K sub-transactions at least. Note that the number of sub-transactions may be more than K, which will be discussed in § VI-A. As shown in Fig. 2, when there are more cross-shard transactions or each cross-shard transaction involves more shards, the transaction throughput of the sharding system decreases.

In practice, according to statistics [12], more than 96% transactions are cross-shard in a sharding system. Moreover, based on the data provided by XBlock [22], we conduct an analysis for Ethereum from July 30th, 2015 to July 6th, 2019 and find more than 15% smart contract related transactions are composed by more than 1 steps and the average number of accounts in a transaction is 3.35. Besides, due to the popularity of more complex smart contracts, the proportion of multi-step transactions is increasing over time.

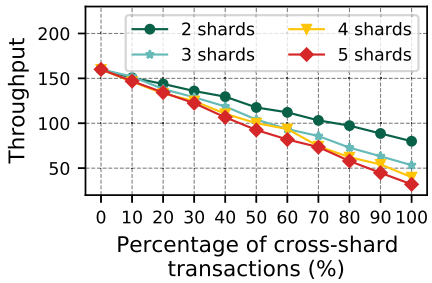


Fig. 2. Performance of sharding with different proportion of cross-shard transactions over a function of number of their related shards.

III. The Pyramid Model

A. Threat & Network Model

Pyramid consists of a set of blockchain nodes following the Byzantine failure model which includes two kinds of nodes, i.e., *honest* and *malicious*. The honest nodes abide by all protocols. The malicious nodes are controlled by a Byzantine adversary and may collide with each other and violate the protocols in arbitrary manners, such as denial of service and tampering, forgery, and interception of messages. Furthermore, similar to other sharding systems [10], [11], [12], we assume that the Byzantine adversary is slowly-adaptive, i.e., the set of malicious nodes and honest nodes are fixed during each epoch and can be changed only between epochs.

The nodes in Pyramid are connected by a partially synchronous peer-to-peer network [23]. In particular, the messages sent by a node can reach any other nodes with optimistic, exponentially-increasing time-outs.

B. Transaction Model

Pyramid adopts the account/balance model for the ledger state in the form of a pair of account and balance. Moreover, Pyramid can be extended to the UTXO model, which is discussed in § VII. We consider a transaction as a payment between two accounts, namely *sender* and *receiver*. A more general case about transactions involving more than two accounts or supporting smart contract is discussed in § IV-G. We leave the transaction model with more semantic information such as blockchain database [24], [25] to future works.

C. Layered Sharding Model

In Pyramid, each blockchain node belongs to a shard. Different from the traditional sharding schemes in which the shards are the same type, the shards in our layered sharding are different types as follows.

1) **i-shard**: Each i-shard stores the state of accounts in the shard and can independently verifies the internal transactions similar to shards in the traditional sharding.
2) **b-shard**: Each b-shard bridges multiple i-shards by storing the state of accounts in the i-shards and dealing with the cross-shard transactions related to the i-shards.

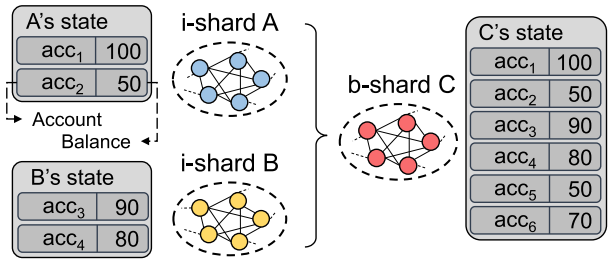


Fig. 3. Illustration for a layered sharding for i-shard A, B and b-shard C.

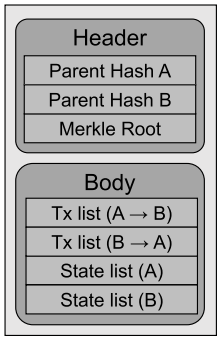


Fig. 4. Illustration for a cross-shard block for i-shard A and B.

For example in Fig. 3, there are three shards, i.e., i-shard A, i-shard B, and b-shard C. The nodes in i-shard A stores the balance of account 1 (denoted by acc_1) and acc_2 while the nodes in i-shard B stores the balance of acc_3 and acc_4. The i-shard A is responsible for the internal transactions involving acc_1 and acc_2 and the i-shard B for its internal transactions. The b-shard C bridging these two i-shards by storing their stored accounts and taking the job for cross-shard transactions among i-shard A and B. Besides, the b-shard C can also store its own accounts, i.e., acc_5 and acc_6.

IV. Architecture

A. Architecture Overview

Similar to most blockchain sharding systems, Pyramid proceeds in fixed time periods named *epochs*, each of which includes two stages, i.e., sharding formation and block consensus, as follows.

In the first stage, based on a pre-determined layered sharding strategy with the guarantee of both security and performance (§ VI), the nodes are assigned to shards to construct a layered sharding system (§ IV-B.)

In the second stage, there are multiple consensus rounds. For each round, similar to the traditional sharding, the i-shards propose and commit blocks composed of internal transactions. The b-shards can propose cross-shard blocks (§ IV-C) composed of cross-shard transactions and commit them via a cooperative cross-shard consensus (§ IV-D). Finally, the state of an i-shard can be updated based on its block and the involved cross-shard blocks.

Moreover, § IV-E solves the conflict among cross-shard blocks proposed by b-shard and the blocks proposed by the i-shards in the same round. § IV-F fills the gap left by the i-shards which do not have the corresponding b-shard.

B. Layered Sharding Formation

1) Strategy Design: At the beginning of Pyramid, the blockchain founders can decide a layered sharding strategy indicating the number of i-shards and b-shards and which i-shards each b-shard bridges. Then, the strategy can be written into the code as one of genesis parameters (such as block size). The strategy can be decided empirically or based on an layered sharding optimization framework as discussed in § VI.

2) Randomness Generation: In each epoch, sim a global randomness will be first generated via a public-verifiable, bias-resistant, unpredictable and available randomness generation method [9], e.g., the verifiable random function [26], verifable delay function [27], and trusted execution environment [15], similar to that of other sharding systems [11], [12]. It can be considered as a separated module in a sharding system and is orthogonal with our work, thus we do not discuss in detail.

3) Participation: To join the epoch, each node is required to solve a PoW puzzle to protect against Sybil attacks. The puzzle is generated based on the node's public key and the epoch randomness. After solving the puzzle successfully, the node needs to append its solution into an identity blockchain to register its identity. The identity blockchain is a PoW-based blockchain used to record identities of nodes, the same as the identity blockchain in [11], [12], and [28].

4) Assignment: Each admitted node is assigned to an i-shard or b-shard randomly based on the identity of the node and the epoch randomness. Note that the results of assignment for all nodes in the epoch are public and they can be computed based on the randomness in the epoch and the identity chain.

C. Cross-Shard Block Design

In Pyramid, each shard has a blockchain at least. The nodes in each i-shard store one blockchain. For each b-shard, besides its own blockchain, the nodes store multiple blockchains, the number of which equals the number of its related i-shards. Since the nodes in a b-shard stores the state of the related i-shards, they can verify the cross-shard transactions, pack them into a new type of block called *cross-shard blocks*, the structure of which is described as follows.

Each cross-shard block is related to multiple i-shards. It is composed of a *header* and a *body*. The header includes the hashes of parent blocks in the related i-shards and the Merkle tree root of the body. The body includes transactions and states involving the related shards. Fig. 4 illustrates a cross-shard block related to i-shard A and B. The body includes the cross-shard transactions from i-shard A to B and vice versa, denoted by Tx list (A → B) and Tx list (B → A), and the states of accounts in i-shard A and B, denoted by State list (A) and State list (B). Besides, although a cross-shard block includes the state of multiple shard, to save space, after a cross-shard block is committed, each i-shard can only store part of the block. For example, the nodes in i-shard A can only store the Merkle root of State list (B) rather than the raw data.

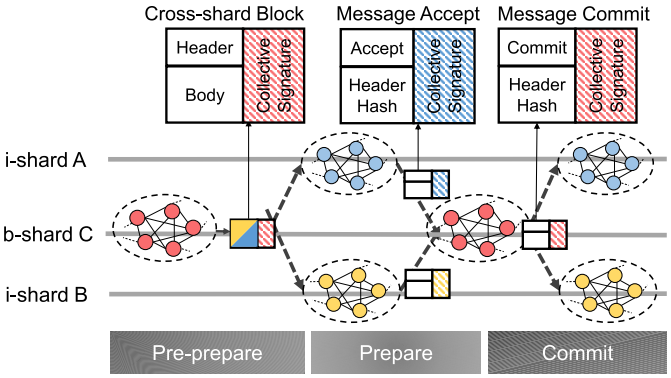


Fig. 5. Illustration for cooperative cross-shard consensus.

D. Cooperative Cross-Shard Consensus

For each consensus round, a leader is first randomly elected from each shard based on the randomness of the epoch. The leader of an i-shard can pack the internal transactions and propose a new block called *internal block*. The internal block can then be committed via a BFT protocol (such as PBFT) similar to the intra-shard consensus in the traditional sharding. In comparison, the leader of a b-shard can pack cross-shard transactions and propose a new cross-shard block that is associated with the state of multiple i-shards. If the block is directly committed via a BFT protocol by the nodes in the b-shard and transferred to the associated i-shards, the block may be conflict with the other blocks committed in associated shards in the same consensus round. Therefore, we design a new consensus to commit the cross-shard blocks with conflict detection named cooperative cross-shard consensus as follows.

Fig. 5 illustrates an example of committing a cross-shard block proposed by the leader in the b-shard involving two i-shards. The normal procedure includes three phases:

1) Block Pre-Prepare: In this round, the leader of the b-shard first picks, validates and executes the cross-shard transactions. Then, it can propose a new cross-shard block related to i-shards A and B as shown in Fig. 4. To protect against invalid cross-shard blocks proposed by a malicious leader, the nodes in the b-shard validate the block and sign if it is valid. A cross-shard block with the signatures of two-third super-majority of nodes attests that the b-shard agrees on it under a Byzantine environment. The signatures can be generated by collective signing protocol in which a decentralized group of nodes can co-sign a multisignature, such as CoSi [29], a scalable protocol that can efficiently scale to thousands of nodes, and Boneh-Lynn-Shacham (BLS) [30] collective proof.

To notify the associated shards for conflict detection, the cross-shard block with the collective proof of the b-shard will be then sent to the associated i-shards, i.e., i-shard A and B.

2) Block Prepare: After receiving the cross-shard block with the collective proof from the b-shard, the nodes in each i-shard can verify the collective signature of the b-shard based on public keys recorded in the identity blockchain. Then, each related i-shard can collective sign the block to denote receiving the block. In particular, the i-shard sends a message of *Accept*, including the hash of header and a collective signature back to the b-shard. Besides, the i-shard can send a message of *Reject* when there is conflict among blocks, which will be described in § IV-E.

3) Block Commit: After the pre-prepare phase, the nodes in the b-shard initialize a counter with the number of its related i-shards. When a node in the b-shard receives a valid message of *Accept* from an associated i-shard, it will decrease its local counter and broadcast the message to other nodes in the b-shard. When the counter equals to 0, the nodes in the b-shard can ensure that the cross-shard block can be committed in this round. As in PBFT, block prepare phase is insufficient to ensure that the block will be committed [28], thus an additional collective signing is needed to guarantee that the cross-shard block will be committed and a message of *Commit* including the hash of header and a collective signature will be sent to the related i-shards.

By default, in the consensus, the inter-shard messages (e.g., cross-shard blocks and messages of *Accept*, *Reject*, and *Commit*) are forwarded to their destination shards by the leaders. To improve the success rate and reduce the safe threshold for network interceptions (which will be discussed in § V), the honest nodes and some trusted infrastructure can also help to relay the inter-shard messages.

E. Conflicting Detection

In § IV-D, the cross-shard block occupies the consensus round of the b-shard and its related i-shards. However, the related i-shards may be in the consensus for their own blocks. Thus, the conflicts are needed to be detected and resolved.

We first define the block conflicts. An intuitive idea is to define that the blocks involving the same accounts are conflicting. However, such a coarse-grained definition can result in frequent abort due to high conflict ratio. Thus, motivated by the idea of commutativity in [17], we propose a fine-grained definition below.

Definition 1: If two blocks commute, i.e., both of them are valid in any order and the final state of shards does not depend on their order, they are not conflict and can be processed in the same round.

For example, as shown in Fig. 6(a), although the cross-shard block X and the internal block Y involve the same accounts, i.e., acc_2, they are not conflicting since either of their relative orderings will increase the balance of acc_2 by 30. In comparison, in the case of Fig. 6(b), the two blocks are conflicting since the balance of acc_2 is 50 and only one block is valid.

Next, in the prepare phase, based on the transaction list in the received blocks, each i-shard can identify the conflicting blocks. Based on the randomness of the epoch, each i-shard accepts one from all conflicting blocks randomly and rejects the other blocks. Thus, among all conflicting blocks, only one block has message *Accept* and the other blocks only have message *Reject*, which prevents the conflicting blocks from being committed and commits the non-conflicting blocks at the same round.

F. Relay Mechanism

In § IV-D, each b-shard can propose cross-shard blocks related to at most its related i-shards. It raises the problem

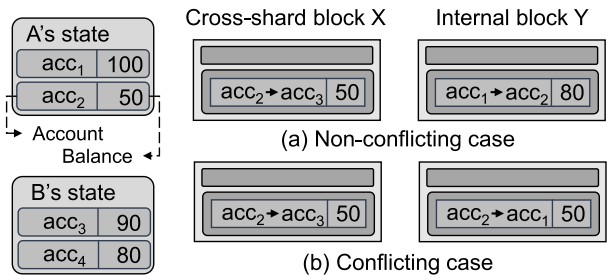


Fig. 6. Illustration for the non-conflicting blocks and the conflicting blocks.

that in order to process all cross-shard transactions, every possible b-shard should exist. However, it is impossible to realize because the number of shards in the blockchain system is limited. Thus, we are going to combine another cross-shard mechanism named *relay mechanism* originating in [13] with PYRAMID. Its main idea is to divide a cross-shard transaction into several sub transactions. Each sub transaction involves one i-shard or multiple i-shards which have the corresponding b-shard, which can solve the problem.

We take some minor modifications compatible with the consensus in § IV-D as follows.

First, each node in a shard is a light node for the other shards. In other words, after a block is committed successfully, both its header and the collective signature will be broadcast to the system and stored in all nodes.

Second, the body includes an additional list named *outbound transaction list*. The list consists of transactions whose senders belong to the related i-shard of the body but receivers do not belong to any related i-shards of the block. For example, the outbound transaction list of the block of Fig. 4 can include transaction whose senders are in i-shard A and receivers are in any i-shard except i-shard A and B.

For the transactions in the outbound transaction list, they are partially validated. In other words, the state of the sender is validated, i.e., the sender has sufficient money, while the state update of the receiver, i.e., the receiver receives proper money, is left to be validated. Although they are not completely committed in this round, the leader or any other nodes can send them and their corresponding Merkle tree path to the next step i-shards or b-shards. Then, in the following consensus rounds, other leaders in i-shards or b-shards can use the Merkle tree path and the block header as a proof to continue the verification for these transactions.

G. General Case

The above design only discuss the case of payments between two accounts, i.e., transactions with single step. However, the multi-step transactions, i.e., transactions involving many interactions among accounts, are common in current systems, which will be discussed in detail in § VI-A. In particular, a multi-step transaction is a sequence of interactions among accounts. Each interaction involves two accounts and can be the money transfer, creation and function-call of smart contracts, etc [2]. The multi-step transaction can be a cross-shard transaction involving several i-shards and be committed by the b-shard bridging these related i-shards using the above cooperative cross-shard consensus and relay mechanism.

Next, we discuss the transaction processing in a more general layered sharding system with five shards, i.e., i-shard A, i-shard B, i-shard C, b-shard D bridging A and B, and b-shard E bridging A, B and C. If there is a transaction involving three accounts in A, B, and C in sequence, respectively. The transaction can be processed in three ways as follows. First, it can be processed by A, B, and C in sequence via the relay mechanism. Second, it can be processed by D and C in sequence via the combination of cross-shard consensus and relay mechanism. Third, it can be processed by E via the cooperative cross-shard consensus. The first one needs three consensus rounds at least, the second one needs two and the third one needs only one.

V. SECURITY ANALYSIS

Similar to other blockchain sharding works [7], [11], [12], [15], the security of our layered protocol includes safety and liveness which are defined and proved as follows.

Definition 2: The safety denotes the honest nodes agree on the same valid block in each round and the liveness denotes the finality for every block, i.e., the block in each round will eventually be committed or aborted.

Theorem 1: The cooperative cross-shard consensus achieves safety if there are no more than $v < \frac{1}{3}$ fraction of malicious nodes in each shard.

Proof: Given no more than $v < \frac{1}{3}$ malicious nodes in each shard, the intra-shard consensus can guarantee the cross-shard block proposed by the b-shard is valid. Then, a message along with a collective signature is honest because honest nodes are the super-majority, i.e., more than two-thirds, of the shard. Meanwhile, the message cannot be modified and forged because the collective signature can be used to detect forgery and tampering. Therefore, the communication among shards can safely proceed if there are no more than $v < \frac{1}{3}$ fraction of malicious nodes in the involved shards, which can guarantee that all related shards can receive the valid cross-shard block. The prepare phase and commit phase in the consensus similar to the two-phase commit protocol in other distributed systems [14], [15]. The prepare phase aims to reach the tentative agreement of commitment for cross-shard transactions and the commit phase aims to perform the actual commit of the transactions among the related shards. Thus, honest nodes in all related shards including i-shards and b-shards agree on the same valid cross-shard block in each round, i.e., the consensus achieves safety. □

Theorem 2: The cooperative cross-shard consensus achieves liveness if there are no more than $v < \frac{1}{3}$ fraction of malicious nodes in each shard.

Proof: According to the system model in § III-A, because the nodes are connected by a partially synchronous network and each shard has no more than $v < \frac{1}{3}$ malicious nodes, the BFT protocol adopted as the intra-shard consensus of each shard can achieve liveness. According to Theorem 1, each shard agrees on the same block in each round. Therefore, no malicious nodes can block the consensus indefinitely and

TABLE I
NOTATION DEFINITIONS

Symbol	Definition
N	Number of blockchain nodes
S	Number of shards
n	Number of nodes in each shard
α_k	Percentage of transactions involving k steps
α	Transaction distribution
β_s	Percentage of cross-shard transactions with s frames
β	Frame distribution
h_i	Hardware capacity of node i
N_i	Maximum number of nodes for b-shards bridging i i-shards
$\mathcal{N}$	Node distribution
K	Block size
T	Block interval
χ	Average size of transactions
λ	Security parameter
d_i	Number of i-shards (or b-shards) bridging i shards
$\mathbf{d}$	Layer distribution

each block will be eventually be committed or aborted, i.e., the protocol achieves liveness. □

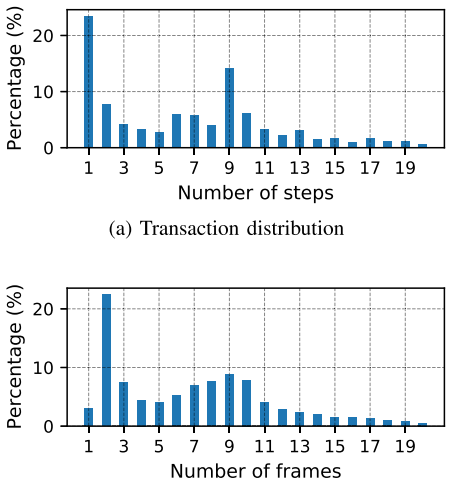
Discussion of Other Attacks: Eclipse attack is an attack to blockchain network and prevents a victim's node from communicating with other honest participants of the network. The attack is difficult to launch, but definitely not impossible. It will break our threat model given in § III-A. Specifically, the cooperative cross-shard consensus can be interrupted when the malicious nodes intercept the inter-shard messages. In the following, we discuss two possible outcomes of the malicious interception. First, if cross-shard blocks in the pre-prepare phase or messages of *Accept* and *Reject* in the prepare phase are intercepted, the consensus in this round will not get to the commit phase; thus, only the round is wasted, and the safety and liveness will not be compromised. Second, if messages of *Commit* in the commit phase are intercepted, the blocks will not be committed in the shards that do not receive the messages. Although these shards can roll back their state and recommit the blocks after receiving the messages for safety, the system's throughput is affected. To raise the bar for eclipse attackers, we can adopt some countermeasures proposed in [31] for the network of PYRAMID. For example, each node in a shard connects a random set of nodes in the other shards. Moreover, we can deploy redundant infrastructure or trusted third parties to help inter-shard communication.

VI. OPTIMIZATION FRAMEWORK FOR LAYERED SHARDING

In this section, we theoretically analyze the performance of layered sharding and design an optimization framework for layered sharding strategy according to the characteristics of blockchain sharding (e.g., transaction demand, node resource and security). The major notations and variables are summarized in Table I.

A. Transaction Model

We define *transaction distribution* for a blockchain system as the distribution of transactions with different steps



(a) Transaction distribution

(b) Frame distribution for 8 shards

Fig. 7. Illustration for transaction distribution and frame distribution in Ethereum from Aug. to Sep. 2021.

$\alpha = \{\alpha_1, \alpha_2, \cdots, \alpha_A\}$ in which a step is the interaction between two accounts (e.g., transfer, function call, and contract creation), α_k is the percentage of transactions involving k steps, and A is the largest possible number of steps involving by a transaction. The transaction distribution α can be collected from a real blockchain system such as Ethereum. For example, Fig. 7(a) illustrates the transaction distribution in Ethereum from Aug. to Sep. 2021 (about 10 millions transactions for smart contracts) based on a dataset provided by XBlock [32].

For each transaction, we define a *frame* as a sequence of steps involving accounts in the same shard. In particular, a frame can be committed by an i-shard or b-shard in a consensus round. Next, given the shard number S, the transaction distribution α can be converted into another *frame distribution* $\beta = \{\beta_1, \beta_2, \cdots, \beta_B\}$ in which β_1 is the percentage of internal transactions, β_s is the percentage of cross-shard transactions with s frames, and B is the largest possible number of frames involving by a transaction. The distribution β can be calculated based on the following theorem. The basic idea of the theorem is that according to the definition of frames, each step that involves two accounts from the same shard can be deleted when computing the number of frames for a transaction.

Theorem 3: For a sharding system with S shards, according to the transaction distribution α, the distribution of transactions with s frames can be calculated by

$$\beta_s = \sum_{k=1}^{A} \alpha_k \binom{k}{k+1-s} \frac{(S-1)^{s-1}}{S^k}. \tag{1}$$

Proof: In a sharding system with S shards, for a transaction involving k steps, where $1 \le k \le A$, it includes $k+1$ accounts (including duplicate ones) that are distributed to S shards uniformly and randomly. To calculate the probability that the number of frames for the transaction is exactly s, we need two following steps. First, we need to pick $k+1-s$

account from all accounts except the first one. For each of these accounts, its related shard is the same as the related shard of its previous account, thus the step including it and its previous account can be deleted. Second, to satisfy that there are s frames, the first frame can belong to S possible shards. Because two consecutive frames should belong to different shards, the remaining frames can belong to $S-1$ possible shards. Thus, the probability that a transaction involving k steps has s frames can be computed as

$$Pr(X=s) = \binom{k}{k+1-s}\frac{S(S-1)^{s-1}}{S^{k+1}} = \binom{k}{k+1-s}\frac{(S-1)^{s-1}}{S^{k}}. \quad (2)$$

Based on Eq. (2) and the distribution of transactions involving different number of accounts α, we can get Eq. (1). □

The most intuitive result shown by the theorem is that when there are more transactions with many steps, the proportion of transactions with many frames increases.

As shown in Fig. 7(b), we illustrate the frame distribution for a sharding system with 8 shards which is derived from the transaction distribution in Fig. 7(c). We emphasize that the above calculation is an approximated method because the duplicate accounts in a transaction are not considered. Besides, the frame distribution β can also be calculated via a simple sampling simulation from the transaction distribution.

B. Node Model

In PYRAMID, there are N nodes, each of which has different hardware capacity of computation, communication and storage. It determines the maximum layer a node can be located in. It is because, as discussed in § IV, the nodes in a b-shard are required to store and verify the state and transactions of the related i-shards. Thus, to prevent the throughput of b-shards from significantly degrading, a b-shard bridging more i-shards requires a higher hardware capacity of nodes.

To determine the maximum number of nodes in different b-shards, we define the hardware capacity of node i as h_i which indicates the bottleneck among the computation, communication and storage of node i. To compute the hardware capacity, we first define a hardware requirement for the nodes in i-shard as H_1, such as 10 GB for storage, 1 MBit/s bandwidth, and 2.40 GHZ CPU. Most of blockchain systems have such a hardware requirement.[1] Then, the hardware requirement for b-shards bridging two i-shards will be defined as a higher value H_2, and so forth for the remainder of the b-shards. The requirement for b-shards bridging most i-shards is H_L. The hardware capacity of a node can be set as the highest hardware requirement of shards it meets. Finally, we can get the maximum nodes able to be in b-shards bridging i i-shards as N_i. Besides, we assume that all nodes are able to be located in i-shards thus $N_1 = N$. We define a *node distribution* as $\mathcal{N} = \{N_1, N_2, \cdots, N_L\}$

[1]https://ethereum.org/en/developers/docs/nodes-and-clients/#requirements

C. Consensus Model

We model the consensus in the layered sharding based on [33] which models several BFT consensus protocols including PBFT, Zyzzyva, and Quorum in a non-sharding blockchain. In the layered sharding, the transaction throughput of each shard depends on two key parameters, i.e., *block size* K and *block interval* T, as follows.

The first one is the number of bytes can be contained in a block, which determines the number of transactions in a block. Moreover, we define the average size of transactions as χ. Note that transactions with different number of steps have the same size. It is because except the first step, the other steps is internal, which means they result from the execution and are not stored in the blockchain [34].

The second one is the average time required for a shard to commit a new block, which is mainly composed of the consensus latency. As discussed in § IV-D, the consensus of layered sharding is composed of three phases of collective signing. Each collective signing includes two round-trips, i.e., one for data distribution and the other for signature aggregation, over the communication tree between the leader and the other nodes, such as [29]. Thus, the consensus latency depends on the practical network environment, e.g., the number of nodes in a shard, the propagation method in the network, and the rate of generating and verifying signatures for each node.

D. Security Model

According to § V, a layered sharding system is secure when there are no more than $v < \frac{1}{3}$ fraction of malicious nodes in every shard. Thus, we can model the probability for forming an unsafe layered sharding system as follows.

We first define X as a random variable serving as the number of malicious nodes assigned to a shard. $n = N/S$ indicates the number of nodes in each shard. Once the number of malicious nodes in a shard exceed $n/3$, the shard can be deem to be unsafe. Finally, based on cumulative binomial distribution function [12], the probability for forming an unsafe shard can be approximated as

$$P[X \geq \lceil n/3 \rceil] = \sum_{x=\lceil n/3 \rceil}^{n} \binom{n}{x} f^x (1-f)^{n-x}. \quad (3)$$

Next, to bound the failure probability of the whole system, we calculate the union bound over S shards. Besides, we adopt a security parameter λ to limit the probability. Therefore, a system can be regarded as safe enough if

$$SP[X \geq \lceil n/3 \rceil] < 2^{-\lambda}. \quad (4)$$

A higher λ can guarantee a safer layered sharding system. For instance, let $\lambda = 4$, then the system is secure if the probability is less than 2^{-4}. Besides, the cumulative hypergeometric distribution-based method for calculating the failure probability [12] is also applicable.

E. Problem Formulation

A sharding strategy in PYRAMID can be defined as a layer distribution $\mathbf{d} = \{d_1, d_2, \cdots, d_L\}$, where d_1 denotes the

number of i-shards, other d_i denotes the number of b-shards bridging i shards and $\sum_{1\le i\le L} d_i = S$. According to § IV-D, a b-shard in d_i can process cross-shard transactions involving i shards at most. Furthermore, for a cross-shard transaction involving more than i shards, the b-shard can process its i frames at most using the relay mechanism. Therefore, the maximum transaction throughput of a layered sharding system can be computed as

$$TPS^{layer} = \frac{\lfloor K/\chi \rfloor}{T}(d_1\underbrace{(\beta_1 + \frac{\beta_2}{2} + \cdots + \frac{\beta_B}{B})}_{(1)} + \sum_{2\le i\le L} d_i\underbrace{(\sum_{s=1}^{B}\beta_s(\frac{i}{S})^{s-1}}_{(2)} + \underbrace{\sum_{s=2}^{B}\beta_s\sum_{j=1}^{s-1}\frac{j}{s}(\frac{i}{S})^{j-1}\frac{S-i}{S})}_{(3)}), \tag{5}$$

in which (1) considers the transactions to be processed by the i-shards and (2) and (3) consider the transactions to be processed by the b-shards. $\frac{i}{S}$ denotes the probability that the related shard of a frame in the transaction is included in the b-shard in d_i. (2) is for the transactions that can be committed in one round and (3) is for the transactions whose j frames can be committed in one round.

In addition, for traditional sharding, each cross-shard transaction related to k shards needs to be divided into k sub-transactions at least. Thus, the maximum transaction throughput of a traditional sharding system with S shards is $TPS^{tradition} = \frac{S\lfloor K/\chi \rfloor}{T}(\beta_1 + \frac{\beta_2}{2} + \cdots + \frac{\beta_B}{B})$. Suppose there are more cross-shard transactions involving more shards in the system. In that case, the transaction throughput of traditional sharding systems shows more serious deterioration than that of the layered sharding, which means the layered sharding achieves better scalability.

Given the frame distribution β, the node distribution $\mathcal{N}$, and the security parameter λ, the selection of optimal sharding strategy $\mathbf{d}$ can be formulated as follows

$$\max_{\mathbf{d}} \quad TPS^{layer} \tag{6}$$

$$s.t. \quad \sum_{l\le i\le L} d_i \le \frac{N_l}{n}, \forall 1 \le l \le L \tag{7}$$

$$SP[X \ge \lceil n/3 \rceil] < 2^{-\lambda} \tag{8}$$

$$d_i \in \mathbb{N}, \forall d_i \in \mathbf{d}. \tag{9}$$

The constraint (7) indicates that the sharding strategy cannot exceed the hardware capacity of nodes in the system. The constraint (8) indicates the strategy should guarantee that the system is secure. The main difficult in solving the problem is that it is integer programming problem and the constraints (7) and (8) are not linear (recall that $n = N/S$ and $\sum_{1\le i\le L} d_i = S$.)

To solve the problem, we first analyze the constraint (8) as follows. Observe that in most sharding systems with thousands of nodes [11], [12], [35], the shard number S is no more than 64 in practice, because the security parameter λ is often roughly set as a big number for high security. Considering that the shard number is a small number, we can enumerate it. Once the shard number S is given, the problem reduces to a linear integer programming as follows

$$\max_{\mathbf{d}} \quad TPS^{layer} \tag{10}$$

$$s.t. \quad \sum_{l\le i\le L} d_i \le \frac{N_l}{N}S, \forall 1 \le l \le L \tag{11}$$

$$\sum_{1\le i\le L} d_i = S \tag{12}$$

$$d_i \in \mathbb{N}, \forall d_i \in \mathbf{d}, \tag{13}$$

which can be efficiently solved by well-developed branch-and cut algorithms [36] or dynamic programming [37].

VII. Extension to UTXO Model

In this section, we extend Pyramid to a UTXO model which is widely adopted by blockchains for cryptocurrencies such as Bitcoin [1] and Litecoin [38].

In a sharding system for the account/balance model, each shard stores a proportion of accounts and their balances (see § III-C.) In comparison, in a sharding system for the UTXO model [11], [12], [15], each shard stores a set of transactions, especially the unspent transactions. For a new transaction, if its inputs are distributed in multiple shards, it is a cross-shard one. We refer to these shards as its *input shards*.

For the cross-shard block structure in the UTXO model, because the transactions are not in the form of sender and receiver, the body of a cross-shard block includes the cross-shard transactions whose input shards are its related shards. For example, the block shown in Fig. 4 includes the cross-shard transactions whose input shards are i-shard A or B.

The cross-shard cooperative consensus runs as follows. In a system shown in Fig. 3, assume that there are two unspent transactions stored in i-shard A and B, respectively. For a cross-shard transaction including these two transactions as inputs, b-shard C validates the transaction, packs it into a cross-shard block, and commits it with the cooperation of i-shard A and B, which is the same as the procedure in § IV-D. Moreover, to avoid double-spending, each output of the transaction is stored in either i-shard A or B. Thus, in the block, each transaction needs to assign an indicator for each output to denote the responsible i-shard.

The relay mechanism in § IV-F is designed for the account/balance model. To transplant it to the UTXO model, we can include the transactions, whose input shards do not all belong to the b-shard, in the outbound transaction list. Besides, we can also use transfer mechanism originating in [12] which is designed for the UTXO model. To apply it in the layered sharding, we need some minor and compatible modification as follows. First, similar to the relay mechanism, each node should be a light node for all shards. Second, for each cross-shard transaction, one of its input shards is chosen as *main input shard* and the others are *sub input shards*. The main input shard can be the shard storing the most number of inputs for

the transaction or a random shard. Third, each sub input shard transfers its stored inputs to the main shard by committing an internal transaction to represent the ownership transfer and then notifies the main input shard by sending the inputs and the corresponding Merkle tree path to the nodes in the main input shard. Finally, after the input transfer is completed, the main shard can commit the transaction using the consensus in § IV-D since it stores all inputs for the transaction.

VIII. Evaluation

A. Implementation

We implement a prototype of Pyramid in Go [39] based on Ethereum [40]. For the intra-shard consensus, we adopt a collective signature-based BFT in Harmony [41]. The communication among nodes or shards is based on libp2p [42]. For the scheduler in the transaction pool, each shard first processes the pending transactions with the oldest creation time and most related shards. We adopt LINGO 19.0 for the linear integer programming in the optimization framework for layered sharding. Besides, we also implement two prototypes of traditional sharding. For a fair comparison, they also adopts the BFT consensus adopted by Pyramid as their underlying consensus. The difference between these two prototypes is the cross-shard transaction processing. The first one uses the relay mechanism in Monoxide [13] and the second one uses the transfer mechanism in RapidChain [12]. Their main ideas are referred to in § IX-C.

B. Experimental Setup

Similar to most running blockchain testbeds, the bandwidth of all connections between nodes are set to 20 Mbps and the links are with a latency of 100 ms in our testbed. The testbed is composed of 16 Amazon EC2 machines, each of which is a c4.2xlarge instance with 8 vCPUs and 15GB RAM. We generate a transaction set with a step of 3. Besides, based on the data provided by XBlock [22], we generate two datasets with the average step of 7.48 and 2.93 to simulate Ethereum and Bitcoin, respectively. The security parameter λ in § VI-D is set as 17, which means the failure probability needs to smaller than $2^{-17} \approx 7.6 \cdot 10^{-6}$, i.e., one failure in about 359 years for one-day epochs. According to § VI-B, we adopt a log-normal distribution with $\sigma = 0.5$ and $\mu = 0.7, 1, 1.3$ to simulate three different node distributions $\mathcal{N}^{low}$, $\mathcal{N}^{medium}$ and $\mathcal{N}^{high}$, respectively. The superscript denotes the level of average hardware capacity in the distribution. For example, $\mathcal{N}^{high}$ has the most nodes able to be located in b-shards among the three distributions.

C. Transaction Throughput & Latency

Fig. 8 shows the transaction throughput in TPS for the traditional sharding and layered sharding with different shard number and node distribution. We can see that the layered sharding improves the transaction throughput by $1.5 \sim 3.2$X against the two traditional sharding prototypes and the improvement is more significant for the higher level of node distribution since there are more b-shards. Furthermore, the average value

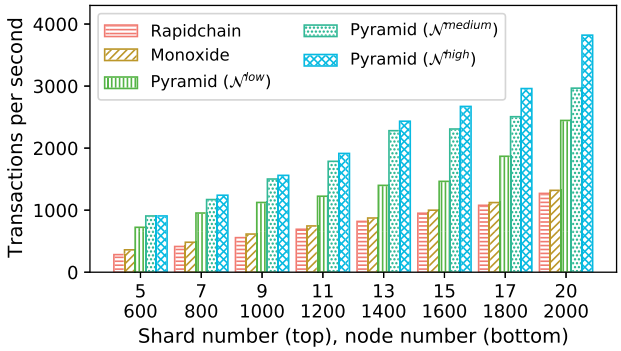


Fig. 8. Transaction throughput for layered sharding with different shard number and node distribution.

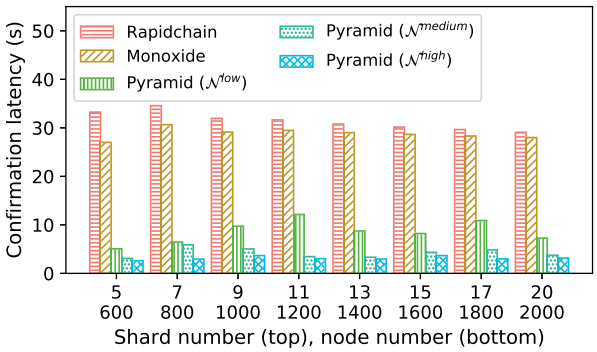


Fig. 9. Latency for layered sharding with different shard number and node distributions.

of $\Delta TPS/\Delta S$ in the layered sharding is about 0.99, which means the TPS scales out as the number of shards including i-shards and b-shards increase. In conclusion, Pyramid exhibits linear scalability, which is better than the traditional sharding, and achieves up to 3821 TPS when there are 20 shards. Besides, the throughput of Rapidchain is slightly lower than that of Monoxide. It is because in our implementation, Rapidchain needs a sub transaction for each account and Monoxide needs a sub transaction for each frame.

Fig. 9 shows the confirmation latency for the traditional sharding and layered sharding. The confirmation latency is another performance metric that denotes the delay between the time that a transaction is issued by a user and the time that the transaction is committed to the blockchain. For a fair comparison, we adjust the transaction demand in different numbers of shards to make their latency close. From the figure, we can see that Pyramid yields a reduction in latency by $59\% \sim 92\%$ in comparison with the traditional sharding systems. This is because the cooperative cross-shard consensus in the layered sharding can commit a cross-shard transaction in less consensus rounds.

D. Storage Overhead

Fig. 10 shows the storage size per node for the traditional sharding and the layered sharding with different node distribution when there are 17 shards. The results include the maximum, average and minimum storage size for the nodes after processing 10 millions transactions. In the figure, the

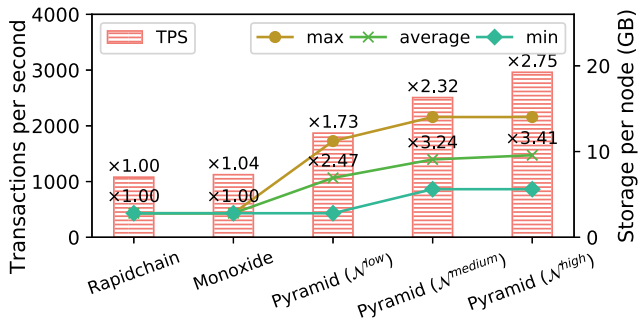


Fig. 10. Trade-off of transaction throughput and storage overhead for layered sharding with different node distribution.

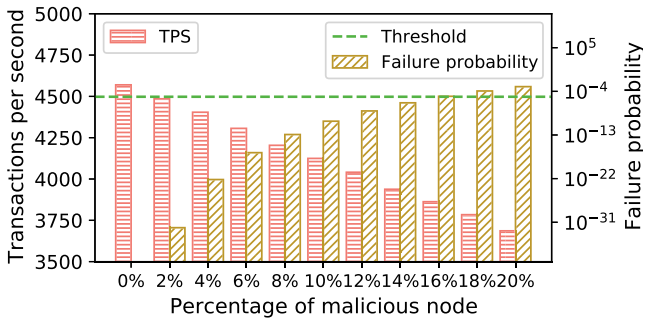


Fig. 11. Transaction throughput and failure probability for different malicious node fraction in a layered sharding system with 17 shards.

number above the bars and the number above the green line denote the increasing times of throughput and average storage size compared with Rapidchain, respectively. From the figure, we can observe that although the layered sharding improves the transaction throughput, it requires more storage for the system. For example, for the medium level of node distribution $\mathcal{N}^{medium}$, it improves the transaction throughput to 2.32 times at the cost of 3.24 times of average storage overhead. Next, the maximum storage size indicates the storage size of nodes in the b-shards bridging the most number of i-shards. The minimum storage size indicates the storage size of nodes in the i-shards or the b-shards bridging the least number of i-shards. From the figure, we can see that a higher level of node distribution has a higher throughput, but all the maximum, average and minimum storage size are increased. However, the storage sacrifice is well worth the performance improvement due to the following reasons. First, the storage is not the main bottleneck in most sharding blockchain systems since there are many state-compaction mechanism such as checkpoint mechanism [11], [43]. Second, as discussed in § VI-B, blockchain nodes can be heterogeneous in practice, which means they have different storage space. The layered sharding aims to fully utilize the storage of the blockchain nodes instead of demanding redundant storage.

E. Security

Fig. 11 shows the performance of the layered sharding with different percentage of malicious nodes. To satisfy the security requirement mentioned in § VIII-B, the failure probability computed by Eq. (4) should be smaller than the threshold 2^{-17}, i.e., the green dotted line illustrated in the figure.

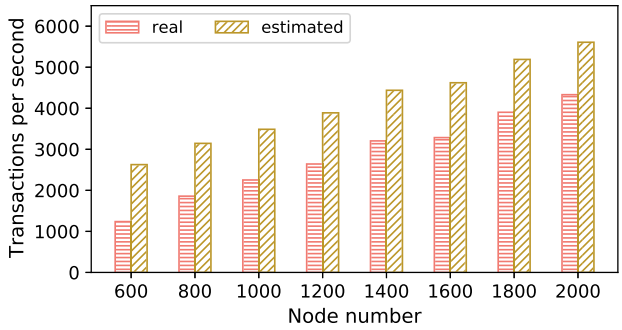


Fig. 12. Real and estimated transaction throughput for optimal sharding strategy with different node number in the same node distribution $\mathcal{N}^{high}$.

We can observe that the layered sharding system is secure when the percentage of malicious nodes is less than 16%. Moreover, when there are more malicious nodes in the system, the performance is worse. This is because within the secure threshold, although a malicious node cannot tamper with the data, it may waste a consensus round when it is a leader. In other words, our consensus in the layered sharding can guarantee that the blocks proposed by malicious nodes will be detected and aborted.

F. Sharding Strategy

Fig. 12 shows the real and estimated throughput of the global optimal sharding strategy. Given the transaction distribution, node distribution and security parameter, LINGO can solve the linear integer programming problem and obtain the global optimal solution in the optimization framework. From the figure, we can see that the real throughput and the estimated one exhibit roughly identical patterns when the number of nodes increase. However, the real throughput is lower than the estimated one, the reason of which is twofold. First, due to the constraints of security and resource, some i-shards do not have the corresponding b-shards. However, Eq. (5) computes the throughput by the probability approach and does not consider the uneven distribution of the b-shard. Second, the shards in the low layers can become the bottleneck of the throughput in practice. In other words, for a cross-shard transaction, some of its frames are quickly committed by the b-shards in the high layers, but the remaining frames need to queue in the i-shards or b-shards in the low layers.

Moreover, we randomly generate 40 sharding strategies under the same node distribution $\mathcal{N}^{medium}$ and security guarantee and their transaction throughput is illustrated in Fig. 13. From the figure, we can see that the transaction throughput of the optimal strategy is 27%, 118%, and 376% higher than the maximum, average, and minimum value of these random sharding strategies, respectively.

G. Workload

We further evaluate the performance of the layered sharding for several workloads with different proportion of cross-shard transactions and different number of transaction steps and the results are illustrated in Fig. 14 and Fig. 15.

As shown in Fig. 14, when the transactions have more steps, the throughput of all sharding systems is reduced. This is

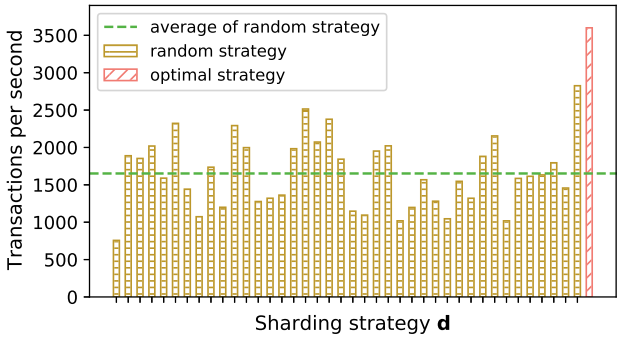


Fig. 13. Transaction throughput for different sharding strategy in the same node distribution $\mathcal{N}^{medium}$.

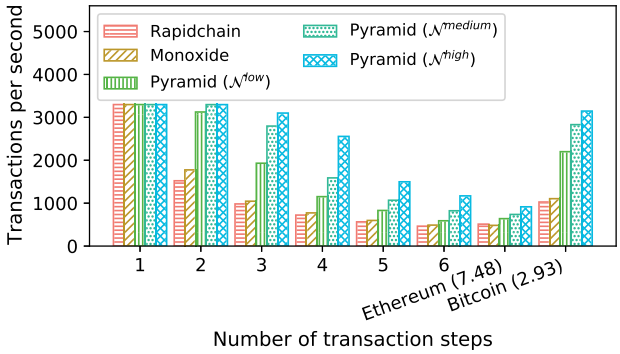


Fig. 14. Transaction throughput for different number of transaction steps in a layered sharding system with 17 shards.

because a transaction with more steps may have more frames after the accounts are sharded according to § VI-A. Since the b-shards in the layered sharding can commit more frames in one consensus round, it has a better performance than the traditional sharding. For the workload of Ethereum and Bitcoin, PYRAMID improves the throughput by up to 90% and 184% compared with the traditional sharding, respectively.

As shown in Fig. 15, when there are no cross-shard transactions, both the traditional sharding and PYRAMID can process a similar TPS. This is because, in this case, all transactions are internal transactions that can be committed in one consensus round in any sharding scheme. Moreover, increasing the percentage of cross-shard transactions significantly reduces the throughput of traditional sharding but only slightly decreases or even increases that of PYRAMID. This is because the strengths of b-shards can fully work when meeting cross-shard transactions. When there are more cross-shard transactions, the throughput improvement brought by the b-shards in PYRAMID compensates for the overhead of cross-shard transactions.

IX. RELATED WORK

A. Centralized Sharding Blockchain

RSCoin [44] is the first sharding blockchain system to support a scalable cryptocurrency whose monetary supply can be controlled by a central bank and whose transactions are validated by the mintettes. Each mintette is a member authorized by the central bank, thus RSCoin is a centralized system and does not work under the Byzantine environment like a public blockchain.

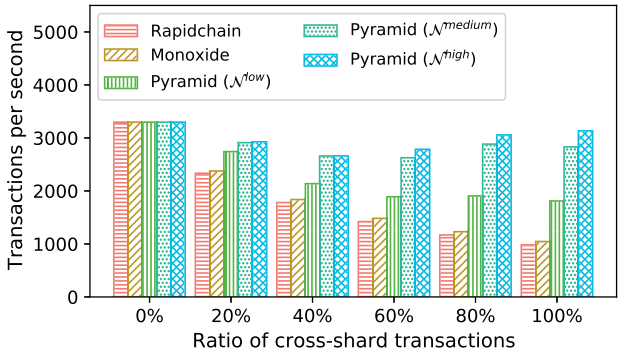


Fig. 15. Transaction throughput for different ratio of cross-shard transactions in a layered sharding system with 17 shards.

B. Partial Sharding Blockchain

ELASTICO [10] is the first decentralized sharding blockchain system. Each node needs to solve a PoW puzzle to join the system. Then, the nodes are distributed to different shards based on the least-significant bits of the solution. Every shard is responsible to validate a set of transactions and achieves consensus based on PBFT. Then, a final shard verifies all the transactions received from shards into a global block which will be then broadcast to and stored in all nodes in the system. Although ELASTICO achieves decentralization and sharding for verification, it does not achieve sharding for storage and bandwidth. It is thus called *partial sharding*. Although there do not exist cross-shard transactions in the system since each node stores complete information of the system, nodes suffer from heavy storage and bandwidth overhead.

Besides, CoSplit [17] is a static program analysis for blockchain sharding and is built on top of a similar partial sharding blockchain named Zilliqa [19]. It is used to infer ownership constraints and commutativity for smart contracts and then concurrently execute transactions in different shards without conflict to maximize parallelism among shards.

C. Complete Sharding Blockchain

To further alleviate the overhead of nodes in the blockchain, a number of researches focus on *complete sharding*, i.e., sharding for transaction verification, storage and communication. The complete sharding, however, brings the challenge of cross-shard transactions and requires cross-shard mechanisms.

Omniledger [11] is the first complete sharding blockchain system. It adopts a client-driven mechanism for cross-shard transactions. To commit a UTXO-based transaction, a client first asks proofs from all input shards and then sends these proofs to all output shards. If any shards reject to provide proof for a transaction, the commitment of transactions will be failed and other shards will roll back the transaction. To support sharding for generic smart contracts, Chainspace [14] is then presented. For privacy, clients need to form a checker program for each of their smart contract. All transactions are executed by the client and the blockchain nodes are only responsible to verify the result provided by the client based on the checkers. To commit a UTXO-based transaction, the client sends the transaction to all input shards and the inputs shards collaborate to run a two-phase commit protocol. However, the above

two client-driven mechanisms put extra burden on typically lightweight user nodes and are vulnerable to denial-of-service (DoS) attacks by malicious users.

RapidChain [12] adopts a shard-driven mechanism for cross-shard transactions. For each cross-shard transaction, the input shards first transfer all involved UTXOs to the output shard by sub-transactions. Then, the cross-shard transaction can be transformed into single-shard transaction and processed in the output shard. Monoxide [13] proposes a relay mechanism for account-based transactions. Each cross-shard transaction will be divided into a sequence of sub-transactions. Each sub-transaction includes the operations involved accounts in one shard. An additional relay transaction is used as a inter-shard message when processing from a sub-transaction to another sub-transaction, i.e., from a shard to another shard.

The cross-shard mechanisms in the above complete sharding systems can guarantee the atomicity and consistency of cross-shard transactions, but the cost to commit them is multiplied. It is because their basic idea is to divide each cross-shard transaction needs into several sub-transactions. Then, all related sub-transactions need to be validated and executed during the consensus. This seriously degrades the sharding performance in terms of throughput and confirmation latency.

Recently, there are some works for the challenge of cross-shard transactions. For example, OptChain [45] proposes a transaction placement method for sharding systems. Its main idea is to place both related and soon-related transactions into the same shards, which can reduce the number of cross-shard transactions as well as temporally balances the workload between shards. Although the method is based on the complete sharding, it can be migrated to the layered sharding by considering that the partition of transactions can overlap, which can be considered in our future work. Tao et al. present a two-layer sharding system [46]. In layer 1, each shard only processes internal transactions. In layer 2, there is a unique shard named MaxShard that stores the complete information, i.e., all blockchains, of the system. Thus, all cross-shard transactions can be validated and processed in the MaxShard directly. However, the two-layer sharding can put a huge burden on nodes in layer 2. The idea of our layered sharding shares a similar idea with Tao et al. [46] but has a more hierarchical sharding structure to distribute the burden among layers.

D. New Achievements

Our previous work [35] proposes a layered blockchain sharding, based on which, this paper has designed several following technologies to optimize this sharding scheme. First, in our previous work, the setting of the layered sharding strategy is heuristic and random, which cannot fully make use of the potential of layered sharding. To solve the problem, we theoretically analyze the factors influencing the performance of layered sharding and present an optimization framework for an optimal layered sharding strategy with the constraint of system security and node resource. The experimental results show that the optimal strategy computed by our framework can increase the throughput by 118% on average. Moreover, our previous work only studies the account/balance model for the transactions. Considering that most of cryptocurrencies such as Bitcoin and Litecoin adopt a UTXO model, we extend our solution to a UTXO model.

X. Conclusion

This paper presented PYRAMID, a layered sharding blockchain system that achieves both linear scalability and efficient cross-shard transactions processing. PYRAMID allows shards to overlap for a layered structure. The shards in the high layer (i.e., b-shards) can validate and process the cross-shard transactions involving the shards in the low layer (i.e., i-shards). We proposed a cooperative cross-shard consensus to enable b-shards to commit the cross-shard transactions without conflict and with the guarantee of security. We developed an optimization framework for layered sharding by modelling the characteristics of transactions, node, security, and consensus in PYRAMID. Based on our experiments, PYRAMID improves the transaction throughput by $1.5 \sim 3.2$X against the traditional sharding works and achieves about 3821 TPS when there are 20 shards. In particular, in the workload with a higher percentage of cross-shard transactions and more transaction steps, PYRAMID has a better performance improvement.

Acknowledgment

The authors thank the anonymous reviewers of IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, which helped improve the paper.

References

[1] S. Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Accessed: Jan. 25, 2022. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[2] G. Wood. *Ethereum: A Secure Decentralised Generalised Transaction Ledger*. Accessed: Jan. 25, 2022. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[3] E. Androulaki et al., "Hyperledger Fabric: A distributed operating system for permissioned blockchains," in *Proc. 13th EuroSys Conf.* New York, NY, USA: Association for Computing Machinery, 2018, pp. 1–15.

[4] K. Francisco and D. Swanson, "The supply chain has no clothes: Technology adoption of blockchain for supply chain transparency," *Logistics*, vol. 2, no. 1, p. 2, Jan. 2018.

[5] A. A. Mamun, F. Yan, and D. Zhao, "BAASH: Lightweight, efficient, and reliable blockchain-as-a-service for HPC systems," in *Proc. Int. Conf. for High Perform. Comput., Netw., Storage Anal.* New York, NY, USA: Association for Computing Machinery, Nov. 2021, pp. 1–18.

[6] M. Li et al., "Bringing decentralized search to decentralized services," in *Proc. 15th USENIX Symp. Operating Syst. Design Implement. (OSDI)*, Jul. 2021, pp. 331–347.

[7] M. J. Amiri, D. Agrawal, and A. El Abbadi, *SharPer: Sharding Permissioned Blockchains Over Network Clusters*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 76–88.

[8] Z. Hong, S. Guo, R. Zhang, P. Li, Y. Zhan, and W. Chen, "Cycle: Sustainable off-chain payment channel network with asynchronous rebalancing," in *Proc. 52nd Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2022, pp. 1–13.

[9] G. Wang, Z. J. Shi, M. Nixon, and S. Han, "SoK: Sharding on blockchain," in *Proc. 1st ACM Conf. Adv. Financial Technol.*, Oct. 2019, pp. 41–61.

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

[11] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "OmniLedger: A secure, scale-out, decentralized ledger via sharding," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2018, pp. 583–598.
[12] M. Zamani, M. Movahedi, and M. Raykova, "RapidChain: Scaling blockchain via full sharding," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2018, pp. 931–948.
[13] J. Wang and H. Wang, "Monoxide: Scale out blockchains with asynchronous consensus zones," in *Proc. 16th USENIX Symp. Networked Syst. Design Implement. (NSDI)*, 2019, pp. 95–112.
[14] M. Al-Bassam, A. Sonnino, S. Bano, D. Hrycyszyn, and G. Danezis, "Chainspace: A sharded smart contracts platform," 2017, *arXiv:1708.03778*.
[15] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi, "Towards scaling blockchain systems via sharding," in *Proc. Int. Conf. Manag. Data*, Jun. 2019, pp. 123–140.
[16] J. Zhang, Z. Hong, X. Qiu, Y. Zhan, S. Guo, and W. Chen, "SkyChain: A deep reinforcement learning-empowered dynamic blockchain sharding system," in *Proc. 49th Int. Conf. Parallel Process. (ICPP)*. New York, NY, USA: Association for Computing Machinery, Aug. 2020, pp. 1–11.
[17] G. Pîrlea, A. Kumar, and I. Sergey, *Practical Smart Contract Sharding With Ownership and Commutativity Analysis*. New York, NY, USA: Association for Computing Machinery, 2021, pp. 1327–1341.
[18] J. Hellings and M. Sadoghi, "ByShard: Sharding in a byzantine environment," *Proc. VLDB Endowment*, vol. 14, no. 11, pp. 2230–2243, Jul. 2021.
[19] Zilliqa Team. *Zilliqa*. Accessed: Jan. 25, 2022. [Online]. Available: https://www.zilliqa.com/
[20] Ethereum. *Shard Chains*. Accessed: Jan. 25, 2022. [Online]. Available: https://ethereum.org/en/eth2/shard-chains/
[21] K. Chodorow, *Scaling MongoDB: Sharding, Cluster Setup, and Administration*. Sebastopol, CA, USA: O'Reilly Media, 2011.
[22] P. Zheng, Z. Zheng, J. Wu, and H.-N. Dai, "XBlock-ETH: Extracting and exploring blockchain data from Ethereum," *IEEE Open J. Comput. Soc.*, vol. 1, pp. 95–106, 2020, doi: 10.1109/OJCS.2020.2990458.
[23] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proc. 3rd Symp. Operating Syst. Design Implement. (OSDI)*, 1999, pp. 173–186.
[24] Y. Zhu, Z. Zhang, C. Jin, A. Zhou, and Y. Yan, "SEBDB: Semantics empowered blockchain database," in *Proc. IEEE 35th Int. Conf. Data Eng. (ICDE)*, Apr. 2019, pp. 1820–1831.
[25] J. Liang, Z. Qin, S. Xiao, L. Ou, and X. Lin, "Efficient and secure decision tree classification for cloud-assisted online diagnosis services," *IEEE Trans. Dependable Secure Comput.*, vol. 18, no. 4, pp. 1632–1644, Jul. 2021.
[26] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *Proc. 40th Annu. Symp. Found. Comput. Sci.*, 1999, pp. 120–130.
[27] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *Proc. Annu. Int. Cryptol. Conf.* Cham, Switzerland: Springer, 2018, pp. 757–788.
[28] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *Proc. 25th USENIX Secur. Symp.*, Austin, TX, USA, Aug. 2016, pp. 279–296.
[29] E. Syta et al., "Keeping authorities 'honest or bust' with decentralized witness cosigning," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2016, pp. 526–545.
[30] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," in *Advances in Cryptology—ASIACRYPT*, C. Boyd, Ed. Berlin, Germany: Springer, 2001, pp. 514–532.
[31] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg, "Eclipse attacks on bitcoin's peer-to-peer network," in *Proc. 24th USENIX Secur. Symp.*, Washington, DC, USA, Aug. 2015, pp. 129–144.
[32] P. Zheng, Z. Zheng, J. Wu, and H.-N. Dai, "XBlock-ETH: Extracting and exploring blockchain data from ethereum," *IEEE Open J. Comput. Soc.*, vol. 1, pp. 95–106, 2020.
[33] M. Liu, F. R. Yu, Y. Teng, V. C. M. Leung, and M. Song, "Performance optimization for blockchain-enabled industrial Internet of Things (IIoT) systems: A deep reinforcement learning approach," *IEEE Trans. Ind. Informat.*, vol. 15, no. 6, pp. 3559–3570, Jun. 2019.
[34] T. Chen et al., "Understanding ethereum via graph analysis," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Apr. 2018, pp. 1–9.
[35] Z. Hong, S. Guo, P. Li, and W. Chen, "Pyramid: A layered sharding blockchain system," in *Proc. IEEE INFOCOM Conf. Comput. Commun.*, May 2021, pp. 1–10.
[36] R. Hemmecke, M. Köppe, J. Lee, and R. Weismantel, *Nonlinear Integer Programming*. Berlin, Germany: Springer, 2010, pp. 561–618.
[37] D. P. Bertsekas et al., *Dynamic Programming and Optimal Control*. Belmont, MA, USA: Athena Scientific, 2011.
[38] Project Development Team. *Litecoin—Open Source P2P Digital Currency*. Accessed: Jan. 25, 2022. [Online]. Available: https://litecoin.org/
[39] Google. *The Go Programming Language*. Accessed: Jan. 25, 2022. [Online]. Available: https://golang.org/
[40] Ethereum. *Go Ethereum*. Accessed: Jan. 25, 2022. [Online]. Available: https://github.com/ethereum/go-ethereum
[41] Harmony. *Harmony*. Accessed: Jan. 25, 2022. [Online]. Available: https://github.com/harmony-one/harmony
[42] Libp2p. *The Go Implementation of the Libp2p Networking Stack*. Accessed: Jan. 25, 2022. [Online]. Available: https://github.com/libp2p/go-libp2p
[43] G. Avarikioti, E. Kokoris-Kogias, and R. Wattenhofer, "Divide and scale: Formalization of distributed ledger sharding protocols," 2019, *arXiv:1910.10434*.
[44] G. Danezis and S. Meiklejohn, "Centrally banked cryptocurrencies," 2015, *arXiv:1505.06895*.
[45] L. N. Nguyen, T. D. T. Nguyen, T. N. Dinh, and M. T. Thai, "OptChain: Optimal transactions placement for scalable blockchain sharding," in *Proc. IEEE 39th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2019, pp. 525–535.
[46] Y. Tao, B. Li, J. Jiang, H. C. Ng, C. Wang, and B. Li, "On sharding open blockchains with smart contracts," in *Proc. IEEE 36th Int. Conf. Data Eng. (ICDE)*, Apr. 2020, pp. 1357–1368.

Zicong Hong (Graduate Student Member, IEEE) received the B.Eng. degree in software engineering from the School of Data and Computer Science, Sun Yat-sen University. He is currently pursuing the Ph.D. degree with the Department of Computing, The Hong Kong Polytechnic University. His current research interests include blockchain, edge/cloud computing, and federated learning.

Song Guo (Fellow, IEEE) is currently a Full Professor at the Department of Computing, The Hong Kong Polytechnic University. His research interests include edge AI, machine learning, mobile computing, and distributed systems. He has published many papers in top venues with wide impact in these areas and was recognized as a Highly Cited Researcher (Clarivate Web of Science). He holds a Changjiang Chair Professorship awarded by the Ministry of Education of China. He is a fellow of the Canadian Academy of Engineering and a fellow of the IEEE Computer Society. He was an IEEE ComSoc Distinguished Lecturer and a member of IEEE ComSoc Board of Governors. He was a recipient of over a dozen Best Paper Awards from IEEE/ACM conferences, journals, and technical committees. He is the Chair of IEEE Communications Society (ComSoc) Space and Satellite Communications Technical Committee. He has also served as the chair for organizing and technical committees of many international conferences. He is the Editor-in-Chief of IEEE OPEN JOURNAL OF THE COMPUTER SOCIETY. He has served for IEEE Computer Society on Fellow Evaluation Committee and has been named on editorial board of a number of prestigious international journals like IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS, IEEE TRANSACTIONS ON CLOUD COMPUTING, and IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING.

Peng Li (Senior Member, IEEE) received the B.S. degree from the Huazhong University of Science and Technology, China, in 2007, and the M.S. and Ph.D. degrees from The University of Aizu, Japan, in 2009 and 2012, respectively. He is currently a Senior Associate Professor at The University of Aizu. He has authored or coauthored over 100 papers in major conferences and journals. His research interests include wired/wireless networking, cloud/edge computing, the Internet-of-Things, and distributed AI systems. He is an Editor of IEEE OPEN JOURNAL OF THE COMPUTER SOCIETY and *IEICE Transactions on Communications*.