SCALING BLOCKCHAIN VIA SHARDING

ZICONG HONG

PhD

The Hong Kong Polytechnic University

2025

The Hong Kong Polytechnic University
Department of Computing

Scaling Blockchain via Sharding

Zicong Hong

A thesis submitted in partial fulfillment of the requirements for
the degree of Doctor of Philosophy
June 2024

CERTIFICATE OF ORIGINALITY

I hereby declare that this thesis is my own work and that, to the best of my knowledge and belief, it reproduces no material previously published or written, nor material that has been accepted for the award of any other degree or diploma, except where due acknowledgment has been made in the text.

Signature: _____

Name of Student: Zicong Hong

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 three challenges. First, cross-shard transactions multiply the overhead of blockchain sharding, since the system needs to split each cross-shard transaction into multiple sub-transactions, each of which costs a consensus round to commit. Second, the independent and random scheduling for cross-shard transactions in different shards results in numerous conflicts and aborts since their involved accounts may be modified or locked by the others before they are committed. Third, for a blockchain database, a new and popular blockchain application, it is challenging to construct a scalable blockchain database through traditional on-chain sharding. Therefore, in chapter 3, we present a new blockchain sharding schema, named *layered sharding*, to improve the scalability via sharding while processing the cross-shard transactions efficiently. In chapter 4, we present PROPHET, a sharding blockchain system with deterministic ordering for conflict-free transactions. In chapter 5, we propose GRIDB, the first scalable blockchain database that distributes tables to different shards and provides efficient cross-shard database services. Evaluation of the real-world datasets for blockchain shows the remarkable performance improvement of our proposed systems over existing solutions.

Publications arising from the thesis

- [**EUROSYS**] **Zicong Hong***, Jian Lin*, Song Guo, Sifu Luo, Wuhui Chen, Roger Wattenhofer, and Yue Yu (*The first two authors have equal contribution). "*Optimus: Warming Serverless ML Inference via Inter-Function Model Transformation*". European Conference on Computer Systems, 2024.

- [**INFOCOM**] Jinyu Chen, Wenchao Xu, **Zicong Hong***, Song Guo, Haozhao Wang, Jie Zhang and Deze Zeng (*mentoring author). "*OTAS: An Elastic Transformer Serving System via Token Adaptation*". IEEE International Conference on Computer Communications, 2024.

- [**WWW**] Enyuan Zhou, Song Guo, Zhixiu Ma, **Zicong Hong***, Tao Guo and Peiran Dong (*mentoring author). "*Poisoning Attack on Federated Knowledge Graph Embedding*". The Web Conference, 2024.

- [**VLDB**] Enyuan Zhou, Song Guo, **Zicong Hong**, Christian S Jensen, Yang Xiao, Dalin Zhang, Jinwen Liang, and Qingqi Pei. "*VeriDKG: A Verifiable SPARQL Query Engine for Decentralized Knowledge Graphs*". International Conference on Very Large Data Bases, 2024.

- [**NDSS**] Jianting Zhang, Wuhui Chen, Sifu Luo, Tiantian Gong, **Zicong Hong**, and Aniket Kate. "*Front-running Attack in Distributed Sharded Ledgers and Fair Cross-shard Consensus*". The Network and Distributed System Security Symposium, 2024.

- [**ICDE**] Wuhui Chen, Ding Xia, Zhongteng Cai, Hong-Ning Dai, Jianting Zhang, **Zicong Hong**, Junyuan Liang, and Zibin Zheng. "*Porygon: Scaling Blockchain via 3D Parallelism*". The IEEE International Conference on Data Engineering, 2024.

- [**VLDB**] **Zicong Hong**, Song Guo, Enyuan Zhou, Wuhui Chen, Huawei Huang, and Albert Zomaya. "*GriDB: Scaling Blockchain Database via Sharding and Off-Chain Cross-Shard Mechanism*". International Conference on Very Large Data Bases, 2023.

- [**INFOCOM**] **Zicong Hong**, Song Guo, Enyuan Zhou, Jianting Zhang, Wuhui Chen, Jinwen Liang, Jie Zhang, and Albert Zomaya. "*Prophet: Conflict-Free Sharding Blockchain via Byzantine-Tolerant Deterministic Ordering*". IEEE International Conference on Computer Communications, 2023.

- [**DSN**] **Zicong Hong**, Song Guo, Rui Zhang, Peng Li, Yufeng Zhan, and Wuhui Chen. "*CYCLE: Sustainable Off-Chain Payment Channel Network with Asynchronous Rebalancing*". IEEE/IFIP International Conference on Dependable Systems and Networks, 2022.

- [**SoCC**] Wuhui Chen, Xiaoyu Qiu, **Zicong Hong**, Zibin Zheng, Hong-Ning Dai, and Jianting Zhang. "*Proactive Look-Ahead Control of Transaction Flows for High-Throughput Payment Channel Network*". ACM Symposium on Cloud Computing, 2022.

- [**ICDCS**] Leijie Wu, Song Guo, Yi Liu, **Zicong Hong**, Yufen Zhan, and Wenchao Xu. "*Sustainable Federated Learning with Long-term Online VCG Auction Mechanism*". International Conference on Distributed Computing Systems, 2022.

- [**INFOCOM**] **Zicong Hong**, Song Guo, Peng Li, and Wuhui Chen. "*Pyramid: A Layered Sharding Blockchain System*". IEEE International Conference on

Computer Communications, 2021.

- [**ICDCS**] Yi Liu, Leijie Wu, Yufeng Zhan, Song Guo, and **Zicong Hong**. "*Incentive-Driven Long-term Optimization for Edge Learning by Hierarchical Reinforcement Mechanism*". International Conference on Distributed Computing Systems, 2021.

- [**WCNCW**] Zhen Zhang, Qingqing Li, Wuhui Chen, and **Zicong Hong**. "*Distributed resource allocation for NOMA-based mobile edge computing with content caching*". IEEE Wireless Communications and Networking Conference Workshops, 2021.

- [**ICPP**] Jianting Zhang*, **Zicong Hong***, Yufen Zhan, Song Guo, and Wuhui Chen (*The first two authors have equal contribution). "*SkyChain: A Deep Reinforcement Learning-Empowered Dynamic Sharding Blockchain System*". International Conference on Parallel Processing (**Best paper runner-up prize**), 2020.

- [**WOWMOM**] Hui Lin, Zetao Yang, **Zicong Hong**, Shenghui Li, and Wuhui Chen. "*Smart contract-based hierarchical auction mechanism for edge computing in blockchain-empowered IoT*". IEEE 21st International Symposium on A World of Wireless, Mobile and Multimedia Networks, 2020.

- [**OJCS**] Jiahang Zhou, Yanyu Chen, **Zicong Hong**, Wuhui Chen, Yue Yu, Tao Zhang, Hui Wang, Chuanfu Zhang, and Zibin Zheng. "*Training and Serving System of Foundation Models: A Comprehensive Survey*". IEEE Open Journal of the Computer Society, 2024.

- [**TC**] Jianting Zhang, Wuhui Chen, **Zicong Hong**, Gang Xiao, Linlin Du, and Zibin Zheng. "*Efficient Execution of Arbitrarily Complex Cross-shard Contracts for Blockchain Sharding*". IEEE Transactions on Computers, 2024.

- [**TMC**] Yi Liu, Song Guo, Yufeng Zhan, Leijie Wu, **Zicong Hong**, and Qihua Zhou. "*Chiron: A Robustness-Aware Incentive Scheme for Edge Learning Via Hierarchical Reinforcement Learning*". IEEE Transactions on Mobile Computing, 2024.

- [**NETWORK**] **Zicong Hong**, Xiaoyu Qiu, Jian Lin, Wuhui Chen, Yue Yu, Hui Wang, Song Guo, and Wen Gao. "*Intelligence-Endogenous Management Platform for Computing and Network Convergence*". IEEE Network, 2023.

- [**TKDE**] Enyuan Zhou, **Zicong Hong**, Yang Xiao, Dongxiao Zhao, Qingqi Pei, Song Guo, and Rajendra Akerkar. "*MSTDB: A Hybrid Storage-empowered Scalable Semantic Blockchain Database*". IEEE Transactions on Knowledge and Data Engineering, 2023.

- [**TPDS**] Zhongteng Cai, Junyuan Liang, Wuhui Chen, **Zicong Hong**, Jianting Zhang, Hong-Ning Dai, and Zibin Zheng. "*Benzene: Scaling Blockchain with Cooperation-Based Sharding*". IEEE Transactions on Parallel and Distributed Systems, 2023.

- [**TMC**] Leijie Wu, Song Guo, **Zicong Hong**, Yi Liu, Wenchao Xu, and Yufeng Zhan. "*Long-Term Adaptive VCG Auction Mechanism for Sustainable Federated Learning With Periodical Client Shifting*". IEEE Transactions on Mobile Computing, 2023.

- [**JSAC**] **Zicong Hong**, Song Guo, and Peng Li. "*Scaling Blockchain via Layered Sharding*". IEEE Journal on Selected Areas in Communications, 2022.

- [**NETWORK**] Leijie Wu, Song Guo, Junxiao Wang, **Zicong Hong**, Jie Zhang, and Yaohong Ding, "*Federated Unlearning: Guarantee the Right of Clients to Forget*". IEEE Network, 2022.

- [**TSC**] Ting Cai, **Zicong Hong**, Shuo Liu, Wuhui Chen, Zibin Zheng, and Yang Yu. "*SocialChain: Decoupling Social Data and Applications to Return*

Your Data Ownership". IEEE Transactions on Services Computing, 2021.

- [**TETC**] Yufeng Zhan, Jie Zhang, **Zicong Hong**, Leijie Wu, Peng Li, and Song Guo. "*A Survey of Incentive Mechanism Design for Federated Learning*". IEEE Transactions on Emerging Topics in Computing, 2021.

- [**IOTJ**] Weikun Zhang, **Zicong Hong**, and Wuhui Chen. "*Hierarchical Pricing Mechanism with Financial Stability for Decentralized Crowdsourcing: A Smart Contract Approach*". IEEE Internet of Things Journal, 2020.

Acknowledgments

I would like to express my heartfelt gratitude to my mother, whose unwavering love, support and encouragement have been the bedrock of my academic journey. Her belief in my abilities, even when I doubted myself, has been a constant source of inspiration. She has been my pillar of strength, always offering words of encouragement and pushing me to overcome obstacles. Her sacrifices and unwavering belief in my potential have fueled my determination to succeed. I am truly blessed to have such a remarkable woman in my life, whose unwavering support has helped shape the person I am today. Thank you, Mum, for being my rock and for instilling in me the values of perseverance, resilience and dedication.

I am immensely grateful to my supervisor, Prof. Song Guo, for his invaluable guidance, expertise and patience. His mentorship and insightful feedback have been instrumental in shaping the direction of this research. I would also like to express my sincere gratitude to Prof. Wuhui Chen and Prof. Peng Li for their valuable input, constructive suggestions and scholarly discussions, which have greatly enriched my understanding of the subject matter.

I would also like to thank my friends for their unwavering support, motivating me during challenging times and providing a much-needed balance to my academic pursuits.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Blockchain, represented by Bitcoin [61], Ethereum [87] and Hyperledger Fabric [4], has attracted growing attention in the area of supply chains [27], high performance computing [56], and search engine [48]. 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 [3]. 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 [61, 87] 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 [82]. 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. 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 [54, 46, 89, 83, 2, 16, 93, 66, 35]. Moreover, for the industry, many blockchains are currently being upgraded to a sharding architecture. For example, Zilliqa has implemented sharding on its mainnet [79] and Ethereum plans to support sharding in its Eth2 upgrade in 2022 [24].

However, the existing works for blockchain sharding face three challenges as follows.

First, 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. More seriously, each transaction may involve more accounts in practice (see subsection 3.3.2.) To commit a cross-shard transaction, the existing sharding works [89, 83] divide it into several sub-transactions, each of which is handled by the associated shard. It seriously degrades the transaction throughput and multiplies the confirmation latency in a sharding system. Therefore, as will be discussed in chapter 3, we propose 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. These nodes store the blockchains of multiple shards, thus they can verify and execute the cross-shard transactions directly.

Second, the existing cross-shard mechanisms perform well below expectations for the smart contracts. According to section 4.6, an existing blockchain system with 32 shards performs even worse than a non-sharding blockchain when meeting the actual workload for smart contracts in Ethereum. This poor performance results from inherent conflicts among cross-shard transactions and the independent and random scheduling for cross-shard transactions in different shards. To eliminate the high abort rates caused by in-deterministic events of race conditions, as will be discussed in chapter 4, a sharding blockchain with global deterministic order for conflict-free smart contracts, which we call *deterministic sharding*. Its basic idea is to introduce

predetermined serial global order for all transactions including single-shard and cross-shard ones in the system.

Third, blockchain database has attracted widespread attention but still suffers from poor scalability due to its underlying non-scalable blockchain. While blockchain sharding is necessary for a scalable blockchain database, it is challenging to support cross-shard database services, such as cross-shard queries or inter-shard workload balancing, through the traditional on-chain manner. This is because they often need massive cross-shard data exchange rather than a simple cross-shard balance check like the conventional blockchain, which places a heavy burden on the consensus involving all nodes in the related shards. Therefore, as will be discussed in chapter 5, we propose GriDB, the first scalable blockchain database that distributes tables to different shards and provides efficient cross-shard database services. It is based on an authenticated data structure (ADS)-based off-chain execution, which can delegate cross-shard communication-intensive tasks to a few nodes in a verifiable manner.

The rest of this thesis presents the research motivation, system designs, protocol implementation and evaluation of my existing works, as well as the future research schedule to complete my thesis and research programme.

Chapter 2

Background

2.1 Preliminary for 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.

Transaction Model. There are two major types of transaction models, i.e., the Unspent Transaction Output (UTXO) model [61] and the account/balance model [87]. 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.

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 [4]. 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 [61]. 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.

2.2 Preliminary for Blockchain Sharding

Sharding is an idea originating from the database partitioning technique that divides a very large database into much smaller parts named *shards* [13]. 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.

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.

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.

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.

2.3 Related Work for Blockchain Sharding

RSCoin [15] 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.

ELASTICO [54] 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 [66] is a static program analysis for blockchain sharding and is built on top of a similar partial sharding blockchain named Zilliqa [79]. 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.

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 [46] 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 [2] 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 [89] 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 trans-

formed into single-shard transaction and processed in the output shard. Monoxide [83] 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 [62] 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 [78]. 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.* [78] but has a more hierarchical sharding structure to distribute the burden among layers. Moreover, Zhang *et al.* present Haechi, a cross-shard protocol immune to front-running attacks [92]. Qi *et al.* propose a sharding blockchain system that enables efficient execution of complex

cross-shard smart contracts [69]. Huang *et al.* propose a dedicated fine-tuned lock protocol which enables real-time processing of the affected transactions during account migration [39]. Jiang *et al.* propose Sharon, a sharding protocol that processes cross-shard transactions via shard rotation rather than transaction division [42].

Chapter 3

Pyramid: A Layered Sharding Blockchain System

Basic idea. This work proposes a new blockchain sharding schema, named *layered sharding*, to improve scalability through sharding while processing cross-shard transactions efficiently. Its main idea is to allow the shards to overlap, which means that some nodes can be located in more than one shard as shown in Fig. 3.1(b). These nodes store the blockchains of multiple shards, and thus they can verify and execute the cross-shard transactions directly. For example, as shown in Fig. 3.1(b), Node 3 can verify and execute the cross-shard transaction involving Shards 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, 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.

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.* Traditional sharding works adopt Nakamoto or Byzantine

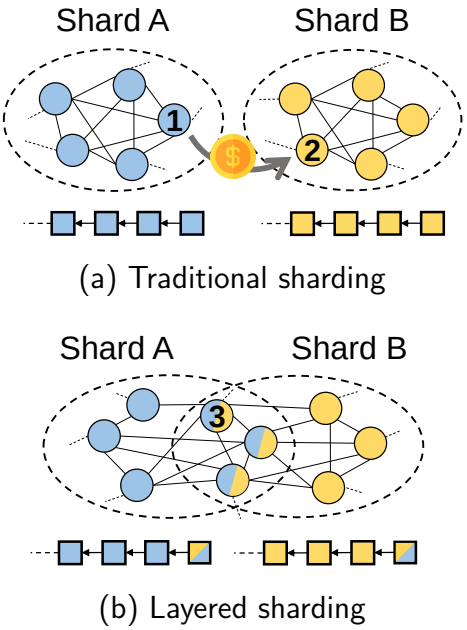


Figure 3.1: Illustration for different blockchain sharding systems.

consensus for the 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 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 the assignment of shards for the layered sharding, which determines the system performance and the level of security.

To illustrate the performance degradation that cross-shard transactions bring to traditional blockchain sharding, we perform a first experiment based on the cross-shard mechanism in Monoxide [83]. Divide each cross-shard transaction related to K shards

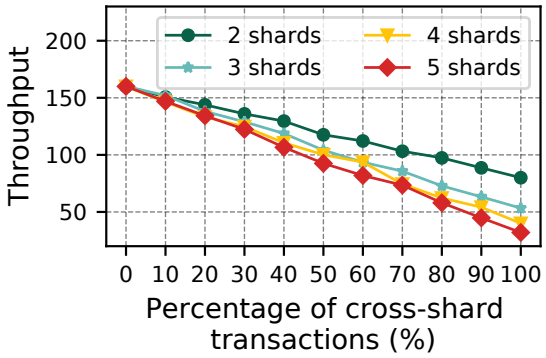


Figure 3.2: Performance of sharding with different proportion of cross-shard transactions over a function of number of their related shards.

into at least K sub-transactions. Note that the number of sub-transactions may be more than K, which will be discussed in section 3.3.2. As shown in 3.2, if there are more cross-shard transactions, or if each cross-shard transaction involves more shards, the transaction throughput of the sharding system decreases.

In practice, according to statistics [89], more than 96% of transactions are cross-sharded in a sharding system. Moreover, based on the data provided by XBlock [96], we conduct an analysis for Ethereum from 30 July 2015 to 6 July 2019 and find that more than 15% of smart contract-related transactions are composed of more than 1 step, and the average number of accounts in a transaction is 3.35. Furthermore, the proportion of multi-step transactions is increasing over time due to the popularity of more complex smart contracts.

To this end, this work 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 between 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 across multiple shards in a single round. The consensus consists of a collective signature-based inter-shard collaboration to commit the cross-shard block.

- We present an optimisation framework for layered sharding. We first analyse the transaction structure and node resource based on the observation of blockchain systems. We then formulate a transaction throughput maximization problem with security and resource constraints, 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 terms of throughput and latency. It improves the throughput by up to 3.2 times compared to the existing works and achieves about 3821 transactions per second (TPS) for 20 shards.

3.1 The Pyramid Model

3.1.1 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 [54, 46, 89], 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 [11]. In particular, the messages sent by a node can reach any other nodes with optimistic, exponentially-increasing time-outs.

3.1.2 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 subsection 3.2.8. 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 subsection 3.2.7. We leave the transaction model with more semantic information such as blockchain database [98, 49] to future works.

3.1.3 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.

For example in Fig. 3.3, 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

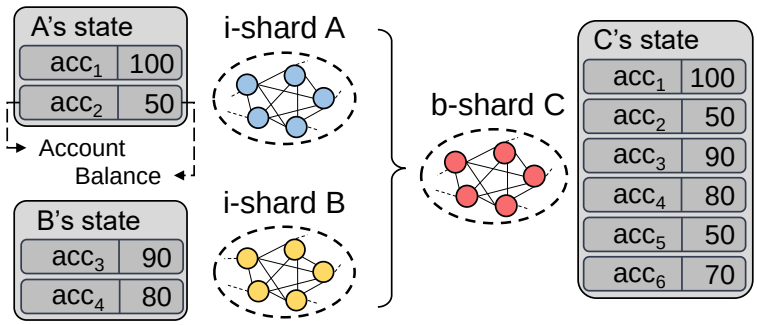


Figure 3.3: Illustration for a layered sharding for i-shard A, B and b-shard C.

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.

3.2 Architecture

3.2.1 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 (subsection 3.3.2), the nodes are assigned to shards to construct a layered sharding system (subsection 3.2.2.)

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 (subsection 3.2.3) composed of cross-shard transactions and commit them via a cooperative cross-shard

consensus (subsection 3.2.4). Finally, the state of an i-shard can be updated based on its block and the involved cross-shard blocks.

Moreover, subsection 3.2.5 solves the conflict among cross-shard blocks proposed by b-shard and the blocks proposed by the i-shards in the same round. subsection 3.2.6 fills the gap left by the i-shards which do not have the corresponding b-shard.

3.2.2 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 subsection 3.3.2.

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 [82], e.g., the verifiable random function [57], verifable delay function [8], and trusted execution environment [16], similar to that of other sharding systems [46, 89]. 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 [45, 46, 89].

4) Assignment. Each admitted node is assigned to an i-shard or b-shard randomly

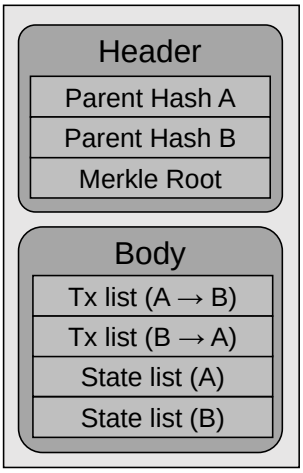


Figure 3.4: Illustration for a cross-shard block for i-shard A and B.

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.

3.2.3 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. 3.4 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 $\rightarrow$ B) and Tx list (B $\rightarrow$ 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.

3.2.4 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. 3.5 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. 3.4. 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

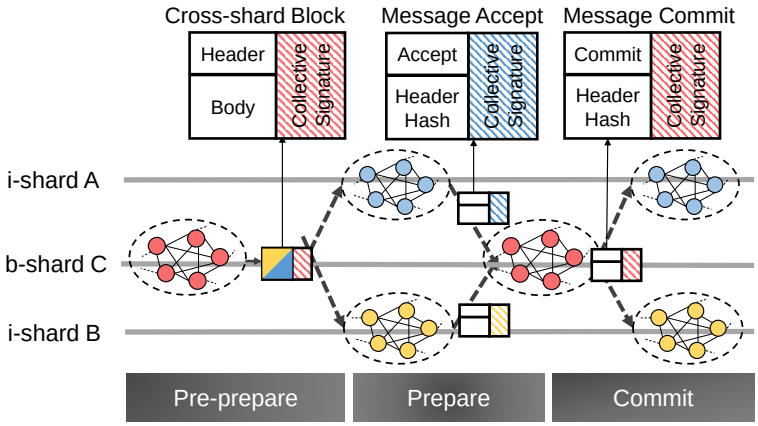


Figure 3.5: Illustration for cooperative cross-shard consensus.

a decentralized group of nodes can co-sign a multisignature, such as CoSi [76], a scalable protocol that can efficiently scale to thousands of nodes, and Boneh-Lynn-Shacham (BLS) [9] 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 subsection 3.2.5.

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 [45], 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 subsection 3.3.1), the honest nodes and some trusted infrastructure can also help to relay the inter-shard messages.

3.2.5 Conflicting Detection

In subsection 3.2.4, 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 [66], 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. 3.6(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. 3.6(b), the two blocks are conflicting since the balance of acc_2 is

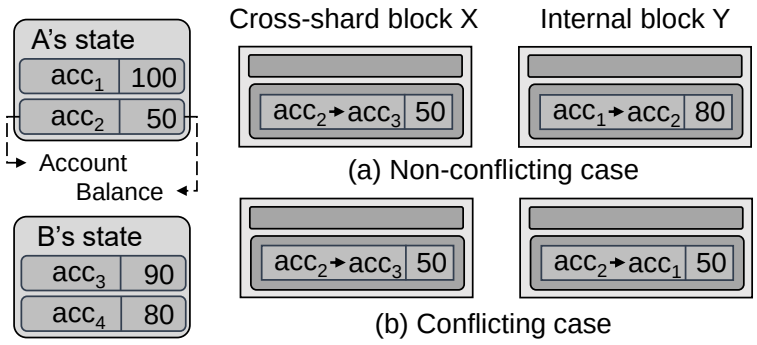


Figure 3.6: Illustration for the non-conflicting blocks and the conflicting blocks.

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.

3.2.6 Relay Mechanism

In subsection 3.2.4, each b-shard can propose cross-shard blocks related to at most its related i-shards. It raises the problem 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 [83] 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 subsection 3.2.4

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. 3.4 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.

3.2.7 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 section 3.3.2. 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 [87]. 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.

3.2.8 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 [61] and Litecoin [67].

In a sharding system for the account/balance model, each shard stores a proportion of accounts and their balances (see subsection 3.1.3.) In comparison, in a sharding system for the UTXO model [46, 89, 16], 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. 3.4 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. 3.3, 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 subsection 3.2.4. 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 subsection 3.2.6 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 [89] 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 subsection 3.2.4 since it stores all inputs for the transaction.

3.3 Analysis

3.3.1 Security Analysis

Similar to other blockchain sharding works [46, 89, 16, 3], 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 [2, 16]. 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.*

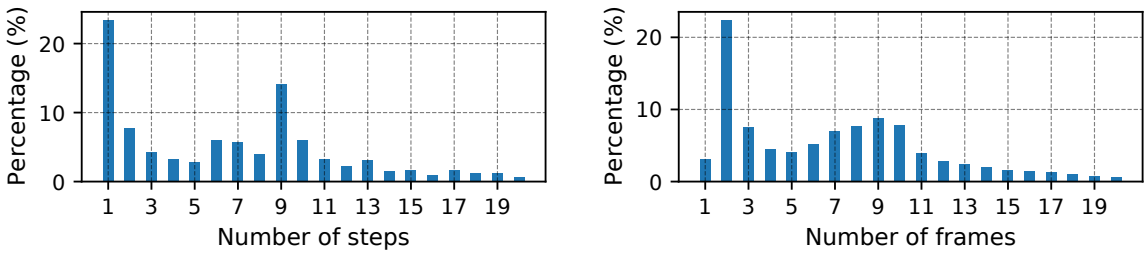
Proof. According to the system model in subsection 3.1.1, because the nodes are connected by a partially synchronous network and each shard has no more than $v < \frac{1}{3}$ malicious nodes, the BFT protocol adopted as the intra-shard consensus of each shard can achieve liveness. According to Theorem 1, each shard agrees on the same block in each round. Therefore, no malicious nodes can block the consensus indefinitely

and 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 subsection 3.1.1. 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 [34] 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.

3.3.2 Performance Analysis

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).



(a) Transaction distribution (b) Frame distribution for 8 shards

Figure 3.7: Illustration for transaction distribution and frame distribution in Ethereum from Aug. to Sep. 2021.

Transaction Model

We define *transaction distribution* for a blockchain system as the distribution of transactions with different steps $\alpha = \{\alpha_1, \alpha_2, \cdots, \alpha_A\}$ 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. 3.7(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 [97].

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*

$$\beta_s = \sum_{k=1}^{A} \alpha_k \binom{k}{k+1-s} \frac{(S-1)^{s-1}}{S^k}. \tag{3.1}$$

Proof. In a sharding system with S shards, for a transaction involving k steps, where $1 \leq k \leq A$, 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

$$Pr(X=s) = \binom{k}{k+1-s} \frac{S(S-1)^{s-1}}{S^{k+1}} = \binom{k}{k+1-s} \frac{(S-1)^{s-1}}{S^k}. \tag{3.2}$$

Based on Eq. (3.2) and the distribution of transactions involving different number of accounts α, we can get Eq. (3.1). □

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.

Fig. 3.7(b) illustrates the frame distribution for a sharding system with 8 shards which is derived from the transaction distribution in Fig. 3.7(a). 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.

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 section 3.2, 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 $\mathcal{N} = \{N_1, N_2, \cdots, N_L\}$

Consensus Model

We model the consensus in the layered sharding based on [52] which models several BFT consensus protocols including PBFT, Zyzzyva, and Quorum in a non-sharding

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

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 [12].

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 subsection 3.2.4, the consensus of layered sharding is composed of three phases of collective signing. Each one 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 [76]. 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.

Security Model

According to subsection 3.3.1, 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 [89], the probability for forming an unsafe shard can be approximated as

$$P[X \geq \lceil n/3 \rceil] = \sum_{x=\lceil n/3 \rceil}^{n} \binom{n}{x} f^x (1-f)^{n-x}. \tag{3.3}$$

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

$$SP[X \geq \lceil n/3 \rceil] < 2^{-\lambda}. \tag{3.4}$$

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 [89] is also applicable.

Problem Formulation

A sharding strategy in Pyramid is 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 $\sum_{1 \leq i \leq L} d_i = S$. According to subsection 3.2.4, 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

$$\begin{aligned} TPS^{layer} = \frac{\lfloor K/\chi \rfloor}{T} (d_1 (\underbrace{\beta_1 + \frac{\beta_2}{2} + \cdots + \frac{\beta_B}{B}}_{(1)}) \\ + \sum_{2 \leq i \leq L} d_i (\underbrace{\sum_{s=1}^{B} \beta_s (\frac{i}{S})^{s-1}}_{(2)} + \underbrace{\sum_{s=2}^{B} \beta_s \sum_{j=1}^{s-1} \frac{j}{s} (\frac{i}{S})^{j-1} \frac{S-i}{S}}_{(3)})), \end{aligned} \tag{3.5}$$

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

$$\max_{\mathbf{d}} \quad TPS^{layer} \tag{3.6}$$

$$s.t. \quad \sum_{l \le i \le L} d_i \le \frac{N_l}{n}, \forall 1 \le l \le L \tag{3.7}$$

$$SP[X \ge \lceil n/3 \rceil] < 2^{-\lambda} \tag{3.8}$$

$$d_i \in \mathbb{N}, \forall d_i \in \mathbf{d}. \tag{3.9}$$

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 [46, 89, 38], 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

$$\max_{\mathbf{d}} \quad TPS^{layer} \tag{3.10}$$

$$s.t. \quad \sum_{l \le i \le L} d_i \le \frac{N_l}{N} S, \forall 1 \le l \le L \tag{3.11}$$

$$\sum_{1 \le i \le L} d_i = S \tag{3.12}$$

$$d_i \in \mathbb{N}, \forall d_i \in \mathbf{d}, \tag{3.13}$$

which can be efficiently solved by well-developed branch-and cut algorithms [36] or dynamic programming [6].

3.4 Evaluation

3.4.1 Implementation

We implement a prototype of Pyramid in Go [31] based on Ethereum [23]. For the intra-shard consensus, we adopt a collective signature-based BFT in Harmony [32]. The communication among nodes or shards is based on libp2p [50]. 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 [83] and the second one uses the transfer mechanism in RapidChain [89].

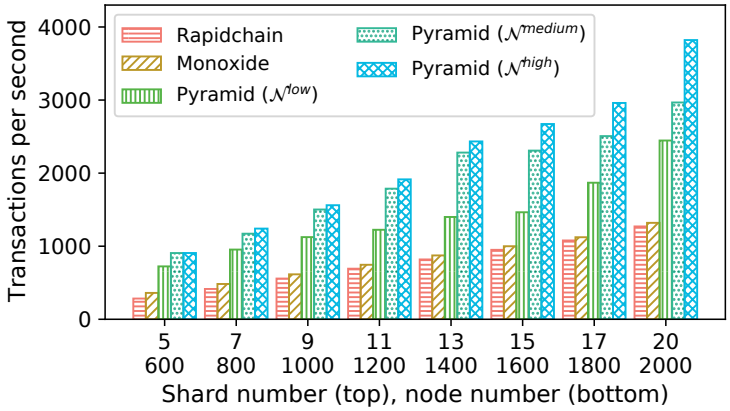


Figure 3.8: Transaction throughput for layered sharding with different shard number and node distribution.

3.4.2 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 [96], 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 section 3.3.2 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 section 3.3.2, 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.

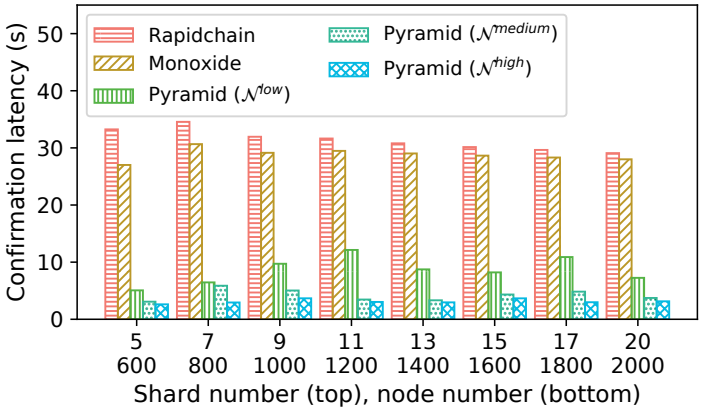


Figure 3.9: Latency for layered sharding with different shard number and node distributions.

3.4.3 Transaction Throughput & Latency

Fig. 3.8 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 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. 3.9 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

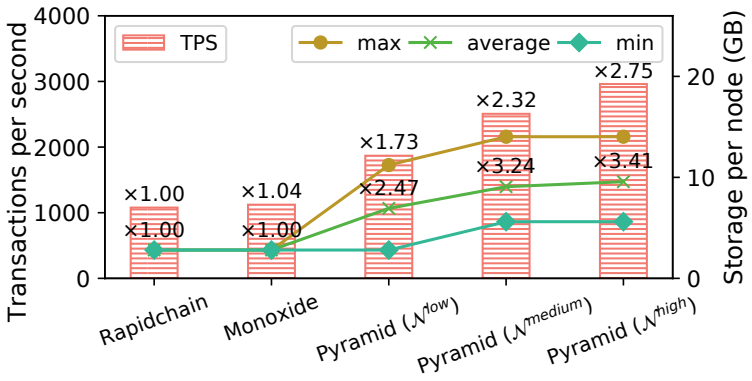


Figure 3.10: Trade-off of transaction throughput and storage overhead for layered sharding with different node distribution.

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.

3.4.4 Storage Overhead

Fig. 3.10 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 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

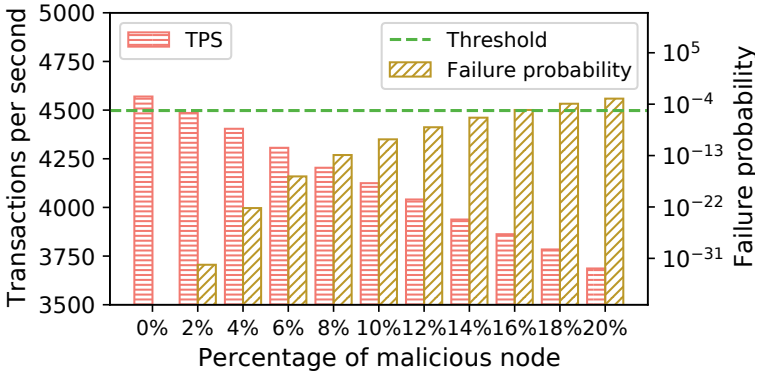


Figure 3.11: Transaction throughput and failure probability for different malicious node fraction in a layered sharding system with 17 shards.

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 [46, 5]. Second, as discussed in section 3.3.2, 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.

3.4.5 Security

Fig. 3.11 shows the performance of the layered sharding with different percentage of malicious nodes. To satisfy the security requirement mentioned in subsection 3.4.2, the failure probability computed by Eq. (3.4) should be smaller than the threshold

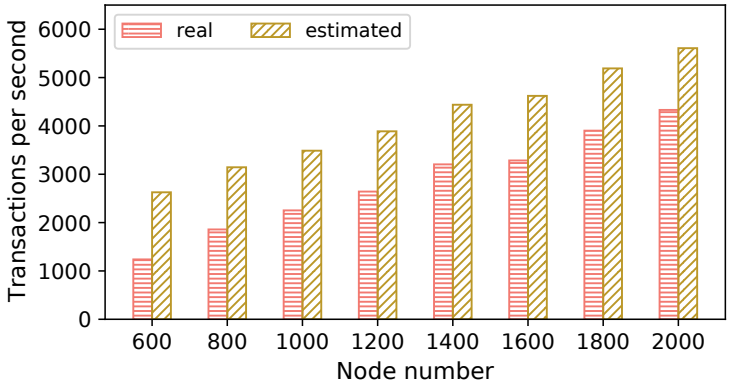


Figure 3.12: Real and estimated transaction throughput for optimal sharding strategy with different node number in the same node distribution $\mathcal{N}^{high}$.

2^{-17}, i.e., the green dotted line illustrated in the figure. 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.

3.4.6 Sharding Strategy

Fig. 3.12 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, 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

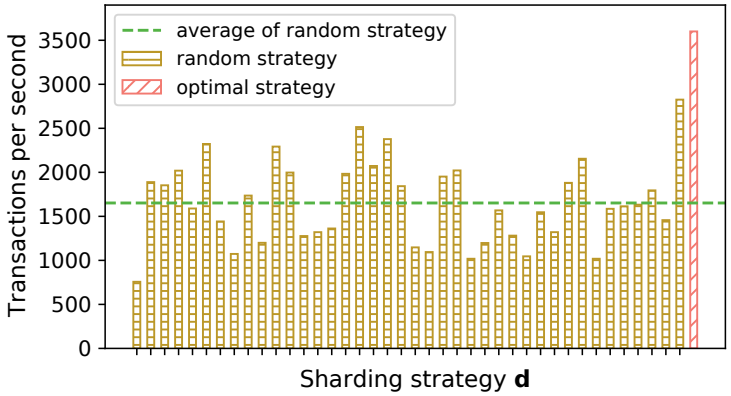


Figure 3.13: Transaction throughput for different sharding strategy in the same node distribution $\mathcal{N}^{medium}$.

do not have the corresponding b-shards. However, Eq. (3.5) 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. 3.13. 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.

3.4.7 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. 3.14 and Fig. 3.15.

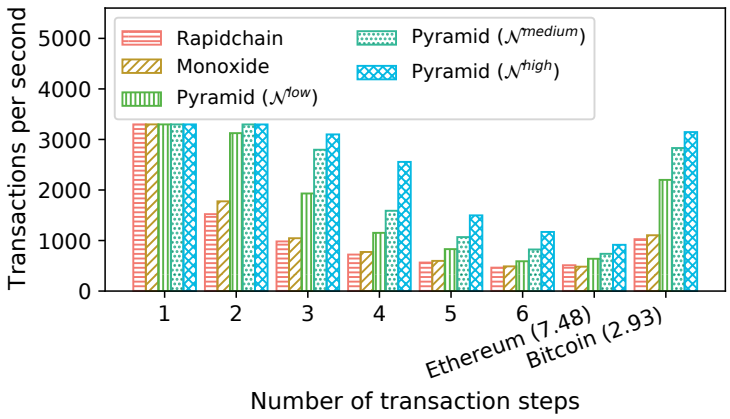


Figure 3.14: Transaction throughput for different number of transaction steps in a layered sharding system with 17 shards.

As shown in Fig. 3.14, when the transactions have more steps, the throughput of all sharding systems is reduced. This is because a transaction with more steps may have more frames after the accounts are sharded according to section 3.3.2. 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. 3.15, 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.

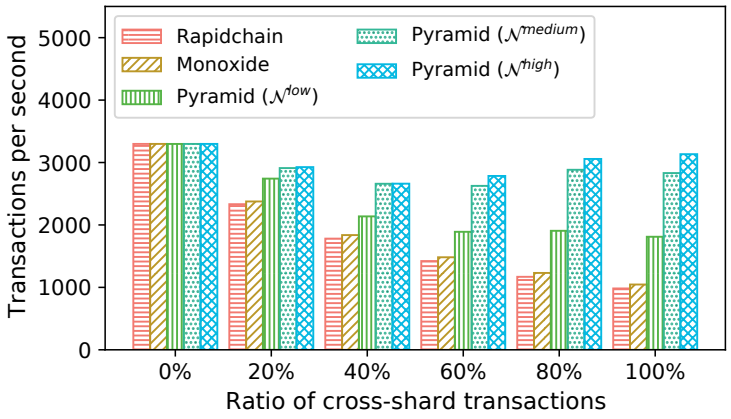


Figure 3.15: Transaction throughput for different ratio of cross-shard transactions in a layered sharding system with 17 shards.

Chapter 4

Prophet: Conflict-Free Sharding Blockchain via Byzantine-Tolerant Deterministic Ordering

The prosperity of blockchain facilitates decentralized applications (dApps), e.g., decentralized exchanges [86] and non-fungible token [84]. In Ethereum, the number of contract calls per day has more than tripled to over 3 million from Jun. to Sep. in 2021 [14]. However, the poor scalability of the existing blockchains, 7 transaction per seconds (TPS) in Bitcoin [61] and 15-45 TPS in Ethereum [87], cannot satisfy this growing demand for dApps. This is because their consensus requires all nodes to validate and execute every transaction, which aggravates the scalability problem and restricts smart contracts from more users and the dApps with more complex logic.

Sharding is one of the most promising technologies for scalability [82]. It divides nodes into multiple consensus groups called *shards* and distributes transactions to shards to process in parallel. The technology has been paid close attention by the academia [54, 46, 89, 83, 38, 66, 35, 40, 68]. For the industry, some blockchains are being or have been upgraded to a sharding architecture, such as Zilliqa [79] and Eth2

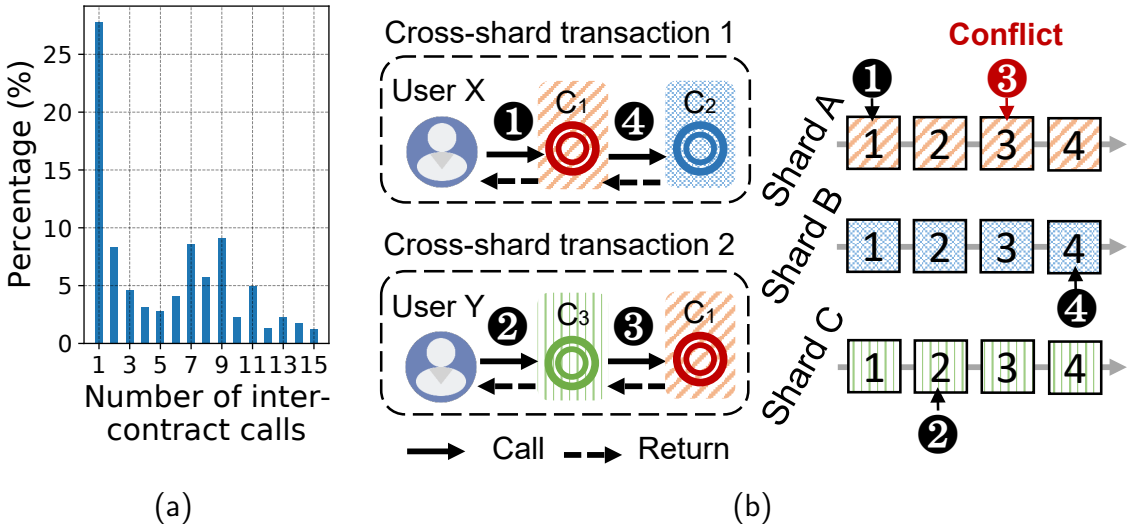


Figure 4.1: (a) Percentage of Ethereum transactions with different number of inter-contract calls from Oct. 2020 to May. 2021, (b) Illustration for conflicting cross-shard transactions. Each concentric circle represents a smart contract. Three contracts are located in three different shards. Each circled number represents the round at which a sub-transaction is committed.

upgrade in Ethereum [24].

While increasing the throughput in proportion to the number of shards, sharding technology introduces *cross-shard transactions*, which are transactions involving the smart contracts in multiple shards. More seriously, similar to the current software composed of numerous programs, a dApp often requires the cooperation of several contracts, significantly increasing the number and complexity of cross-shard transactions. As shown in Figure 4.1(a), more than 70% of Ethereum [87] transactions for smart contracts include more than 2 inter-contract calls and the average number is 8.94. After introducing sharding, the contracts are located in different shards thus the inter-contract calls result in massive cross-shard transactions. To guarantee the atomicity and consistency of cross-shard transactions, each sharding system needs a cross-shard mechanism. The mechanism divides each cross-shard transaction into several sub-transactions, each of which corresponds to a shard, and commits each sub-transaction to the corresponding shard.

Unfortunately, we observe that the existing cross-shard mechanisms perform well below expectations in practice. As evaluated in section 4.6, an existing blockchain with 32 shards performs even worse than a non-sharding one when meeting the Ethereum transactions. It upends our usual understanding on sharding and drives us to reconsider the technology of sharding. This poor performance results from inherent conflicts among cross-shard transactions and the independent and random scheduling for cross-shard transactions in different shards. In particular, multiple cross-shard transactions may access the state of the same smart contract (e.g., Cxn.[1] 1 and 2 access Contract C_1 in Figure 4.1(b)). Although Cxn. 1 is issued first, the state of C_1 can be modified by Cxn. 2 before Cxn. 1 is committed. The existing sharding systems mainly adopt two-phase locking (2PL) (e.g., [46, 16, 2]) or optimistic concurrency control (OCC) (e.g., [83, 38]) to avoid conflict and guarantee serializability of transactions. However, as evaluated in section 4.6, more than 50% of transactions are aborted or rolled back due to race conditions, because real smart contract workloads contain frequent read-write conflicts. Therefore, both 2PL and OCC exhibit high abort rates and strongly limit the performance of blockchain sharding.

To eliminate the high abort rates caused by non-deterministic race conditions, an intuitive idea is to introduce a predetermined serial global order for pending transactions before processing them. The idea is inspired by distributed and deterministic database with a sequencing layer, which collects all database transactions for producing a global order before database execution [80, 53, 51, 26, 1]. However, the sequencing layer in distributed databases is designed with a strict assumption that the layer contains trusted machines for storing the whole database state, which is far from trivial for the blockchain. Due to the intertwining of the information isolation among shards (i.e., each node only stores a proportion of contracts) and Byzantine environment (i.e., blockchain nodes do not trust each other), there are no trusted

[1] In this paper, for simplification, we use Cxn. to denote cross-shard transaction, Txn. to denote single-shard transaction, and Blk. to denote block.

nodes to predetermine such an order for transactions involving the state of different shards.

Therefore, we propose PROPHET, a conflict-free sharding blockchain, based on a new idea named *Byzantine-tolerant deterministic ordering*. Specifically, to overcome the challenge of information isolation, the nodes from different shards are allowed to form self-organizing coalitions to pre-execute pending transactions, including single-shard and cross-shard ones, for prerequisite information about ordering. Then, a random shard is delegated to sequence the pre-executed transactions for a global order based on prerequisite information. To deal with Byzantine failures in blockchain, a shard-cooperation proof sharing is proposed to verify and correct untrusted pre-execution results without interruption of consensus. With such an order, transactions in different shards can be executed and committed orderly without conflicts. The new architecture will not break any decentralization principle of blockchain.

The contributions of this work are summarized as follows.

- We propose an idea of Byzantine-tolerant deterministic ordering and develop a conflict-free sharding blockchain named PROPHET, minimizing the number of transaction aborts caused by non-deterministic contract contention.
- On top of the shards, PROPHET introduces two new types of parties, i.e., sequence shard and reconnaissance coalition, with a little additional overhead and designs a new cooperative consensus for a global order based on the cooperation and joint supervision among shards.
- Based on the characteristics of smart contracts, such as inter-contract calls and contract instructions, we present the designs of fine-grained ordering, asynchronous correction, and parallel pre-execution for efficiency.
- We develop a prototype for PROPHET and conduct a comprehensive evaluation. PROPHET improves the throughput by 3.11× (i.e., 1203 TPS) on 1 millions

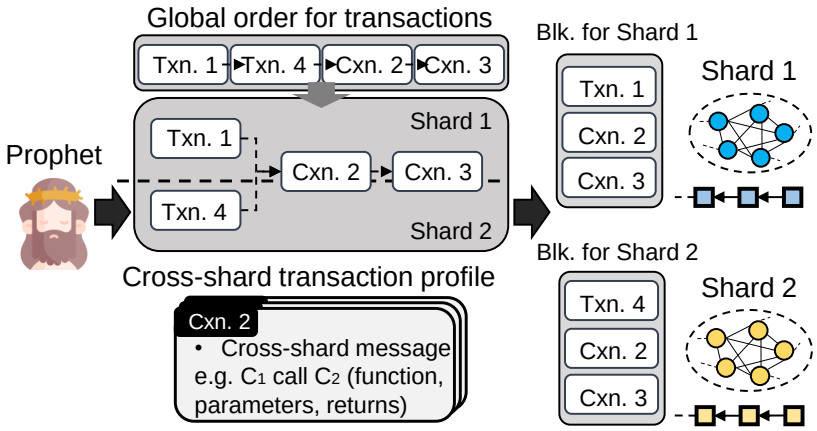


Figure 4.2: An ideal cross-shard mechanism for sharding.
Ethereum transactions compared with state-of-the-art sharding systems.

4.1 Strawman: An Ideal Cross-Shard Mechanism

We first describe an ideal cross-shard mechanism that ignores the decentralized and Byzantine environment of blockchain. It motivates the main design of PROPHET in section 4.2. This mechanism guarantees that no transactions are aborted.

As shown in Figure 4.2, the mechanism introduces a new node called *prophet*. Assume that the prophet is fully trusted and has infinite computing power and storage capacity. For each consensus round, it first validates and executes transactions sequentially for a global order of the transactions, such as 1423 in the figure. Next, according to the global order, the prophet generates a serial order for each shard's block. Specifically, for a single-shard transaction, since it only reads or writes the state of contracts in a shard, its execution only depends on the latest executed transaction in the shard. For a cross-shard transaction, it involves the state of multiple shards, thus its execution depends on several transactions from different shards. For example, in Figure 4.2, Cxn. 2 executes following Txn. 1 and Txn. 4. After validation and execution, the blocks generated by the prophet correspond to a global order which shows the data dependency of all transactions in this round.

Moreover, the prophet can record some meta information required for the execution of each cross-shard transaction in each shard, called *cross-shard transaction profile*. The profile includes all cross-shard inter-contract calls and their parameters and returns, called *cross-shard message*. The profile enables each shard to execute transactions without communicating with the other shards during execution.

Finally, the prophet sends the serial order and the profile to the corresponding shard with a signature. The shards only need to replicate and execute transactions one after the other based on their received serial orders. All transactions can be committed without any conflicts according to the global order.

Although the mechanism eliminates transaction aborts and requires minimal coordination among shards, the assumption of such a special node is too ideal. In particular, a fully trusted node violates the decentralized inherence of public blockchain and it has to locally maintain the whole state for all shards to pre-execute transactions. Thus, it raises a question about how to implement such a prophet in the decentralized and Byzantine environment of blockchain sharding. In the following, we present Prophet, which achieves a similar effect through the cooperation and supervision among shards in a distributed manner and without any trusted third party.

4.2 Byzantine-Tolerant Deterministic Ordering for Blockchain Sharding

4.2.1 System Model & Threat Model

Similar to the existing blockchain sharding [54, 46, 89, 38], Prophet proceeds in *epochs*, each of which includes multiple rounds. It consists of a set of 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

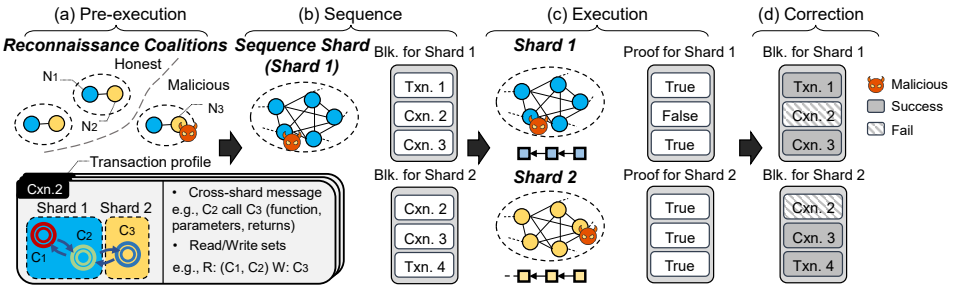


Figure 4.3: The architecture of PROPHET.

by a Byzantine adversary and may collide with each other and violate the protocols in arbitrary manners, e.g., denial of service, tampering, and forgery. All nodes are randomly divided into multiple shards. The nodes in each shard store a proportion of contracts and verify and execute the transactions involving the stored contracts. Besides, each lightweight client only stores its accounts and do not store any contracts. Since the world state of current blockchain is huge (e.g., total size in Ethereum is currently more than 130 GB and keeps increasing [72]), in PROPHET, neither a node or a client can store the state of all contracts.

4.2.2 Motivation & Overview

We start with the problems that the strawman in section 4.1 highlights and build up our design step by step.

According to section 4.1, the function of the prophet includes two tasks, i.e., pre-execution and ordering for pending transactions. Specifically, the pre-execution task requires the storing of the blockchain state, while the ordering task does not require it. It is because the pre-execution needs to output the prerequisite information about ordering (i.e., read/write sets and cross-shard messages) by executing transactions based on the blockchain state. In comparison, the ordering is based on the prerequisite information provided by the pre-execution and does not need to execute any transaction or store any contract; thus, it is *stateless*.

Based on their characteristics (i.e., requirements and workload), we delegate these two tasks to different parties as follows. First, since any node cannot store the state of all contracts (as discussed in subsection 4.2.1), to pre-execute all possible transactions, we introduce a new type of parties named *reconnaissance coalitions* in which nodes from different shards cooperate with each other to pre-execute pending transactions (see subsection 4.2.3). Nodes can freely form or dissolve reconnaissance coalitions, which are off-chain. PROPHET distributes a proportion of transaction fees to reconnaissance coalitions with successfully committed pre-executed transactions. (A more detailed analysis of incentives in reconnaissance coalitions will be left as our future work.) Second, the task of ordering is stateless and does not need intensive computation; thus, any node or shard can do it. For security, we let every shard take on the ordering task in turn, and we call the shard responsible *sequence shard* (see subsection 4.2.4). The sequence shard will be updated in each epoch for load balancing. With the help of these new parties, PROPHET proceeds in four phases, i.e., *pre-execution*, *sequence*, *execution*, and *correction*, for each round.

4.2.3 Phase 1: Pre-execution

During this phase, each reconnaissance coalition selects a disjoint set of pending transactions with some specific range of transaction hash and executes them one by one. In each reconnaissance coalition, if a node meets an inter-contract call to a contract located in another shard when pre-executing a transaction, it turns to the other nodes in the same reconnaissance coalition. For example, in Figure 4.3, when pre-executing Cxn. 2, since Contract C_3 belongs to Shard 2, Node N_1 needs to send the function call and parameters to and get the returns from Node N_2. Thus, each reconnaissance coalition can be regarded as an individual able to execute the single-shard or cross-shard transactions for its related shards.

In particular, each transaction reads from the current state of the blockchain, executes

its logic, and keeps the writes in a local write set. Since the change of each transaction is kept in local write set, the state read by each transaction is always the same. For each transaction, the reconnaissance coalition records its read/write set and its cross-shard messages during execution. To clarify our basic idea, we define the read/write set of a transaction as the addresses of smart contracts, which will be extended to a fine-grained one in subsection 4.3.2. Moreover, the serial execution will be extended to a more efficient one in subsection 4.3.4. The cross-shard message about a cross-shard inter-contract call includes its function name, parameters and return.

We emphasize that there is no guarantee that all reconnaissance coalitions are honest because their formation is free and cannot guarantee that all nodes in a reconnaissance coalition are honest. In a Byzantine environment, the read/write sets and cross-shard messages recorded by a reconnaissance coalition with malicious nodes will be wrong. For example, in Figure 4.3, there is a malicious reconnaissance coalition since there is a malicious node N_3 in the coalition. Since each successful pre-execution gains transaction fees, we assume that honest nodes tend to form and stay in reconnaissance coalitions involving the other honest nodes and leave coalitions involving malicious nodes (the detection of malicious nodes in a reconnaissance coalition will be discussed in subsection 4.2.6).

4.2.4 Phase 2: Sequence

After a reconnaissance coalition pre-executes a transaction, it passes the transaction and the corresponding transaction profile to the sequence shard. The sequence phase starts when the leader of the sequence shard receives enough (i.e., more than a predefined threshold) transactions.

For each round, in the sequence phase, based on the pre-execution results passed by the reconnaissance coalitions, the leader of the sequence shard can determine which transactions will be included in the global order. To avoid transaction conflicts, the

sequence shard only allows the transactions involving disjoint read/write sets to be packed into the order. In such an order, the transactions are processed as the same as they are in the pre-execution phase since they are not in conflict. Based on the order, the sequence shard generates a serial order for the block of every shard and provides a transaction profile for transactions included in the block. For example, in Figure 4.3, the sequence shard proposes two new blocks to Shard 1 and 2, respectively. Finally, based on the intra-shard consensus, the nodes in the sequence shard send the new blocks with a collective signature (such as CoSi [76] or BLS [9] in the existing sharding [46, 38]) to the shards.

The sequence shard has two characteristics. The first one is stateless, which means the nodes in the sequence shard can generate a global order depending on transaction profiles received from reconnaissance shards and without storing the state of all contracts and pre-executing transactions. The second one is trusted, which means each message published by the sequence shard is via an intra-shard consensus.

4.2.5 Phase 3: Execution

As discussed in subsection 4.2.3, in the pre-execution phase, the pre-execution results cannot be guaranteed because the reconnaissance coalitions may be malicious. Moreover, in the sequence phase, because the sequence shard is stateless and only responsible for ordering the transactions instead of validating, there can be invalid transactions or conflict events existing in the order. Therefore, the shards need to execute and validate the transactions included in the received blocks and compare them with the read/write sets and transaction profile.

During the execution, each shard runs an intra-shard consensus and executes all transactions based on the transaction profile. For example, in Figure 4.3, Shard 2 executes Cxn. 2 based on the function call from Contract C_2 to C_3 and the corresponding parameters in the transaction profile. If a shard finds that the read/write sets or

cross-shard messages of a transaction are different from those provided by the reconnaissance coalitions, it can mark the transaction as invalid. However, if the execution results of a transaction exactly match its transaction profile, the shard can mark it as valid. For example, Shard 1 marks Cxn. 2 as invalid if the cross-shard message or read/write set in the transaction profile is incorrect. The intra-shard consensus can guarantee that the result published by any shards is trusted.

4.2.6 Phase 4: Correction

The confirmation of a cross-shard transaction, i.e., a shard commits the transaction and updates the state of contracts based on the transaction, requires the proof generated by all the related shards of the transaction in the execution phase. At the end of each round, every shard shares its validation results (i.e., proof generated in the execution phase) with the other shards. The proof denotes the validity of each pre-executed transaction included in the global order. Each cross-shard transaction can be committed in a shard only when the shard receives the validity proof from all the other shards related to the transaction. For example, in Figure 4.3, Cxn. 2 is related to Shard 1 and 2, thus it cannot be committed without the proof of both these two shards. Since the proof of Shard 1 marks it as invalid, it will not be committed. In addition, the honest nodes in the reconnaissance coalition responsible for Cxn. 2 can leave the coalitions and mark the nodes responsible for the invalid part (contracts in Shard 1) as malicious.

4.2.7 Discussion

Different from the traditional blockchain sharding that only has the execution phase, PROPHET has three additional phases (The overhead will be analyzed in section 4.4). In each round, a deterministic global order for all transactions, including single-shard and cross-shard ones, can be generated and shared by all shards through these three

phases. Following the order, the shards can orderly execute and commit transactions and update the blockchain state without conflicts. The cooperation within reconnaissance coalitions solves the challenge of information isolation among shards, while the stateless ordering in the sequence shard and the inter-shard proof sharing in the final correction phase deal with Byzantine failures. A rigorous theoretical analysis is provided in section 4.4.

4.3 Design Refinement

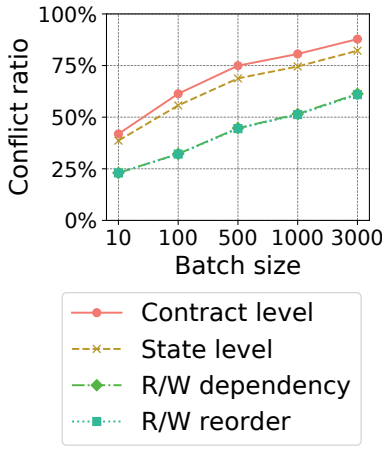
4.3.1 Parallelization of Sequencing and Execution

Problem of additional consensus. PROPHET introduces an additional sequence phase in each round. This phase requires an intra-shard consensus in the sequence shard, doubling the consensus time for each block.

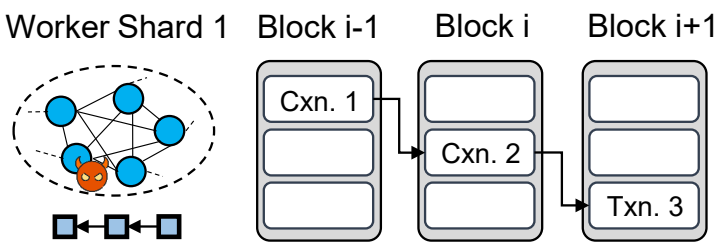
Design. To solve the problem, PROPHET parallelizes the sequence phase and execution phase. Specifically, the leader of the sequence shard can send a global order to the shards before the sequence shard validates the new order through consensus. Then, in the execution phase, the sequence shard validates the new global order and pre-execution results. An invalid order results in an invalid proof generated by the sequence shard. Thus, an invalid order proposed by a malicious leader of the sequence shard can be detected in the correction phase.

4.3.2 Fine-grained Ordering

Problem of coarse-grained ordering. In the above system, the reconnaissance coalitions simply define the read/write sets of transactions as the addresses of smart contracts in the pre-execution phase. Then, in the sequence phase, the sequence shard only pack the transactions that are not related to the same contracts. In such



(a)



Transaction	{Accessed contracts} – Shard
Cxn. 1	{C1, C2} – Shard 1, {C3} – Shard 2
Cxn. 2	{C2} – Shard 1, {C4} – Shard 3
Txn. 3	{C2} – Shard 1

(b)

Figure 4.4: (a) Conflict ratio of transactions in a batch with varying batch size in the pre-execution phase; (b) Transaction confirmation rule in asynchronous correction.

a coarse-grained manner, the transactions accessing the same contract are considered conflict and thus cannot be packed in a global order. If there are a majority of conflict transactions in the demand, the throughput of the sequence phase may become a bottleneck of PROPHET.

Observation. To illustrate the performance of the coarse-grained ordering in section 4.2 in practice, we collect the history of transactions from Nov-25-2019, Feb-17-2020, and Apr-19-2020 in Ethereum. Then, we execute the transactions in the batch of $10, 100, 500, 1000, 3000$ in parallel to simulate the pre-execution phase in PROPHET and evaluate their conflict ratio. We denote the approach by contract level. As shown in Figure 4.4(a), the result shows that the conflict ratio increases with the batch size and nearly 90% transactions are in conflict when the reconnaissance coalitions pre-execute 3000 transactions.

Design. Therefore, we propose a fine-grained read/write ordering approach that is composed of two following steps.

1) First, we design a fine-grained read/write set identification. We first redefine the read/write sets of a transaction as the blockchain storage that it reads or writes.

Then, the transactions related to the same contract can access different positions of its storage. For example, if two transactions access the same contract but read or write different variables of the contract, they are not in conflict. For such a fine-grained identification, we take a deep dive into smart contracts [87]. All contract fields and mappings are saved in blockchain storage and each transaction is a sequence of instructions among which `SSLOAD` and `SSTORE` are the two instructions for blockchain persistent storage read and write, respectively. Thus, during the pre-execution, the reconnaissance coalitions record the instructions `SSLOAD` and `SSTORE` and their corresponding addresses. As shown in Figure 4.4(a), the approach (denoted by state level) reduces the conflict ratio by about 5.34% compared with the contract-level approach.

2) Second, we design an ordering rule considering the read/write dependency for the sequence phase. If the sequence shard decides an order in which the execution result of any transaction will not influence the read/write sets of its following transactions, the execution of transactions in the execution phase will be the same as those in the pre-execution phase thus there are no conflicts in the order. To achieve it, we allow a transaction to have a read-after-read or write-after-read dependency with its previous transactions in the order. As shown in Figure 4.4(a), the approach (denoted by R/W dependency) reduces the conflict ratio by about 21.42% compared with the state level approach. Besides, we also evaluate a reorder rule [53]. In detail, for two transactions with read-after-write dependency, the rule can change their position if the new order does not violate the before ordering rule. However, it only reduces the conflict ratio by 0.21%.

4.3.3 Asynchronous Correction

Problem of synchronous correction. In the correction phase, for a shard, the validity of its related transactions can be proved and the state of its stored contracts can be updated only when the shard receives all proofs from the other shards. However,

in the practice, for each round, there is great uncertainty about both the consensus latency in each shard [47, 7] and the latency of cross-shard communication. This can result in an barrel effect, which means the round time of each shard in PROPHET will depend on the slowest shard.

Observation. The consensus latency has high variance for Proof-of-Work (PoW) protocols adopted by Bitcoin and Ethereum or Byzantine fault tolerance (BFT) protocols adopted by Hyperledger Fabric. For example, although Bitcoin theoretically produces one block every 10 minutes, for 5% of the time, Bitcoin's inter-block time is at least 30 minutes [7]. For PBFT, the consensus latency is uncertain because the state of network environment is often dynamic and elusive [47].

Design. To overcome the problem, PROPHET adopts an asynchronous correction design. Specifically, in the execution phase, before receiving the proof from the other shards, a shard can optimistically assume that all the transactions are valid. Next, it can update the state of its stored contracts based on the current block and move to the next round. When a shard receives an invalid proof for a previous transaction from another shard, this previous transaction will be invalidated. All the following transactions related to the contracts involved by the invalid transaction will be also invalidated. In other words, PROPHET has the following confirmation rule for transactions.

Rule 1. *A transaction $\mathcal{T}$ can be confirmed by a shard only when the shard receives all the related proofs of $\mathcal{T}$ and all the related proofs of the previous transactions related to $\mathcal{T}$.*

For example, Figure 4.4(b) shows three blocks, i.e., Block i-1, i, and i+1, of Shard 1. In these three blocks, there are three transactions, i.e., Cxn. 1, Cxn. 2 and Txn. 3, all of which access the same contract C_2 stored in Shard 1. Based on the confirmation rule of transactions, the confirmation of Txn. 3 depends on the proof of Shard 2 in Block i-1, the proof of Shard 3 in Block i, and the proof of Shard 1 in Block i+1.

4.3.4 Parallel Pre-Execution

Problem of serial pre-execution. The throughput of PROPHET also depends on the total pre-executed throughput of reconnaissance coalitions. The above system considers a serial pre-execution approach in which the nodes in each reconnaissance coalition execute transactions one by one. However, when the communication accounts for a higher portion than the contract execution as proved below, each node may spend a lot of time on the communication of cross-shard inter-contract calls, keeping its CPU idle most of the time and restricting the pre-executed transaction throughput.

Observation. To find the main bottleneck of pre-executed throughput, we evaluate the simplest cooperation mode for a reconnaissance coalition as shown in Figure 4.5(a). Specifically, the nodes in the reconnaissance coalition executes transaction one by one. Based on the transactions collected in subsection 4.3.2, the communication time accounts for 87.5% of the total time, since most contracts' computation is simple.

Design. To minimize the communication overhead, we propose an overlap cooperation mode that overlaps the computation process and communication process during pre-execution. For example, as shown in Figure 4.5(b), after meeting the first cross-shard contract call in Cxn. 1, Node N_1 can transmit a cross-shard message to Node N_2 while simultaneously executing the computation task of Cxn. 2. Through this way, the computing resource and communication resource could achieve nearly full utilization. Furthermore, since all transactions are pre-executed based on the state of the previous block, we also propose a parallel cooperation mode. In particular, each node executes different transactions at the same time using redundant computation resources. For example, as shown in Figure 4.5(c), Node N_1 executes Cxn. 1 and 2 using two threads, i.e., Thread 1 and 2, respectively. As evaluated in subsection 4.6.2, the parallel scheme can increase the pre-execution throughput by 86.9% $\sim$ 280.0%

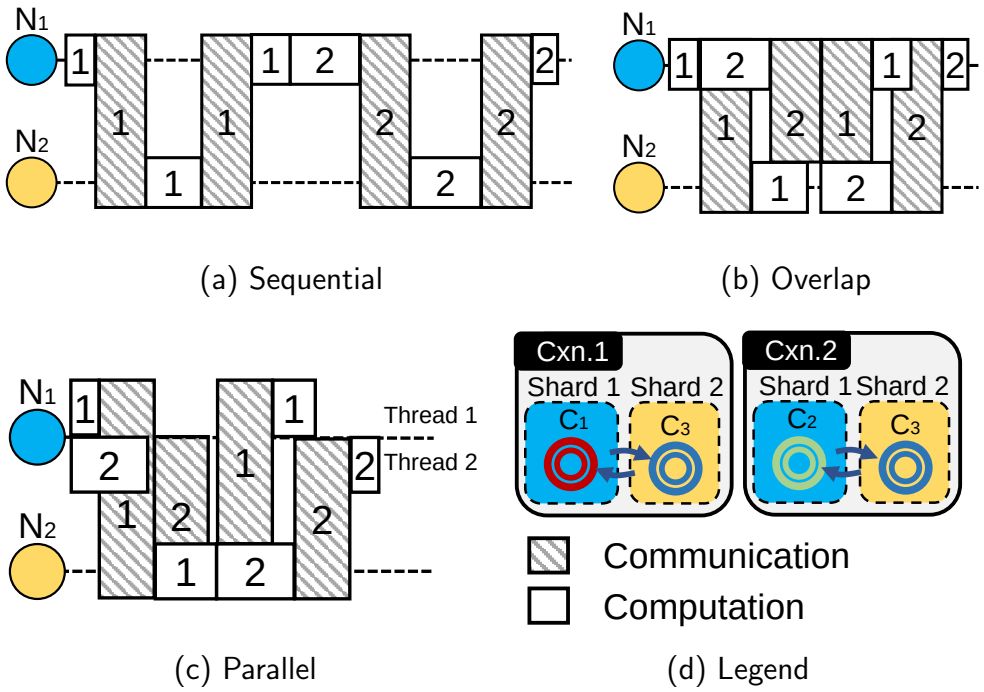


Figure 4.5: Comparison of three cooperation modes for a reconnaissance coalition in the pre-execution phase. The number inside each rectangle denotes the transaction ID to which the computation or communication time belongs.
compared with the sequential scheme.

4.4 Analysis

We first show how PROPHET achieves both determinism and serializability. The former one means that the same result is always produced in all honest node for each shard. The later one requires transactions in the system to produce the results following some serial order. The analysis depends on the intra-shard consensus of shards in PROPHET thus we define v as the fault threshold of the adopted intra-shard consensus [58]. For example, the synchronous protocol in Rapidchain [89] tolerate up to $v = 1/2$ Byzantine faults, while the asynchronous or partially synchronous protocol in Omniledger [46] tolerates only up to $v = 1/3$ Byzantine faults.

Theorem 4. PROPHET *achieves determinism and serializability if there are no more than v fraction of malicious nodes in each shard.*

Proof. When there are no more than $v < \frac{1}{3}$ malicious nodes in each shard, the intra-shard consensus can guarantee safety [89, 16, 38], i.e., the honest nodes in each shard agree on the same valid block in each round. Thus, the intra-shard consensus can guarantee that both the order proposed by the sequence shard follows the rule in subsection 4.2.4 and subsection 4.3.2 and the validation proofs proposed by the shards are valid. It also guarantees that a message along with a collective signature is honest because malicious nodes are the minority, i.e., no more than v, of the shard. Moreover, the message of each party (e.g., transaction profile, order, and proof) cannot be modified and forged since the collective signature can be used to detect forgery or tampering. Finally, because the correction phase guarantees that any invalid transaction will not be committed in all its related shards, all the honest nodes in the sharding system can run an identical batch of transactions based on the same global serial order and the same blockchain state. Additionally, the code of smart contracts is deterministic [87], which means each node can get the same result given the same input for a contract method. It guarantees the determinism of the consensus in PROPHET.

Next, we prove the serializability by contradiction as follows. Assume the global order produced by the sequence shard is: $\cdots \rightarrow T_i \rightarrow \cdots \rightarrow T_j \rightarrow \cdots$ where T_i and T_j can be two transactions in the same shard or in the different shards. There are two possible outcomes to violate the serializability. The first one is that T_j's update is overwritten by T_i's update. The second one is that T_i reads T_j's update. However, for subsection 4.2.4, T_i and T_j will not access the same contract. And, for subsection 4.3.2, the sequence shard only allows read-after-read and write-after-read dependency, thus both outcomes result in a contradiction and the consensus in PROPHET achieves serializability. In addition, we emphasize that even if T_i is invalidated in the correction phase, the following transactions in the global order will

not be influenced. It is because T_i is not allowed to change the state of contracts that are read or written by its following transactions considering the read-after-write is not allowed. □

Similar to the other sharding systems [54, 89, 46, 38], PROPHET has a global fault threshold for the whole sharding system denoted by f and a security parameter denoted by λ. After dividing each node to a random shard, the proportion of malicious nodes in each shard for PROPHET can be proven to be lower than the fault threshold v with low probability, i.e., the probability is no more than $2^{-\lambda}$, thus Theorem 4 can be guaranteed with high probability in PROPHET.

Overhead Analysis. In terms of the time overhead, PROPHET parallelizes the sequence phase and execution phase in subsection 4.3.1; thus, there is one consensus in every shard for each round, similar to the existing blockchain sharding. Besides, asynchronous correction in subsection 4.3.3 enables each shard to move to the next round without waiting for the other shards' proof after the execution phase, saving the time of the correction phase. Therefore, only the pre-execution phase introduces an additional time overhead for each round. In terms of the computation overhead, the pre-execution phase introduces some additional computation tasks to reconnaissance coalitions.

4.5 Implementation

We implement a prototype of PROPHET based on Geth [23], the Go language implementation of Ethereum. The smart contracts in PROPHET run in EVM in Geth. We adopt a BFT consensus with BLS multi-signature [33] as the intra-shard consensus of PROPHET. For comparison, we also implement two non-deterministic sharding prototypes. Since the intra-shard consensus in PROPHET can be substituted by any other BFT consensus, to ensure the result will not be affected by the difference in intra-

shard consensus, we adopt the same consensus for the intra-shard consensus in these two non-deterministic sharding prototypes. Moreover, for a fair comparison, both prototypes are equipped with the fine-grained read/write approach in subsection 4.3.2. The difference between two prototypes is the cross-shard transaction processing. The first one uses the OCC mechanism in Monoxide [83] and the second one uses the 2PL mechanism in Chainspace [2].

4.6 Evaluation

Dataset. To evaluate our sharding system PROPHET on the historical transactions in Ethereum, we implement a smart contract recorder/replayer based on EVM stateless state transition tool [22] similar to [44] and collect the blocks from Nov-25-2019 to May-04-2020 (block height: 9,000,000-10,000,000) from Ethereum mainnet blockchain.

Setup. The number of nodes in each shard is set as 50. In OCC and 2PL, the maximum retry count for the transactions is set as 10, which means that a transaction with more than 10 retries will be aborted. The testbed is composed of 16 machines, each of which has an Intel E5-2680V4 CPU and 64 GB of RAM, and a 10 Gbps network link. Similar to [89, 46], to simulate geographically-distributed nodes, we set the bandwidth of all connections between nodes to 20 Mbps and impose a latency of 100 ms on the links in our testbed. The proportion of malicious nodes in the system is set as 12.5%. In our setting, the malicious nodes in each reconnaissance coalition provide invalid cross-shard messages and read/write sets to interrupt the pre-execution phase. We repeat each experiment three times and compute the average as its result.

Metrics. We measure the performance of a sharding system using the following metrics. 1) *Transaction throughput*: the throughput of the confirmed transactions

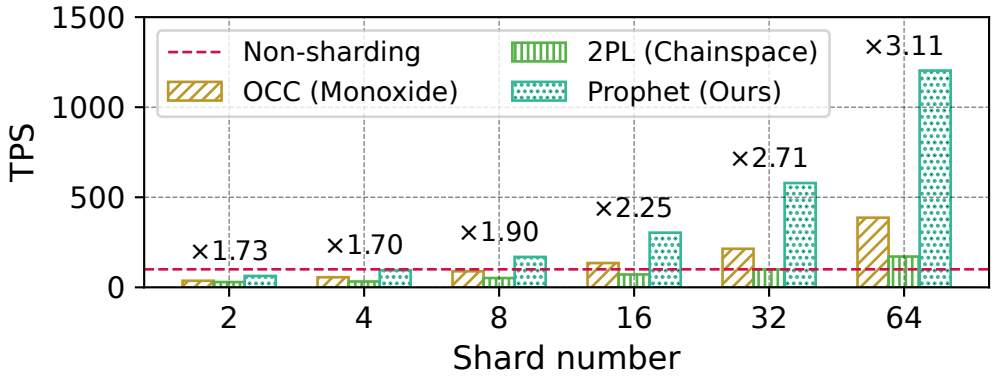


Figure 4.6: Transaction throughput of PROPHET and the existing sharding systems (The number above each bar denotes the ratio of the throughput of PROPHET over that of OCC.)

measured in TPS. 2) *Confirmation latency*: the delay between the time that a transaction is issued by a client until it can be confirmed by any (honest) node in the system. 3) *Abort ratio*: the ratio of aborted transactions during commitment, i.e., the transaction whose retry count exceeds the maximum retry count as discussed above. 4) *Invalid ratio*: the ratio of invalid transactions found in the correction phase for PROPHET.

4.6.1 Performance

To evaluate the performance, we measure the throughput in TPS for the two non-deterministic sharding blockchains and PROPHET with varying number of shards. As shown in Figure 4.6, all three sharding systems achieve the linear scalability. However, PROPHET improves the throughput 1.73 $\sim$ 3.11X against the two traditional sharding systems and the improvement is more significant when there are more shards. Moreover, the throughput of the sharding systems is even worse than that of non-sharding system when there are less shards, the reason of which is twofold. First, introducing sharding into the blockchain results in cross-shard transactions, each of which

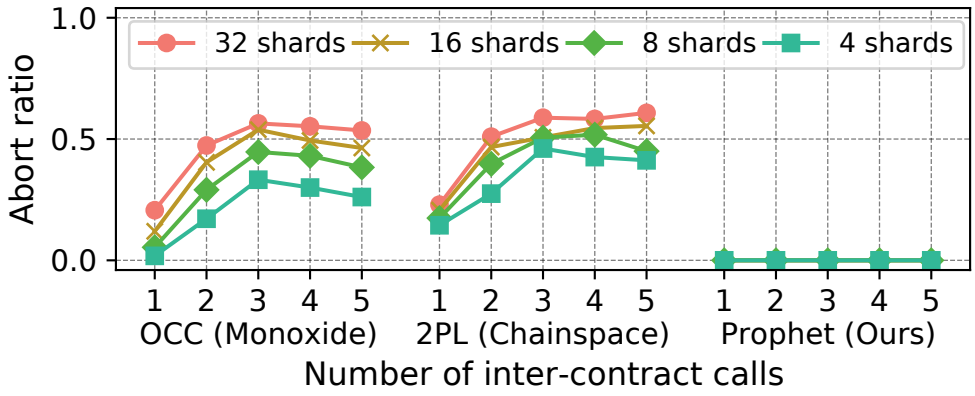


Figure 4.7: Abort ratio of transactions during commitment in Prophet and the existing sharding works.

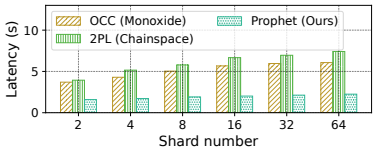


Figure 4.8: Confirmation latency of Prophet and the other blockchain sharding systems.

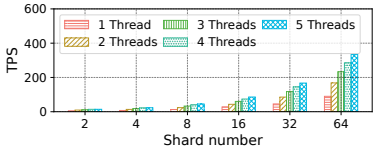


Figure 4.9: Pre-execution throughput of a reconnaissance coalition for Prophet with different number of shards.

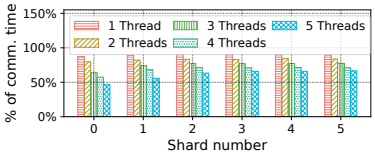


Figure 4.10: Percentage of communication time in the pre-execution in Prophet.

needs to be divided into multiple sub-transactions and processed in multiple consensus rounds. Second, the contention of cross-shard transactions results in frequent aborts of transactions thus most of the throughput in the two traditional sharding systems is wasted.

To investigate the wasted throughput reason as described above, we measure the abort ratio of transactions with varying number of inter-contract calls for the two non-deterministic sharding blockchains and Prophet. Note that although there are many transactions with more than 4 inter-contract calls in the system, Figure 4.7 only shows the transactions with 1-5 inter-contract calls due to the space constraint. Figure 4.7 shows that the abort ratio in both of all sharding systems increases as the

number of shards increases. Specifically, the OCC-based system aborts about 50% transactions with more than 2 inter-contract calls. It limits the dApps consisting of complex smart contract interactions. In comparison, PROPHET keeps the abort ratio being 0 no matter how many cross-shard contract calls the transactions include.

We then evaluate the confirmation latency of transactions in PROPHET and the non-deterministic sharding blockchains with varying number of shards. Figure 4.8 shows that the latency increases as the number of shards increases in the non-deterministic systems. This is because the increase of shards can introduce more cross-shard transactions that need more consensus round to be committed, thus the confirmation latency is higher. In comparison, the latency in PROPHET is low and relatively stable since any cross-shard transaction can be committed by PROPHET in one round.

4.6.2 Micro-benchmark

To analyze the effectiveness of our communication efficient pre-execution proposed in subsection 4.3.4, we evaluate the pre-execution throughput of a reconnaissance coalition with varying number of shards. As shown in Figure 4.9, the pre-execution throughput increases with the number of threads in each node. When there are more shards in PROPHET, the number of nodes in a reconnaissance coalition increases and each node has a thread. This parallelism improvement outweighs the increase of cross-shard transactions. Furthermore, a reconnaissance coalition achieves 334 TPS with 5 threads. Based on combining this observation and Figure 4.4(a), we can get that when there are more than 10 reconnaissance coalitions, the pre-execution throughput can exceed the maximum throughput (i.e., 1203 TPS in Figure 4.6) in PROPHET. Therefore, the bottleneck is not in the pre-execution phase.

To further investigate the improvement of communication efficient pre-execution, we evaluate the proportion of communication time in the total time in the pre-execution phase and the result is shown in Figure 4.10. Note that when the communication is

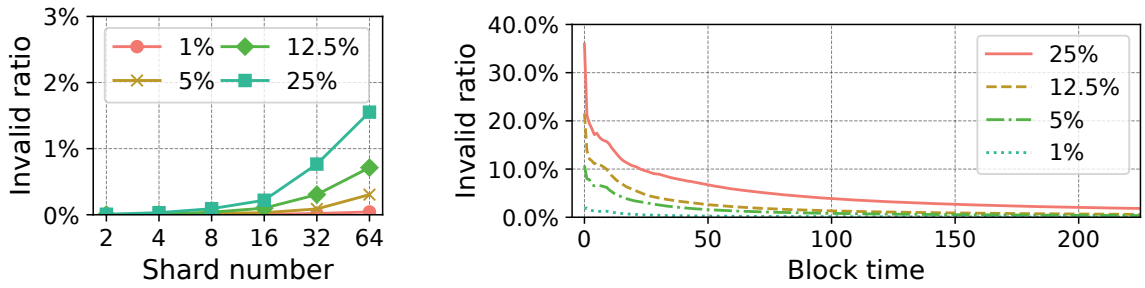
Table 4.1: The average total size of cross-shard messages for a transaction in a system with varying number of shards.

Shard number	2	4	8	16	32	64
Message size (Byte)	177	240	267	310	319	322

overlapped by the computation, we do not consider the communication time in the total time. From the figure, we can see that the proportion of communication time is less in a node with more threads.

We also evaluate the average total size of cross-shard messages for a transaction in a system with varying number of shards. Table 4.1 shows that the average total message size increases with the number of shards. It is because the number of shards can result in more cross-shard contract calls in a transaction. Moreover, the increment of message size gradually decreases. Specifically, doubling the shard number from 32 to 64 only increases the message size by 3 Bytes. It is because the number of cross-shard contract calls will be mainly influenced by the number of contract calls when the shard number is more than the number of contract calls.

Although malicious nodes cannot make the invalid transactions be confirmed because of the correction phase, they can occupy the throughput for the valid transactions. We evaluate the invalid ratio of Prophet with different percentage of malicious nodes (in particular 1%, 5%, 12.5%, and 25%) and the result is illustrated in Figure 4.11(a). The malicious nodes have only a limited impact (less than 2%) on the throughput of Prophet. Figure 4.11(b) shows that the invalid ratio decreases over time because of the gradual construction of honest coalitions. As discussed in subsection 4.2.3, to gain more transaction fees, the honest nodes tend to form and stay in reconnaissance coalitions involving the other honest nodes and leave coalitions involving malicious nodes.



(a) Average invalid ratio.

(b) Fluctuation of invalid ratio for 64 shards.

Figure 4.11: Ratio of invalid transactions in the correction phase.

Chapter 5

GriDB: Scaling Blockchain Database via Sharding and Off-Chain Cross-Shard Mechanism

Characterized by trustworthiness, transparency, and traceability, blockchain technologies have been integrated into many areas, such as cryptocurrency [61], supply chain [41], international trade [28], etc. In database management, blockchain technologies have attracted considerable interest in upgrading traditional databases to blockchain-empowered distributed databases [74], which forms an emerging research direction namely *blockchain databases*.

Compared with traditional distributed databases, blockchain databases transact and record data via blockchains and construct an abstract database layer supporting various query functionalities on top of blockchains, which endow the distributed databases with immutability and traceability [19, 64, 98, 65, 88, 91, 29]. For example, BlockchainDB provides shared tables as easy-to-use abstractions as well as a key/value interface to read/write data stored in the blockchain [19]. Pei *et al.* introduces a Merkle Semantic Trie-based index to support semantic query, range

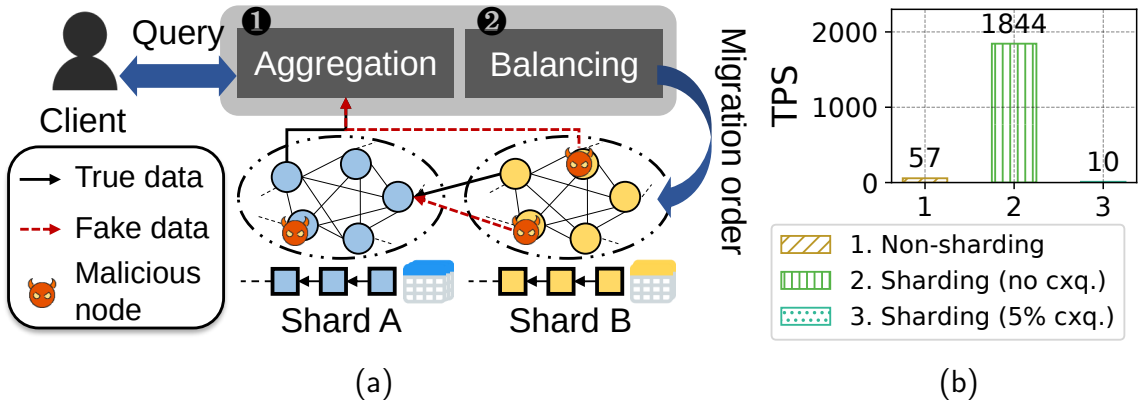


Figure 5.1: (a) Illustration for sharding blockchain database, which requires two new functions, i.e., data aggregation for query and workload balancing for management. (b) Transaction throughput of non-sharding and sharding blockchain databases. (cxq. represents cross-shard query.)

query and fuzzy query on the blockchain [64]. SEBDB adds relational data semantics into blockchain storage and thus supports SQL query [98] and FalconDB presents a blockchain database with SQL query with time window [65].

Due to the underlying non-scale-out blockchains, most existing blockchain databases suffer from poor scalability. For example, schemes in [98, 65] adopt Tendermint which achieves throughput of about 1000 transactions per second (TPS) but its network scale is less than 100. Schemes in [64, 91] adopt Ethereum aiming to support thousands of participants but only have tens of TPS. The poor scalability makes the blockchain databases hardly meet the quality of service required in large-scale business in practice.

Sharding is one of the most promising technologies for the blockchain scalability [54, 46, 89, 83, 16, 66, 35]. It divides the nodes into small groups called *shards*, which can handle transactions in parallel and alleviate the storage overhead for each node. In such an approach, the transaction throughput scales linearly with the number of nodes. To develop a scalable blockchain database, this paper is going to construct

an abstract database layer on a sharding blockchain by distributing database data and the corresponding task of storing, querying, or updating to different blockchain shards. However, such a sharding for database storage and workload introduces a new requirement namely *cross-shard database services*, i.e., *data aggregation* for query and *workload balancing* for management.

As shown in Figure 5.1(a), the data aggregation is caused by the sharding for storage. Particularly, the data related to a query may be stored by the blockchain nodes from multiple shards; thus, the query requires the involvement of several shards. For example, if there are two tables stored in Shard A and B, respectively, then a SQL `JOIN` query combining these two tables involves both shards. Moreover, the workload balancing is caused by the sharding for workload. Particularly, due to the sharding, each shard is only responsible for the workload of query and update to its storage. The demand imbalanced and dynamic nature of applications results in workload imbalance among shards, which significantly degrades the performance of sharding blockchain database.

The Byzantine environment of blockchain databases makes the technologies of traditional distributed databases no longer applicable. Malicious nodes may collude with each other and violate the protocol in arbitrary manners. For example, in distributed databases [17], a query can be easily realized by requesting from one database node in every related shard. However, in a sharding blockchain database, the correctness, completeness, and freshness of cross-shard queries can hardly be guaranteed when accidentally requesting from a malicious node. Moreover, elastic workload balancing is the first-class feature in modern databases [77] achieved by load migration. However, unlike one-to-one crash-tolerant migration in most distributed databases, the sharding blockchain database requires a many-to-many migration across shards in which malicious nodes can intercept, tamper or forge the migrating table.

To resist the Byzantine faults for cross-shard database services, an intuitive idea is to process them through the cross-shard mechanism of blockchain sharding. In detail,

the core of cross-shard database services is to transfer tables among shards despite Byzantine failures. The cross-shard mechanism guarantees that each data transfer (e.g., money transfer in the conventional blockchain) is agreed by the majority of honest nodes in all its related shards. Transferring a table among shards through the mechanism can guarantee that the table transfer will not be compromised (detailed in section 2.2 and section 4.1). However, such an idea is costly. On the one hand, each query or migration involves a massive set of semantics-related data (e.g., rows belonging to a table) in a blockchain database. On the other hand, all existing cross-shard mechanisms are on-chain (i.e., requiring the consensus of all the related shards). Therefore, all nodes in the related shards need to participate in the consensus on numerous data. As proved in Figure 5.1(b), the existing on-chain cross-shard mechanisms cannot support even 5% cross-shard queries in a sharding blockchain database with 32 shards (detailed in section 3.4).

To this end, this paper focuses on relational blockchain database and proposes the first relational sharding blockchain database, named GriDB. In comparison with the previous blockchain databases, GriDB guarantees high scalability while providing support for relational database services in blockchain sharding. Motivated by the idea of off-chain payments and verifiable computing, GriDB enables an off-chain execution of the cross-shard database services by adopting *authentication data structure* (ADS) to delegate cross-shard communication-intensive tasks to a few nodes in a verifiable manner. We summarize our contributions as follows.

- GriDB introduces relational data semantics and query functionality into blockchain transactions to abstract a sharding blockchain as a distributed relational database. The clients can send requests to any untrusted blockchain nodes for storing, manipulating and retrieving data.

- To provide a query layer of abstraction on sharded data, we design a cooperative delegation-based approach with a constant complexity of data transfer among

shards without sacrificing security. It delegates the tasks of data aggregation to a few nodes in different shards and constructs a succinct proof used to on-chain verify.

- To meet the dynamically skewed workloads and achieve inter-shard balancing, we propose an off-chain live migration that migrates the database service among shards with security, low cost, and minimum interruption.
- We develop a prototype for GriDB and conduct a comprehensive evaluation. The result shows that GriDB achieves a scalable throughput for SQL linearly increasing with the shard number compared with the non-sharding works.

5.1 System Model

System Components. In GriDB, there are two types of entities:

1) The *database clients* are the service consumers of GriDB. They neither participate in the consensus nor store the whole content of blockchains locally because they are often lightweight devices.

2) The *blockchain nodes* are the consensus participants for the blockchain and are divided into a number of shards. Each node is responsible for verifying, processing and storing transactions of its located shard. GriDB is a layer-2 database framework constructed on top of existing blockchain sharding systems, and it is *sharding-agnostic*, which means the underlying system can adopt any sharding schemes (including shard formation, intra-shard consensus and cross-shard mechanism) from [46, 89, 83, 38]. However, the underlying blockchain sharding system's intra-shard consensus should satisfy both safety and liveness, and its cross-shard mechanism guarantees the ACID of each cross-shard transaction (see section 2.2). We emphasize that the cross-shard mechanism of the underlying blockchain sharding system is one of the important components of the off-chain cross-shard mechanism of GriDB, which will be described in

the following sections. To avoid confusion, we will call the former *on-chain cross-shard mechanism*.

GriDB considers an outsourced database scenario [90, 59] in which the clients outsource their data management to the blockchain. The nodes host the client's databases and the clients send requests to the nodes to create, store, update and query their databases.

Threat Model. The threat model of GriDB relies on that of the underlying blockchain sharding composed of two kinds of blockchain nodes: *honest* and *malicious*. The honest nodes abide by all protocols in GriDB while malicious nodes may collude with each other and violate the protocols in arbitrary manners, such as denial of service, or tampering, forgery and interception of messages. Although there are malicious nodes in the shards, the sharding blockchains [89, 46, 83] can guarantee that each shard is trusted with high probability, i.e., the result published by any shards is trusted. Different from [19], GriDB does not require a strong assumption that each client trusts the nodes it connect.

Transaction Model. The requests of clients are processed in the form of blockchain transactions that are divided into two types: *data* and *query transaction*. The first one is used to update (such as insert, update, and delete) the database state and the second one is used to query the database state. Besides, in subsection 5.3.3, there are some *control transactions* used to support the database management such as database migration. The type division will not affect the compatibility for the underlying blockchain because there is a "data" field in the transactions of most blockchains, and GriDB places different data in the field for different types of transactions. The details of transactions will be described as follows.

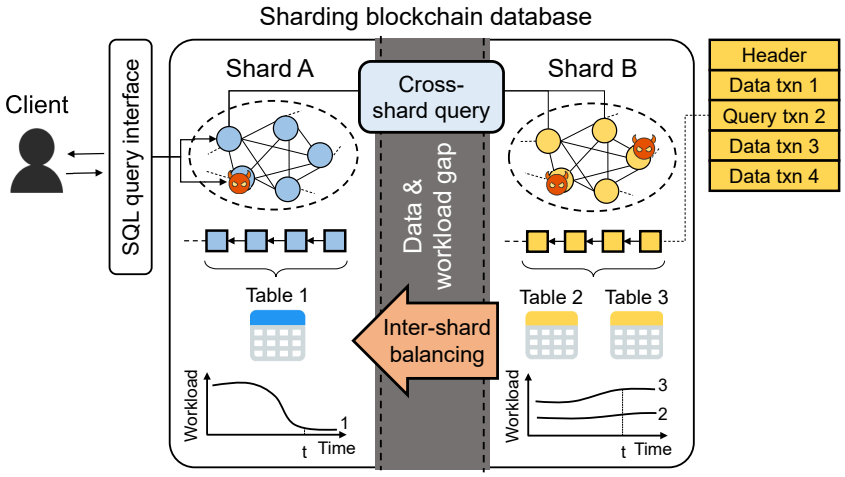


Figure 5.2: System overview for GriDB.

5.2 GriDB Overview

5.2.1 System Overview

As shown in Figure 5.2, GriDB considers a distributed relational database storing a number of tables. For scalability, the workload of each table is distributed to a shard (A fine-grained sharding for blockchain database through table partition will be described in subsection 5.4.4.) If a client decides to query or update a table, it can issue requests to any nodes in the shard responsible for the table. As described in the threat model, the nodes receiving the requests may be malicious, thus they are required to return a proof used for authentication. To generate a proof, based on the request received, a node first proposes a transaction which can be one of the following.

1) A data transaction is used to manipulate (such as insert, update, and delete) the data and includes an `INSERT`, `DELETE` or `UPDATE` SQL statement. In GriDB, the data is in the form of a relational model. The same type of data has a unified semantic description as a schema composed of several attributes. An insert statement inserts new data based on explicitly specified values or from the existing data via a nested sub-

query. Since blockchain is append-only, a delete statement is implemented by marking old data as invalid, i.e., cannot be queried, and an update one is implemented by a sequence of delete and insert operations to overwrite. Based on the data transactions recorded in the blockchain, each node in a shard maintains a tamper-proof copy of a relational database.

2) A query transaction is used to record query results to clients. A query statement begins with a keyword `SELECT` followed by a subset of column names, and then a keyword `FROM` followed by a table (or a `JOIN` sub-clause used to combine multiple tables in a later section.) Following these, a `WHERE` clause is followed by a sequence of predicates connected by logical operators (e.g., `AND`, `OR`, `NOT`) that restrict the rows used when computing the output. After processing the request, the node can put the query statement and result into the data field of a query transaction. To avoid occupying too much on-chain storage, the query results can be offloaded to an off-chain storage and the query transactions only store the hash of query results, according to which the clients can download the correct results from the off-chain storage based one the hash.

Next, the node broadcasts the proposed transaction to the network. If a data transaction is committed to the blockchain, the majority of honest nodes accept and execute the data transaction, which means the database state has been successfully updated. If a query transaction is committed, the majority of honest nodes agree on the query results. Finally, the client can authenticate the result returned by its connected node by validating if the transaction for its request is committed. This transaction validation for clients has been implemented in most blockchains, such as Simplified Payment Verification (SPV) in Bitcoin and Ethereum.

In addition to a Merkle tree storing transactions like traditional blockchains, to enable every node or client to know every table's location, every node maintains an additional Merkle tree. The tree stores the location of all tables in the form of *table name-shard id* pairs. It is updated in each epoch according to a global cross-shard control transaction

including a new planning strategy for the following epoch (refer to subsection 5.4.2). Thus, depending on the tree, every node or client can find the correct shard storing the target table.

5.2.2 Challenges

Dividing the tables across different shards improves the blockchain database's scalability. However, it is not enough for a scalable blockchain database due to two following problems.

Problem 1 (cross-shard query): A client's query involving tables only in a single shard can be served by one shard because each node of the shard can validate and agree on the query result in an intra-shard consensus. However, a request to query the tables from different shards cannot be completed by a single shard and requires the cooperation of multiple shards. For example, in Figure 5.2, a query joining on Table 1 and 2 involves the data of Shard A and B thus cannot be completed by only one of them.

Problem 2 (inter-shard workload imbalance): It is hard to guarantee that the workload of every table is the same and static, thus some shards can be overloaded while some others remain idle. For example, in Figure 5.2, Shard A is responsible for Table 1 and Shard B is responsible for Table 2 and 3. At the beginning, the total workload in Shard A and B is similar. However, if the workload of Table 1 drops over time, Shard A becomes idle. To fully utilize the throughput, dynamically migrating the workload among shards and alleviating the effect of hotspots are crucial.

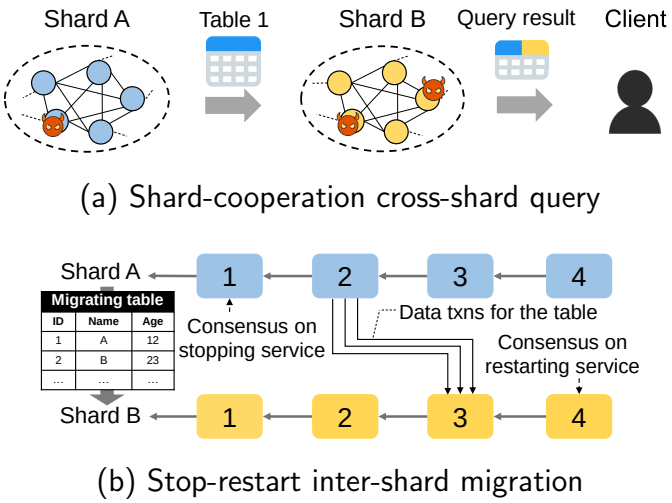


Figure 5.3: Overview for the on-chain strawman.

5.3 System Design

In this section, to outline GriDB's design, we first describe a strawman sharding blockchain database based one the on-chain solution for the challenges discussed in subsection 5.2.2. Next, to address the drawbacks in the strawman system, we discuss the primary cause of the drawbacks and introduce the key designs in subsection 5.3.2 and subsection 5.3.3.

5.3.1 Strawman

For the two challenges above, we first describe a strawman sharding blockchain database only based on the on-chain cross-shard mechanism of the existing sharding systems as follows.

Consider a cross-shard query involving two tables, i.e., Table 1 and 2, located in Shard A and B, respectively. For such a cross-shard query, we present a *shard-cooperation approach* based on the on-chain cross-shard mechanism. Shard A first commits many cross-shard data transactions involving Shard A and B, including the data of Table 1 via the cross-shard mechanism. As described in section 2.2, the mechanism guarantees

the transactions can be committed in Shard A and B. Then, Shard B can get Table 1 from the transactions, compute the query result, and commit a query transaction with the query result. However, when there are many cross-shard queries, the table transfer among shards will be frequent, resulting in system overloaded and network blocked.

For the inter-shard load balancing, the latest version of the table should be transferred from the source shard to the destination shard. If there is data being left out, the completeness of queries on the table cannot be guaranteed after migration. Thus, we present a *stop-restart migration approach* based on the on-chain cross-shard mechanism as follows. Shard A first stops processing new transactions for the table via an intra-shard consensus, which avoids the migrating table being modified during migration. Then, Shard A commits many cross-shard data transactions to reconstruct the latest version of the table in Shard B. After all transactions are committed, Shard A commits a cross-shard transaction to mark the end of the migration, and Shard B can restart the service of the table. However, when the migrating table is enormous, the approach incurs a high penalty due to the prolonged service interruption for the migrating table and the influence on other tables' throughput.

To solve these drawbacks, we introduce two key designs for our off-chain cross-shard mechanism in subsection 5.3.2 and subsection 5.3.3.

5.3.2 Cross-Shard Query Authentication

Motivation. Although the above shard-cooperation approach is safe, it is expensive since the table transfer among shards for each cross-shard query is between groups of nodes (to guarantee that there is a majority of honest nodes for each shard participating). An intuitive idea to avoid this overhead is to pick one delegate from each shard. Then, for each cross-shard query, a delegate downloads the related tables from the other delegates and evaluates the query results. However, any malicious delegate

can easily tamper with the query result by providing a fake or out-of-date table.

We aim to design an ADS to allow the delegates to prove the validity of cross-shard query results. The existing outsourced databases have designed some ADS for SQL [90, 59]. For example, for a `JOIN` query involving the same column of two tables, a node treats the columns of these two tables as two sets and constructs a proof for their intersection through VSO [95]. However, the existing ADS for SQL cannot be applied in our cross-shard query due to two difficulties. First, different from the outsourced database in which there is no sharding and a server stores the whole data copy, any delegate in GriDB only stores the tables of its located shard and downloads tables from the other untrusted delegates and thus cannot construct a valid proof by itself. Second, to support arbitrary verifiable SQL queries, besides VSO, the other outsourced databases need to adopt interval trees [95] or zero-knowledge proof [94], which costs tens of minutes for a query [94] due to high computation complexity.

Thus, we propose a *delegation-based approach* by integrating VSO with the intra-shard consensus to implement an efficient and secure ADS for arbitrary SQL query in GriDB. Its main idea is to divide each query into some algebra operators with different input data. Particularly, it validates the operators involving multiple shards' data through VSO and those involving single shard's data through the intra-shard consensus. The cross-shard query validity can finally be proved through a chain of trust, i.e., proving the validity of every operator from beginning to end. Such a manner makes the best use of the advantage of both the shard-cooperation approach (i.e., low computation) and the existing ADS for verifiable SQL (i.e., low communication) and bypasses their disadvantage.

Design. The overall cross-shard query procedure is given in algorithm 1. For each cross-shard query, we identify the related shard of the table following `FROM` as *main shard* and the shards of the tables following `JOIN` as *sub shards*. A client can issue a cross-shard query request to any nodes in the main shard. Next, one node is chosen from each related shard (Line 2), which can be round-robin or randomly by a verifiable

random function [30]. The malicious or low-response delegates can be replaced by a view change similar to PBFT. The delegated node in the main shard is called the *main node* denoted by $\mathbb{M}$ and those in the sub shards are called *sub nodes* denoted by $\mathbb{S}$. The main node downloads each involved table for the query from the sub node in the corresponding sub shards (Line 3). After downloading all involved tables, the main node evaluates the query result and generates a proof (Lines 4, 8-14).

To generate the proof, the main node first translates each SQL query into a relational algebra tree composed of algebra operators [70], e.g., the right part of Figure 5.4. Each node in the tree denotes a unary (or binary) algebra operator taking one (or two) inputs, applying a function, and outputting its result to the next operator. The edges represent data flow from bottom to top.

In the tree, we identify join (or union) operators involving tables in different shards as *cross-shard operators* and the others as *intra-shard operators*. Each intra-shard operator can be processed by the nodes of the corresponding shard based on their stored tables. In comparison, each cross-shard operator involves the data gap among shards, thus the main node needs to generate a proof. The proof is composed of the accumulation values of the corresponding columns in the tables to be joined or unioned, a VSO proof, and a position indicator for the intermediate result (or final result). The position indicator is a bitmap to indicate which rows are chosen in a table. For example, considering the SQL statement in Figure 5.4, we denote the oid columns of these two tables after processing the selection operators as C^i and C^j, respectively. The generated proof Υ is $\langle acc(C^i), acc(C^j), \pi, [1,0,1], [1,0,0] \rangle$. The position indicators $[1,0,1]$ and $[1,0,0]$ mean that the first and third rows in Table 1 and the first row in Table 2 are chosen, respectively. A cross-shard query may include multiple cross-shard operators thus the main node will produce a list of proofs Υ, each of which is for a cross-shard operator.

After the query result and the corresponding proof are generated, the main node proposes a cross-shard query transaction, including the result and the proofs and

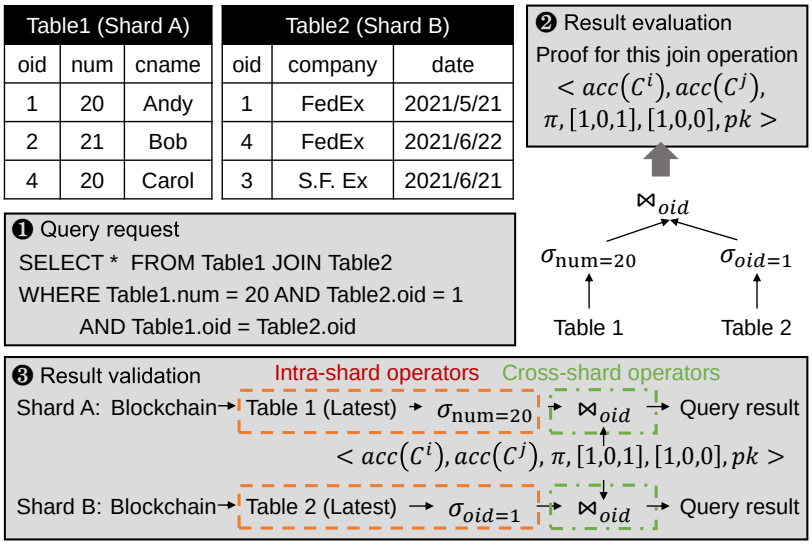


Figure 5.4: Example for ADS proof generation of two tables distributed in Shard A and B, respectively. (σ is an operator to select rows from a relation and $\bowtie$ is an operator to join tables based on a specified column.)

involving the related shards (Line 5). Then, to validate the cross-shard query, each related shard runs an intra-shard consensus on the transaction by evaluating each algebra operator for their stored tables and verifying the proofs related to the tables of the shard (Lines 15-20). For example, as shown in Figure 5.4-❸, during the consensus on the cross-shard query transaction, each node can validate and execute intra-shard operators based on the local data and validate and execute cross-shard operators based on the VSO proof. During the validation, they can optimistically assume that the accumulation values related to the other shards' tables are valid. Finally, if the cross-shard query transaction passes the validation of every related shard (Line 6), it will be committed in the blockchains of all related shards and the client can accept the query result included in the transaction via SPV (Line 7). Besides, if a malicious main node sends different copies of a transaction to shards, the client can detect the inconsistency by checking the Merkle proofs of the transaction via SPV (Line 7).

Security Analysis. The analysis relies on the intra-shard consensus of blockchain

sharding thus we define v as the fault threshold [58] of the adopted blockchain sharding in GriDB. For example, Rapidchain [89] tolerates up to $v = 1/2$ Byzantine faults, while the asynchronous or Omniledger [46] tolerates only up to $v = 1/3$ Byzantine faults. Next, we describe the formal definition [94, 95] of our cross-shard query's security as follows.

Definition 3. *A query is secure if any polynomial-time adversary's success probability is negligible in the following experiment:*

*For a query q, the adversary is picked as main node or sub node for the generation of query transaction including result R. The adversary succeeds if the query transaction is committed in all related shards and one of the following results is true: 1) R includes a row which does not satisfy q (**correctness**); 2) There exist a row which is not in R but satisfies q (**completeness**); 3) R includes a row not from the latest tables generated by all the committed data transactions (**freshness**).*

Theorem 5. *Our proposed cross-shard query mechanism satisfies the security property as defined in Definition 1 if the proportion of malicious nodes in each shard is no more than the fault threshold v.*

Proof. We prove Theorem 1 in three cases, corresponding to how GriDB defends against the three different adversaries in Definition 1 for each cross-shard query in correctness, completeness, and freshness. We first describe the three cases: Case 1: This case means a tampered or fake row within the result is returned, which does not satisfy the query q. In this case, the tampered or fake row can pass the client's verification under the correctness in Definition 1. Case 2: This case means a row that satisfies q is missing from R. In this case, the incomplete result can pass the verification of the client under the completeness in Definition 1. Case 3: This case means the result R involves an old row that satisfies q but is not from the latest tables. In this case, the old result can pass the client's verification under the freshness in Definition 1.

If any of the above three cases occur, it means the computation of at least one relational operator (intra-shard or cross-shard operator) for a committed query is invalid, i.e., the malicious nodes in a related shard tamper with the executing of intra-shard operators during intra-shard consensus, or the main node generates a wrong result in the executing of cross-shard operators. However, this contradicts two assumptions. The first one is that when the proportion of malicious nodes in each shard is no more than the fault threshold v, the safety of the intra-shard consensus holds [82]. Second, according to the unforgeability of VSO under the q-SDH assumption [63, 10], the ADS for set operations guarantees that the computation of each cross-shard operator in delegates is valid, and any invalid results can be detected by the intra-shard consensus. □

Performance Analysis. We analyze the time for a cross-shard query involving m cross-shard operators as follows. Three steps occupy most of the time, i.e., the table transfer (Line 3), the proof generation (Line 4), and the confirmation latency of the query transaction (Line 6), which is also proved in section 3.4. Thus, the analysis is developed around these three steps. For the table transfer, the time cost is linear to the size of the related tables. We introduce several refinements to reduce this time cost and improve query efficiency in subsection 5.4.1. Next, according to [63, 10], the proof generation time for each set operation involving N elements is $O(N \log^2 N \log \log N)$. Thus, the proof generation time is $O(mN \log^2 N \log \log N)$. Finally, the confirmation latency denotes the delay between the time that the query transaction is issued from the main node until the transaction is committed, which depends on the throughput, demand, and number of block confirmations of the blockchain.

5.3.3 Inter-Shard Load Balancing

Motivation. Observe that the drawback of the stop-restart approach in the strawman system results from the interruption for transaction processing during migration.

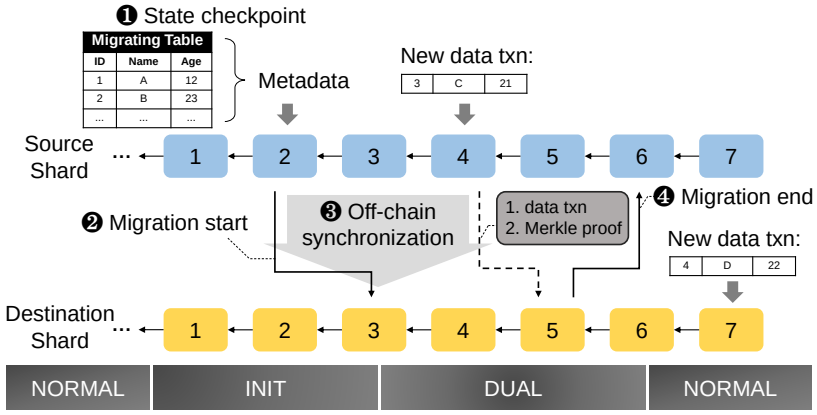


Figure 5.5: Overview of off-chain live migration. A solid line with arrowhead represents a cross-shard transaction and a dotted line with arrowhead represents an off-chain cross-shard communication.

Moreover, because the approach is on-chain, the migration occupies the transaction throughput of the shards involved, which interrupts the new transactions of the other tables. Thus, to avoid these drawbacks, we design an *off-chain live migration* approach for GriDB. Its main idea is to design an off-chain technique to minimize the number of on-chain transactions and a dual mode with cross-shard synchronization and concurrency control to minimize the impact of interruption to the migrating table during migration.

Design. Figure 5.5 illustrates the timeline of cross-shard migration and the messages exchanged between two shards. The life cycle of a table includes the following three modes.

1) Normal Mode: The normal mode for a table (called $\mathcal{T}$) is the period in which the data or query transactions of the table are processed normally by the shard it belongs to. The normal mode accounts for most of the time for the table.

2) Init Mode: When $\mathcal{T}$ is going to be migrated from the source shard (called $\mathcal{S}$) to the destination shard (called $\mathcal{D}$), the init mode starts. (We will discuss the trigger of table migration in subsection 5.4.2 which aims for load balancing and guarantees

that there is a super majority of honest nodes in $\mathcal{S}$ knowing $\mathcal{T}$ and $\mathcal{D}$.) The nodes in $\mathcal{S}$ first construct the metadata for $\mathcal{T}$ via a hash function such as SHA (Figure 5.5-❶) and commit a cross-shard control transaction involving $\mathcal{S}$ and $\mathcal{D}$ (Figure 5.5-❷). The transaction includes the metadata and a block number, representing a checkpoint for $\mathcal{T}$ in this block number. When the control transaction is committed in both shards by the on-chain cross-shard mechanism, the init mode ends.

3) Dual Mode: In the dual mode, $\mathcal{S}$ begins to transmit $\mathcal{T}$ to $\mathcal{D}$. The transmission among shards is pluggable and can be implemented by one-to-one communication or gossip mechanisms (Figure 5.5-❸). The nodes in $\mathcal{D}$ only accept the table matching the metadata in the control transaction. Because the download of the whole table may cost a lot of time, we adopt a pre-copy scheme in which the nodes in $\mathcal{D}$ can pre-download $\mathcal{T}$ from $\mathcal{S}$ in the normal or init mode and validate it after the commitment of the control transaction.

To keep the service for $\mathcal{T}$ during the dual mode, $\mathcal{S}$ continues to process the newcoming data and query transactions related to $\mathcal{T}$. The new data transactions in $\mathcal{S}$ may change the content of $\mathcal{T}$, thus $\mathcal{D}$ should be notified. It can be realized by committing all new data transactions as cross-shard transactions, however, which slows the service of $\mathcal{T}$ due to the overhead of cross-shard mechanism and blocks the throughput of $\mathcal{S}$ and $\mathcal{D}$ when the demand is high. Thus, we adopt an off-chain cross-shard notification mechanism based on Merkle tree as follows. First, in GriDB, similar to the other sharding systems [83, 38], each node will be a light node for the other shards and store the block headers of all shards. It does not hurt the scalability, since the light nodes do not need to participate in the consensus and each header occupies little storage space and bandwidth. The notification is in the form of an off-chain message including a data transaction and its Merkle proof. Any nodes in $\mathcal{S}$ can notify $\mathcal{D}$ via gossip mechanism [43]. Based on the notification received, $\mathcal{D}$ gets the latest data transactions for $\mathcal{T}$. For example, as shown in Figure 5.5, a new data transaction for the migrating table arrives in $\mathcal{S}$ and is committed at the 4-th block. Any honest node

in $\mathcal{S}$ can send the new transaction with its Merkle proof in the 4-th block to $\mathcal{D}$ for synchronization of $\mathcal{T}$ between $\mathcal{S}$ and $\mathcal{D}$.

After a node in $\mathcal{D}$ completes downloading, it proposes a cross-shard control transaction involving $\mathcal{D}$ and $\mathcal{S}$ or participates in the consensus on the one proposed by another node to show that it has downloaded the table successfully. Thus, the transaction can be committed if the majority of honest nodes in $\mathcal{D}$ confirm that they have downloaded the migrating table (Figure 5.5-❹). We assume that the off-chain notification arrives reliably and without latency here, which will be discussed later. Finally, the migration is completed and $\mathcal{D}$ has full ownership of the migrating table. It means that the later transactions (e.g., data/query transactions and migration requests) for the migrating table are processed by $\mathcal{D}$ only.

Asynchronous Issues. In the above, we ignore some problems resulting from the network latency or malicious nodes. Thus, we discuss them and provide some designs as follows.

Problem 1: In the init mode, due to the transaction latency existing in the blockchain, i.e., the delay between the time that a node sends a transaction to the network until the time that the transaction can be confirmed by all (honest) nodes, the metadata generated by different nodes may be in different versions. Thus, $\mathcal{S}$ may be unable to reach a consensus on the same control transaction. To synchronize the metadata among nodes in $\mathcal{S}$, GriDB sets a rule as follows. When a node begins to generate the metadata, it stops processing any new data transactions of $\mathcal{T}$ and disagree on blocks including these transactions during consensus until the init mode ends. Note that a node still accepts the newly committed blocks and updates its local database state and the corresponding metadata even if it disagrees them. Moreover, before a cross-shard control transaction is committed successfully, the nodes in $\mathcal{S}$ keep updating the metadata they generate based on the new block. If there is already the same control transaction proposed by other nodes waiting to be committed, the node can participate in its consensus.

Problem 2: In the dual mode, we adopt an off-chain notification mechanism to minimize the impact of interruption during migration. However, the off-chain communication among shards is not reliable thus the notifications may get lost. For the problem, we adopt the following designs. First, every new data transaction in the dual mode will be assigned an increasing sequence number before being committed in $\mathcal{S}$. Thus, if a node in $\mathcal{D}$ finds itself missing some transactions, it can directly request the corresponding notifications from the nodes in $\mathcal{S}$. Second, after the control transaction is committed (Figure 5.5-❹), $\mathcal{S}$ needs to commit a control transaction including the total number of new data transactions in the dual mode and sends the control transaction with a Merkle proof to $\mathcal{D}$. Besides, the nodes in $\mathcal{D}$ can actively ask the control transaction. Each node in $\mathcal{D}$ begins to process new transactions for $\mathcal{T}$ until it gets the total number of notifications and downloads all data transactions. Finally, $\mathcal{D}$ continues the service of $\mathcal{T}$ when the majority of honest nodes in $\mathcal{D}$ finish downloading.

Security Analysis. We first describe the formal definition of the security [21] for our off-chain live migration as follows.

Definition 4. *A migration is secure if achieving safety and liveness despite Byzantine failure. The safety requires serializable isolation, i.e., the migrating table's transactions run in serial order during migration, and durability, i.e., the committed transactions will not get lost after migration. The liveness indicates it eventually terminates.*

Theorem 6. *Our proposed off-chain live migration satisfies the security property as defined in Definition 2 if the proportion of malicious nodes in each shard is no more than the fault threshold v.*

Proof. During the migration, only one of $\mathcal{S}$ and $\mathcal{D}$ has full ownership and processes transactions for the migrating table. The intra-shard consensus guarantees that there is a serializable order for transactions among nodes in each shard, thus achieving serializable isolation. For durability, in the init mode, because the honest nodes

update their metadata before the control transaction is committed, $\mathcal{D}$ can download all data for $\mathcal{T}$ that are committed before the dual mode begins. The durability for transactions that are committed in the dual mode can be guaranteed by the final control transaction, including their total number in $\mathcal{S}$. Furthermore, the liveness can be guaranteed since the end conditions for each mode depend on the commitment of cross-shard transactions whose liveness has been shown to hold under the threat model of super-majority honest in each shard in any sharding works [82]. □

If the malicious nodes in $\mathcal{S}$ construct a wrong metadata for $\mathcal{T}$, the intra-shard consensus guarantees that the invalid transaction including the wrong metadata would not be committed. The nodes in $\mathcal{D}$ can detect wrong tables and wrong notifications according to the on-chain metadata and Merkle root, respectively.

Performance Analysis. The time for each migration is at least the latency of two cross-shard control transactions, one of which denotes the beginning of dual mode and the other the end. In parallel with the first one, $\mathcal{T}$ is transferred between shards. Its time depends on the adopted communication methods, network status, and network scale. After the second one, to guarantee that all data transactions are received by $\mathcal{D}$, there is a control transaction in $\mathcal{S}$ and the nodes in $\mathcal{D}$ need to download the data transactions that they miss. The time of the step also depends on the network environment. In the worst case, if all data transactions during migration are missed by the majority of honest nodes, the nodes may need some time to download the transactions they missed. Besides, the service of $\mathcal{T}$ may be halted for a time due to the first and third designs for the asynchronous issues during migration.

5.4 Design Refinement

5.4.1 Cross-shard Query Efficiency

Although the delegation-based approach in subsection 5.3.2 reduces the complexity for table transfer in a cross-shard query from $O(SN^2)$ (derived from the strawman in section 4.1) to $O(S)$ where N is the number of nodes in a shard and S is the number of related shards for the query, transferring a huge table from the sub nodes to the main node still costs a lot. However, many rows are useless in practice. For example, for query #5 in subsection 5.6.2 involving tables with millions of rows, the size of its final result only have single-digit items. GriDB optimizes the transferring of table among delegates as follows.

We first optimize by applying the unary operators and binary operators involving tables in the same shard early in the tree of each cross-shard query. Particularly, each sub node processes all selection operators related to its table and transfers the processed table to the main node. For example, in Figure 5.4, the selection operation $\sigma_{num} = 1$ is moved to the bottom of the tree and processed by the sub node in Shard A, reducing the size of Table 1 to be transferred to the main node in Shard B. Because the execution order of the tree is bottom-up, the main node in Shard B downloads the part of tables including these temporary outputs, based on which it can continue to process the next operation. Moreover, some projection operators also can be applied early similar to the selection operators.

Next, we adopt bloom filter (BF) for each cross-shard operator to filter out unnecessary data before transferring tables. BF [71] is a space-efficient probabilistic data structure used to test whether an element belongs to a set or not. In GriDB, before downloading the tables for a cross-shard operator, the main node can build a BF for the target column in its table and send the filter to the sub nodes. The sub nodes use it to filter their own table before transferring tables to the main node. Thus,

most useless data are filtered out before being transmitted, reducing communication overhead.

5.4.2 Load Balancing Scheduler

A critical problem to achieve inter-shard balancing is how to generate a good planning strategy to distribute the load to shards and how to apply the strategy in a distributed and safe manner in GriDB.

For a planning strategy, similar to distributed databases [20, 85], GriDB follows a widely-used greedy planning algorithm [77]. It iterates through the list of tables, starting with the one with the hottest demand. If the shard currently holding this table has a load exceeding the average demand, the algorithm migrates the table to the least load shard. The algorithm is easy to implement and has been proved to be efficient in many database scenarios [77, 20, 85].

To run the above algorithm in a decentralized manner, GriDB extends it by considering its execution in the existing sharding blockchains. Particularly, resharding phase is an important phase during the lifecycle of sharding blockchains [54, 46, 89]. A sharding blockchain proceeds in epochs, where each epoch consists of a resharding phase followed by multiple intra-consensus rounds. During the resharding phase of an epoch, a shard will be elected as the reference shard based on a round-robin rule. In GriDB, the reference shard can act as the load balancing scheduler in the resharding phase. The leader of every shard computes the demand of each table and reports it to the reference shard in a cross-shard control transaction via the cross-shard mechanism. After receiving the demand of every table, the leader of the reference shard proposes a cross-shard control transaction, involving all shards and including a new planning strategy in the following epoch, via the cross-shard mechanism. The cross-shard mechanism can guarantee that the new planning strategy is known by a majority of honest nodes in every shard. Based on this strategy, the tables can be

migrated among shards using the approach in subsection 5.3.3.

Security Discussion. The greedy planning algorithm and the table demand computing can be deterministic, thus any node can check the validity of their results. This guarantees that only the cross-shard control transactions, including valid table demand or valid planning strategy, can be committed in all the related shards and the invalid ones will be aborted via the cross-shard mechanism.

5.4.3 Cross-shard Insertion/Deletion/Update.

In GriDB, a data transaction can include a SQL statement for an insert, delete or update operation with nested subqueries or a multiple-table delete/update operation [60]. If the nested subquery is cross-shard in the first case or the related tables belong to multiple shards in the second case, the data transaction involves multiple shards.

For the first case, a data transaction including the cross-shard subquery result (or its hash) can be processed by the delegation-based approach in subsection 5.3.2 and committed as a cross-shard transaction. The transaction involves both the shard for the inserted/deleted/updated table and the related shards for the nested subquery. For the second case, a multi-table deletion/update can be considered as deleting/updating the specified rows in multiple tables based on a query to these related tables. Thus, it can also be processed as a cross-shard data transaction including query results similar to the first case. Finally, because the cross-shard mechanism guarantees the atomicity of cross-shard transactions (see section 2.2), the cross-shard insertion, deletion, and update take effect in all related shards. Any invalid cross-shard data transactions (e.g., including wrong subquery results) will be aborted by the cross-shard mechanism.

5.4.4 Horizontal/Vertical Table Partition

For each table, besides storing the entire table in one shard as discussed in subsection 5.2.1, GriDB can be developed into a fine-grained sharding blockchain database through *horizontally* or *vertically* partitioning the table into partitions. The former allows the table to be partitioned into disjoint sets of rows and the latter disjoint sets of columns. Load balancing can benefit from this fine-grained sharding for blockchain database since the database workload can be more evenly distributed to the blockchain shards.

The partitions of each table are distributed to different shards, thus a table is stored in multiple shards. For a horizontally partitioned table, each query needs to commit the same query transaction to all the related shards of the query. For a vertically partitioned table, each of its partitions can be regarded as an individual table. If a query involves the columns within a partition or the partitions related to the same shard, the query involves one shard and can be processed as a query transaction in the shard. However, if the query involves multiple columns of several partitions from different shards, it needs to be committed as a cross-shard query transaction.

5.5 Discussion

5.5.1 Permissioned and Permissionless Setting

GriDB can be applied in both permissioned and permissionless scenarios, relying on the underlying blockchain sharding system. For a permissioned scenario, only a set of known, identified, but untrusted nodes can serve as blockchain nodes similar to the permissioned blockchain databases [19, 65]. For a permissionless scenario, the blockchain database is public and open, and anyone can become a blockchain node without a specific identity. To resist Sybil attacks caused by the permissionless

setting, GriDB can use a PoW-based identity generation as described in section 2.2, which is widely adopted by the permissionless blockchain sharding [46, 89]. Moreover, to compensate for the consensus overhead of blockchain nodes and avoid the Verifier Dilemma [55], GriDB will explicitly charge fee for each transaction and reward the blockchain nodes [73]. We leave an incentive mechanism design for GriDB as our future work.

5.5.2 General Join

The cross-shard query authentication in subsection 5.3.2 works for equality join, because the cryptography primitive adopted in GriDB supports set intersection only. For a general join case such as non-equijoin (i.e., join operation using comparison operator like >, <, >=, <= with conditions), we can resort to cryptographic technologies with more general verifiable computing capacity, e.g., Trusted Execution Environment (TEE) and Succinct Arguments of Knowledge (SNARK), which will be left as our future works.

5.6 Experimental Evaluation

Implementation. We implement a prototype of GriDB in Go [31] based on Ethereum [23] and Harmony [32]. We adopt a BFT consensus with BLS multi-signature [33] as the intra-shard consensus and a library named ate-pairing [37] for the VSO. The on-chain cross-shard mechanism of GriDB is similar to that of Monoxide [83]. Particularly, to commit a cross-shard transaction, each of its related shards needs to validate and commit it. Only if the transaction is committed in the blockchains of all its related shards, it is regarded as being committed successfully. This can be checked based on a list of Merkle proofs, each corresponding to a related shard. Besides, by checking the transaction hash included in every Merkle proof,

it can be guaranteed that every related shard commits the same transaction. The optimization designs in section 5.4 are also implemented. To implement a MySQL interface to GriDB, we adopt a storage-agnostic SQL engine with in-memory table implementation [18].

Setup. The testbed is composed of 16 machines, each of which has an Intel E5-2680V4 CPU and 64 GB of RAM, and a 10 Gbps network link. Similar to [89, 46], to simulate geographically-distributed nodes, we set the bandwidth of all connections between nodes to 20 Mbps and impose a latency of 100 ms on the links in our testbed.

Baseline. For comparison, we implement a non-sharding blockchain database. This type of blockchain database does not need to consider the challenge of cross-shard query and inter-shard balancing because each node stores and processes the whole database. For a fair comparison, this blockchain database also adopts the signature-based BFT consensus adopted by GriDB as its underlying consensus. The basic idea of the non-sharding blockchain database is similar to that of the existing works such as FalconDB and SEDBD [65, 98] except that they adopt the other variants of BFT consensus and support some other functionalities (such as indexes). Moreover, we implement an on-chain sharding blockchain database including shard-cooperation cross-shard query and stop-restart inter-shard migration based on our strawman system in section 4.1.

Workloads. We evaluate the performance of GriDB using TPC-H [81] which is widely used by the database community. It consists of 8 tables for each dataset and 22 types of SQL queries. Our experiments are run on a database with 16 TPC-H datasets which are uniformly split across shards. Besides, we add data transactions, each of which insert, delete or update a new row, for the workload of each dataset. To simulate the cross-shard query, there is a proportion of query transactions involving tables in different shards and the proportion is called *cross-shard ratio*. To simulate the workload imbalance, similar to [20, 85], we set two imbalanced settings. For low imbalance, we adopt a Zipfian distribution where two-thirds of the accesses go to

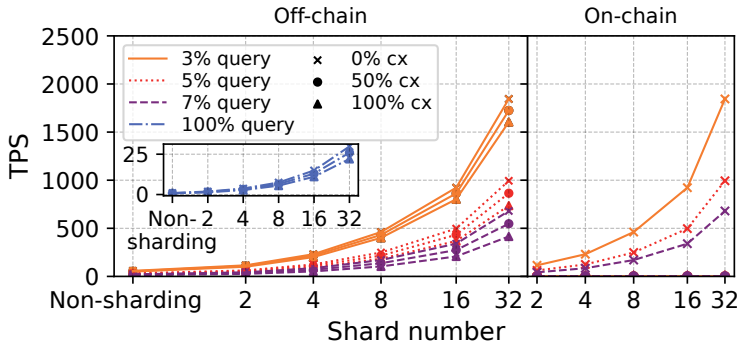


Figure 5.6: Transaction throughput for GriDB, the on-chain sharding blockchain database, and the non-sharding blockchain database (cx means cross-shard ratio.)

one-third of the datasets. For high imbalance, 40% of transactions follow the Zipfian in low imbalance, and the other transactions target 4 datasets initially on the first shard.

5.6.1 Overall Performance

To evaluate the scalability, we measure the transaction throughput in TPS for the non-sharding blockchain database and GriDB with varying percentages of query transactions and cross-shard ratios. We deploy 30 nodes for each shard. Figure 5.7 shows that the measured TPS of GriDB increases linearly with the number of shards and decreases when there are more cross-shard query transactions in the workload. It is because the data transactions only involve one shard, and the verification is simple. However, a query transaction is computationally-intensive (it requires 0.17 $\sim$ 2.38 seconds even in a local database as discussed in subsection 5.6.2) and needs the delegation-based procedure for cross-shard verification, thus, committing query transactions costs more. Moreover, a query involving more shards causes more table transfers and more complex proof generation among the delegated nodes, which will be further studied in subsection 5.6.2. In comparison with GriDB, the on-chain sharding blockchain database has a similar throughput when there are no cross-shard

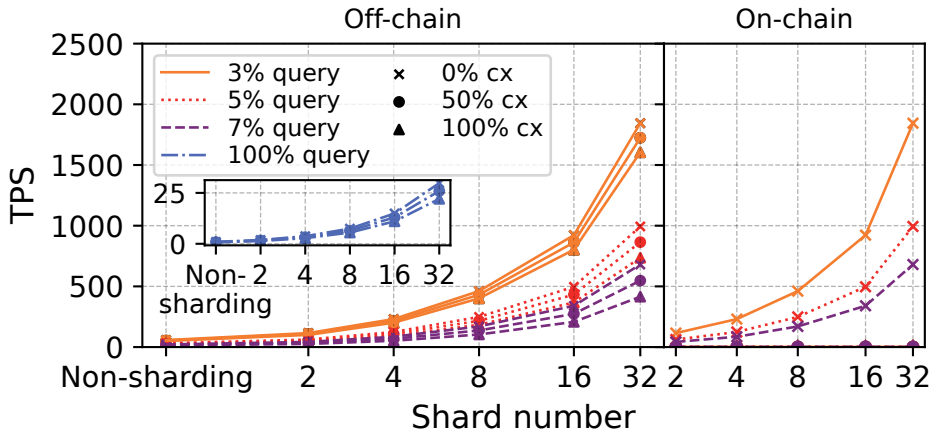


Figure 5.7: Transaction throughput for GriDB, the on-chain sharding blockchain database, and the non-sharding blockchain database (cx means cross-shard ratio.)

queries. However, its throughput drops to nearly 0 when 50% or 100% of queries are cross-shard. It is because, for the on-chain one, the table transfer among shards caused by cross-shard queries can result in serious network blocked.

To evaluate the performance of GriDB for cross-shard data transactions, we pack cross-shard queries (used to delete the cross-shard query results) into cross-shard data transactions. According to Figure 5.7, GriDB's throughput for cross-shard data transactions is similar to that for cross-shard query transactions. It is because, as described in subsection 5.2.1, GriDB implements a delete statement by marking old data as invalid. Except for reaching consensus on cross-shard query results like a cross-shard query transaction, a cross-shard data transaction needs to include the information of marking the results as invalid, and each node needs to delete the results from its in-memory tables. However, these additional overheads are negligible. Thus, the expense of cross-shard data transactions and cross-shard query transactions are similar.

We also evaluate the storage overhead per node after loading all tables in the non-sharding blockchain database and GriDB with varying shard numbers. The results are given in Figure 5.8. Because each row is committed in the form of a data transac-

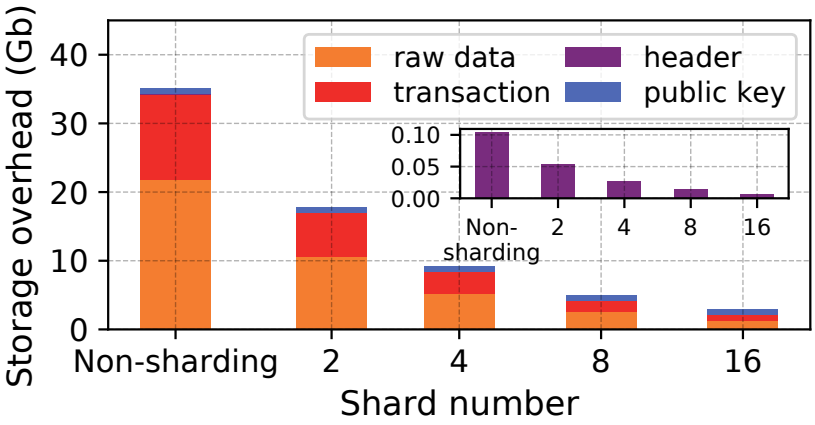


Figure 5.8: Storage overhead per node for GriDB and the non-sharding blockchain database.

tion and the data transactions are packed into blocks, loading the tables will introduce the block-related data including transaction-related and header-related data. From Figure 5.8, we can observe that, first, as the number of shards increases, the storage overhead for each node is reduced. Second, the transaction-related data cost half storage compared with the raw data. Third, compared with the other data, the storage of headers can be ignored. Forth, because the largest table in the evaluation consists of 6 million rows, the public key size of verifiable set operation (VSO) is about 0.76 GB. We regard the storage overhead caused by the public key as acceptable in the case of tables with millions of rows since it is considerably less than the recommended storage space of most blockchain nodes (such as 2 TB in Ethereum [25]) nowadays. Additionally, the storage of the on-chain sharding database is the same as that of GriDB.

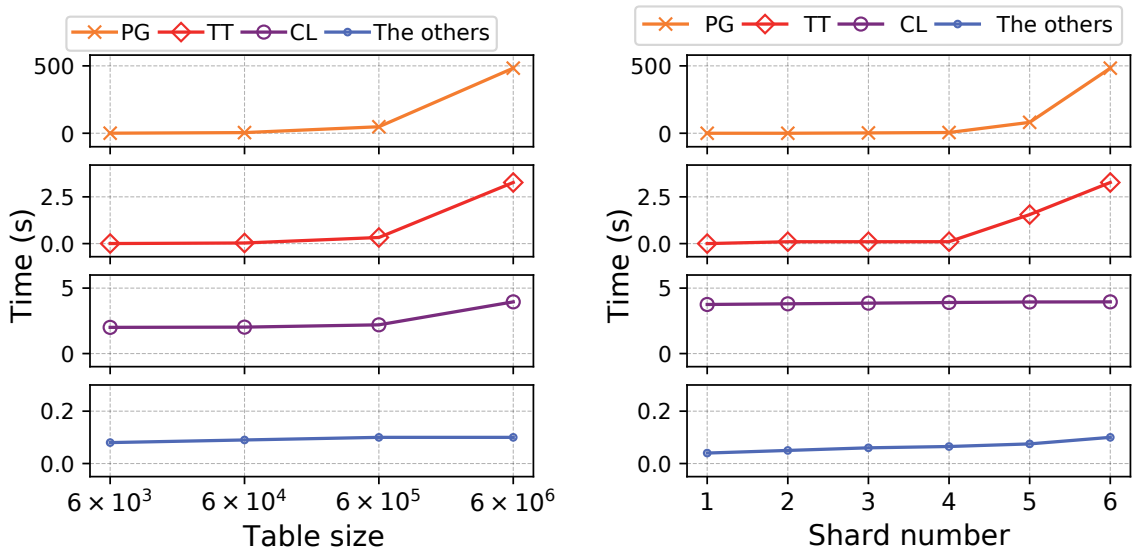
5.6.2 Performance of Cross-shard Query

We evaluate the performance of cross-shard queries. For comparison, we adopt two approaches providing the same functionality as our cross-shard query. These ap-

proaches are motivated by two previous works, i.e., vSQL [94] and libsnark [75], which can support arbitrary SQL queries based on interactive proof and SNARKs, respectively. Depending on either of these two works, any nodes can directly provide the result of a cross-shard query and a proof to the clients without consensus. We also evaluate the performance of the local computation for SQL in our nodes, based on MySQL. The server time is the time required for the server to evaluate the query and produce a valid proof and the client time is the time for the client to verify the proof. In GriDB, the server time is the duration from Line 2 to Line 6 in algorithm 1, and the client time is the duration of Line 7 in algorithm 1.

As a representative example, we pick the query #19, #6, #5, #2 in TPC-H and the results are given in Table 5.1. These queries include most SQL types, e.g., join, range, min and nested query. According to Table 5.1, the server time of GriDB is orders of magnitude less than that of vSQL and SNARKs while the client time is similar. For the server time, it is because our cross-shard query only constructs the expensive ADS for a few cross-shard operators while the security of the other operations depends on the intra-shard consensus. For the client time, it is because the clients of GriDB only need to check whether their query transactions are confirmed or not via SPV. Note that the evaluation is based on the worst case, which means the tables for each cross-shard query are all located in different shards.

The time cost of each step for the queries in GriDB is summarized in Table 5.2. The results shows that the three steps occupy most of the time, matching the performance analysis in subsection 5.3.2. Furthermore, from Table 5.2, we have the following observations. First, according to the result of MySQL in Table 5.1, query #19 is the most complex one and the nodes spend more time on validating it during the intra-shard consensus, thus its confirmation latency is the most. Then, the proof generation and table transfer of query #5 is the most, because the query needs to join six tables, which results in six cross-shard operators in the worst case. Finally, the time cost of query #6 is the least, because it is a simple 3-dimensional range



(a) Table size

(b) Number of the related shards

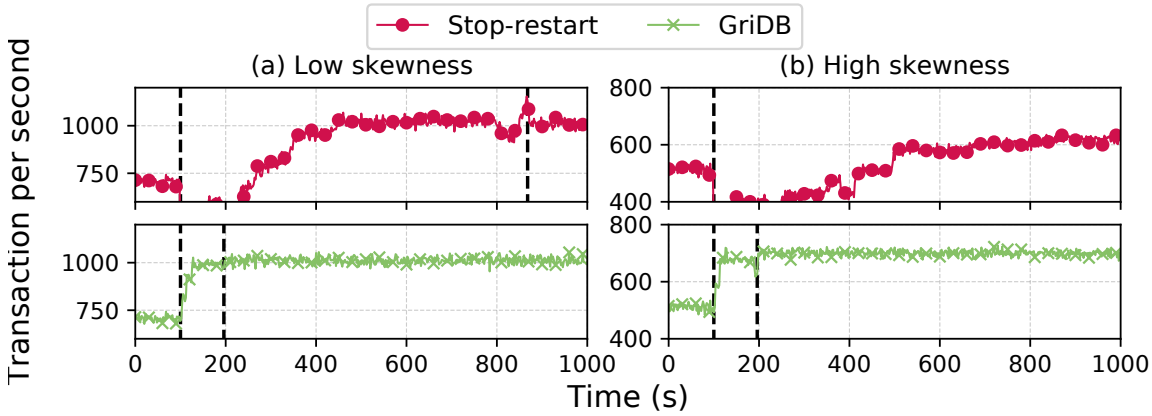
Figure 5.9: Performance for query #5 with different table size and number of related shards in GriDB.

query followed by an aggregation for a single table.

We also evaluate the performance of the cross-shard query of GriDB with varying table size and number of related shards. We scale the number of rows in the largest participating table in query #5 from 6×10^3 to 6×10^6 and distribute its participating tables to $1 \sim 6$ shards. Figure 5.9(a) and Figure 5.9(b) show that the time cost is significantly reduced when the participating tables are smaller or there are fewer related shards. It is because the complexity of proof generation depends on the participating table size and the number of cross-shard operators, matching the analysis in subsection 5.3.2.

5.6.3 Performance of Inter-shard Balancing

We evaluate the throughput during migration via the off-chain live migration in GriDB and the stop-restart approach in the on-chain sharding blockchain database



(a) Fluctuation of throughput during migration.

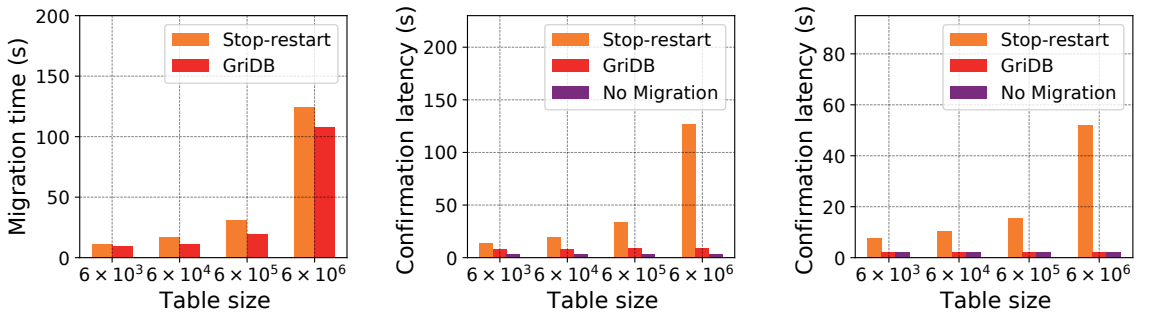
Migration time (s), TPS	Low skewness	High skewness
Stop-restart	770, 981	10678, 623
GriDB	96, 1012	96, 700

(b) Statistics on migration time and throughput.

Figure 5.10: Transaction throughput during inter-shard migration with varying skewness.

with various skewed workloads and the results are given in Figure 5.10. The process includes 48 migrations. After migration, the throughput increases by 1.40× for the low skewness and 1.37× for the high skewness. It shows the load balancing among shards is helpful for the performance of sharding blockchain database. The off-chain live migration can shorten the migration time by nearly 87% compared with the stop-restart approach for the low skewness and 99% for the high skewness. Furthermore, the performance degradation in GriDB is minimal during migration. It is because, in GriDB, the off-chain manner significantly reduces the number of on-chain transactions, avoiding the massive overhead for consensus, and the dual mode minimizes service interruption during migration using the cross-shard off-chain notification.

Figure 5.11 plots the impact of the table size on the migration time, the confirmation latency of transactions for the migrating table and the other tables in the shards involved. Figure 5.11(a) shows that it costs more time to migrate a bigger table for



(a) Migration time

(b) Latency of txs for the migrating table

(c) Latency of txs for the other tables

Figure 5.11: Inter-shard migration for tables with varying size.

both approaches. However, the migration time in GriDB is less than that in the stop-restart approach because there are only two on-chain transactions in GriDB, and a bigger table only requires more transmission time rather than more consensus rounds like the stop-restart approach. According to Figure 5.11(b) and Figure 5.11(c), in GriDB, the confirmation latency for the tables during the migration is similar to that during normal mode (i.e., "No Migration" in the figures). Furthermore, the latency of transactions in the migrating table during migration is more than the latency during normal mode. It is because, in the dual mode, they are required to notify the destination shard.

Algorithm 1: Cross-Shard Query Authentication

Input: query request Q involving tables in a set of shards S

Output: query result R, verification object VO

Delegates $\mathbb{M}$ and $\mathbb{S}$ are selected from S
$\mathbb{M}$ downloads the related tables from $\mathbb{S}$
$\mathbb{M}$ evaluates query result R and get proof Υ via $\mathsf{genProof}(Q)$
$\mathbb{M}$ proposes a cross-shard query transaction txn involving S and including R and Υ
if `validateCx(`S, txn`)` $==$ *True* **then**
    $VO \leftarrow$ the list of SPV proofs in S for txn
Function `genProof(`Q`):`
    for cross-shard operator $op \in Q$ **do**
        Set C^i and C^j as the columns involved by op and pk as the public key
        $(C^*, \pi) \leftarrow \text{prove}(C^i, C^j, pk)$
        Get bm^i and bm^j based on C^i, C^j and C^*
        Add $\langle acc(C^i), acc(C^j), \pi, bm^i, bm^j \rangle$ to Υ
    return Υ
Function `validateCx(`S, txn`):`
    for shard $s \in S$ **do**
        if Υ *or* R is invalid **then**
            return $False$
        txn is committed in the blockchain of s
    return $True$

Table 5.1: Comparison of server and client times for evaluating queries using different approaches (The results for vSQL and SNARKs are provided in [94].)

	vSQL		SNARKs		GriDB		MySQL
Query	Server	Client	Server	Client	Server	Client	
#19	4892s	162ms	196000s	6ms	41.14s	221ms	2.38s
#6	3851s	129ms	19000s	6ms	4.93s	221ms	1.44s
#5	5069s	398ms	615000s	110ms	490.33s	221ms	1.95s
#2	2346s	508ms	58000s	40ms	56.86s	222ms	0.17s

Table 5.2: Time of each step for queries in GriDB (CL: Confirmation latency, PG: Proof generation, TT: Table transfer.)

Query	CL	PG	TT	The others
#19	4.38s	36.74s	4.04ms	10ms
#6	3.44s	1.44s	0s	5ms
#5	3.95s	483.01s	3.2s	100ms
#2	2.17s	54.46s	139.57ms	80ms

Chapter 6

Conclusions and Suggestions for Future Research

6.1 Work Summary

As a prerequisite that blockchain can be broadly applied and provide ubiquitous service, scalability is one of the essential properties. Unfortunately, most of the existing popular blockchain systems suffer from poor scalability. Although sharding is one of the most promising and popular ones to improve blockchain scalability, it suffers from cross-shard transactions and transaction conflict for smart contracts, and is difficult to be applied in the area of blockchain database. To solve these three challenges for blockchain sharding, this report is mainly composed of three following parts.

- We present 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 propose a cooperative cross-shard consensus to enable b-shards to commit the cross-shard transactions without conflict and with the guarantee of security. Based on our experiments, PYRAMID improves the transaction throughput by 1.5 $\sim$ 3.2X against the traditional sharding works and achieves about 3821 TPS when there are 20 shards.

- We present PROPHET, the first sharding blockchain with deterministic ordering for conflict-free transactions. PROPHET achieves conflict-free by introducing a layer-2 sharding architecture on top of the existing shards of blockchain sharding. The running of the architecture depends on the cooperation and supervision among reconnaissance shards, sequence shard, and worker shards. PROPHET also features several improved designs for ordering efficiency, such as fine-grained ordering, asynchronous correction, parallel execution, and trustworthy incentive. Experimental evaluations show that PROPHET boosts the throughput of 3.11$\times$ (i.e., 1203 TPS) compared with previous sharding works.

- We present GRIDB, a sharding blockchain database that achieves a few thousand transactions per second on thousands of nodes in a Byzantine environment while supporting the functionalities of data insert/update, relational queries, and database management. Delegation-based cross-shard query and off-chain live migration are key contributions in GRIDB. They offer a database layer of abstraction on top of the existing sharding blockchain and hide the complexity of the data and workload partition in the underlying sharding blockchain from the clients. GRIDB also includes some database key components, including query optimization, and load scheduler.

6.2 Future Plan

Blockchain's decentralization, security, and scalability have been well studied in recent years, and many potential solutions are being trialed in the running blockchain systems. However, blockchain researchers or companies still have a common and lingering question, i.e., how to use blockchain to improve people's lives? Most of the existing popular blockchain applications, such as crypto games, gambles, and non-fungible tokens (NFT), only have played a role in entertainment. To broaden the application of blockchain and reform industries, we will conduct my future research on the following aspects.

First, in database management, traditional distributed databases are upgraded to blockchain databases. They transact and record data via blockchains and construct an abstract database layer supporting various query functionalities on top of blockchains, which endow the distributed databases with immutability and traceability. Motivated by them, we aim to develop a blockchain knowledge graph to build a trustworthy infrastructure for the semantic web and enable everyone to be both data creator and consumer in Web 3.0. In the platform, every data owner can share its semantic data to construct a global and distributed knowledge graph while collectively maintaining an immutable authenticated data structure through blockchain for data integrity and query verifiability in the knowledge graph.

Federated Learning (FL) is an emerging approach to overcome the challenges of privacy and resource constraints in distributed learning. In FL, model training is distributed across multiple devices or nodes that collaboratively train a shared global model, while keeping the raw data localised on the devices. This is achieved by exchanging model updates, such as gradients or model parameters, rather than sharing sensitive data. Such a decentralised approach ensures privacy and reduces communication overhead compared to sending raw data to a central server. However, as recent work has identified various adversarial attacks on FL, such as data poisoning

and model poisoning, FL accountability, i.e. logging and auditing of the data stream and model training process, is critical and contributes to attribution. Accountability requires tracking model updates, verifying the integrity of participants' contributions, and identifying malicious or faulty nodes. However, it is challenging to implement a trusted party for logging and auditing in the decentralised and Byzantine environment of FL, where participants may act maliciously or fail arbitrarily. In addition, FL systems face scalability issues due to the massive amount of intermediate learning data (e.g., gradients, weights, and updates) generated during training. Therefore, we aim to develop a blockchain-based logging and auditing platform for FL to establish tamper-proof and distributed accountability. Blockchain's decentralised and immutable ledger can serve as a reliable mechanism to log model updates, detect anomalies, and ensure the integrity of the training process. Most importantly, the platform will address the challenges of massive but necessary intermediate learning data and limited functional smart contracts in existing blockchain systems, possibly through off-chain storage solutions or enhanced smart contract designs tailored to FL-specific requirements.

Third, digital government is transforming economic and social life by utilizing digital technology to improve the efficiency and effectiveness of service delivery in the public sector. Blockchain has shown its great potential in the digital government such as blockchain electronic invoices for taxation in Shenzhen. However, the automation level is low, which means the blockchain only provides the functionality of recording data and the people still need to process government affairs manually. Thus, considering that smart contracts can execute the predefined logic automatically and mandatorily, we aim to design smart contract language design for law expression and engine design for law enforcement. Moreover, the technology will become the cornerstone to construct decentralized autonomous organizations (DAO) in metaverse.

References

[1] Daniel J. Abadi and Jose M. Faleiro. An overview of deterministic database systems. *Commun. ACM*, 61(9):78–88, 2018.

[2] Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, and George Danezis. Chainspace: A sharded smart contracts platform. *CoRR*, abs/1708.03778, 2017.

[3] Mohammad Javad Amiri, Divyakant Agrawal, and Amr El Abbadi. *SharPer: Sharding Permissioned Blockchains Over Network Clusters*, page 76–88. Association for Computing Machinery, 2021.

[4] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh, Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. Hyperledger fabric: A distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18. Association for Computing Machinery, 2018.

[5] Georgia Avarikioti, Eleftherios Kokoris-Kogias, and Roger Wattenhofer. Divide and scale: Formalization of distributed ledger sharding protocols, 2019.

[6] Dimitri P Bertsekas et al. *Dynamic programming and optimal control*. Belmont, MA: Athena Scientific, 2011.

[7] George Bissias and Brian N. Levine. Bobtail: Improved blockchain security with low-variance mining. In *NDSS Symposium*, 2022.

[8] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual International Cryptology Conference*, pages 757–788. Springer, 2018.

[9] Dan Boneh, Ben Lynn, and Hovav Shacham. Short signatures from the weil pairing. In Colin Boyd, editor, *Advances in Cryptology — ASIACRYPT 2001*, pages 514–532. Springer Berlin Heidelberg, 2001.

[10] Ran Canetti, Omer Paneth, Dimitrios Papadopoulos, and Nikos Triandopoulos. Verifiable set operations over outsourced databases. In Hugo Krawczyk, editor, *Public-Key Cryptography – PKC 2014*, pages 113–130. Springer Berlin Heidelberg, 2014.

[11] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation (OSDI 99)*. USENIX Association, 1999.

[12] T. Chen, Y. Zhu, Z. Li, J. Chen, X. Li, X. Luo, X. Lin, and X. Zhange. Understanding ethereum via graph analysis. In *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018.

[13] Kristina Chodorow. *Scaling MongoDB: Sharding, Cluster Setup, and Administration*. O'Reilly Media, Inc., 2011.

[14] CoinDesk. Soaring defi usage drives ethereum contract calls to new record. `https://www.coindesk.com/soaring-defi-usage-drives-ethereum-contract-calls-to-new-record`.

[15] George Danezis and Sarah Meiklejohn. Centrally banked cryptocurrencies. *CoRR*, abs/1505.06895, 2015.

[16] Hung Dang, Tien Tuan Anh Dinh, Dumitrel Loghin, Ee-Chien Chang, Qian Lin, and Beng Chin Ooi. Towards scaling blockchain systems via sharding. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19. ACM, 2019.

[17] Azure SQL Database. Scaling out with azure sql database, 2022.

[18] DoltHub. go-mysql-server. `https://github.com/dolthub/go-mysql-server`.

[19] Muhammad El-Hindi, Carsten Binnig, Arvind Arasu, Donald Kossmann, and Ravi Ramamurthy. Blockchaindb: A shared database on blockchains. 12(11):1597–1609, July 2019.

[20] Aaron J. Elmore, Vaibhav Arora, Rebecca Taft, Andrew Pavlo, Divyakant Agrawal, and Amr El Abbadi. Squall: Fine-grained live reconfiguration for partitioned main memory databases. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, SIGMOD '15, page 299–313. Association for Computing Machinery, 2015.

[21] Aaron J. Elmore, Sudipto Das, Divyakant Agrawal, and Amr El Abbadi. Zephyr: Live migration in shared nothing databases for elastic cloud platforms. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data*, SIGMOD '11, page 301–312. Association for Computing Machinery, 2011.

[22] Ethereum. Evm state transition tool. `https://github.com/ethereum/go-ethereum/tree/master/cmd/evm`.

[23] Ethereum. Go ethereum. `https://github.com/ethereum/go-ethereum`.

[24] Ethereum. Shard chains. `https://ethereum.org/en/eth2/shard-chains/`.

[25] Ethereum. Hardware requirements for go-ethereum, 2022.

[26] Jose M. Faleiro, Daniel J. Abadi, and Joseph M. Hellerstein. High performance transactions via early write visibility. *Proc. VLDB Endow.*, 10(5):613–624, 2017.

[27] Kristoffer Francisco and David Swanson. The supply chain has no clothes: Technology adoption of blockchain for supply chain transparency. *Logistics*, 2(1):2, 2018.

[28] Emmanuelle Ganne. *Can Blockchain revolutionize international trade?* World Trade Organization Geneva, 2018.

[29] Zerui Ge, Dumitrel Loghin, Beng Chin Ooi, Pingcheng Ruan, and Tianwen Wang. Hybrid blockchain database systems: Design and performance. *Proc. VLDB Endow.*, 15(5):1092–1104, 2022.

[30] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 51–68. Association for Computing Machinery, 2017.

[31] Google. The go programming language. `https://golang.org/`.

[32] Harmony. Harmony. `https://github.com/harmony-one/harmony`.

[33] Harmony. Harmony consensus protocol design. `https://github.com/harmony-one/harmony/tree/main/consensus`.

[34] Ethan Heilman, Alison Kendler, Aviv Zohar, and Sharon Goldberg. Eclipse attacks on Bitcoin's Peer-to-Peer network. In *24th USENIX Security Symposium (USENIX Security 15)*, pages 129–144. USENIX Association, August 2015.

[35] Jelle Hellings and Mohammad Sadoghi. Byshard: Sharding in a byzantine environment. *Proc. VLDB Endow.*, 14(11):2230–2243, 2021.

[36] Raymond Hemmecke, Matthias Köppe, Jon Lee, and Robert Weismantel. *Nonlinear Integer Programming*, pages 561–618. Springer Berlin Heidelberg, 2010.

[37] Herumi. High-speed software implementation of the optimal ate pairing over barreto-naehrig curves. `https://github.com/herumi/ate-pairing`.

[38] Zicong Hong, Song Guo, Peng Li, and Wuhui Chen. Pyramid: A layered sharding blockchain system. In *IEEE INFOCOM 2021 - IEEE Conference on Computer Communications*, pages 1–10, 2021.

[39] Huawei Huang, Yue Lin, and Zibin Zheng. Account migration across blockchain shards using fine-tuned lock mechanism. In *IEEE International Conference on Computer Communications (INFOCOM)*, 2024.

[40] Huawei Huang, Xiaowen Peng, Jianzhou Zhan, Shenyang Zhang, Yue Lin, Zibin Zheng, and Song Guo. Brokerchain: A cross-shard blockchain protocol for account/balance-based state sharding. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 1968–1977, 2022.

[41] IBM. Blockchain for supply chain solutions. `https://www.ibm.com/blockchain/industries/supply-chain`.

[42] Shan Jiang, Jiannong Cao, Cheung Leong Tung, Yuqin Wang, and Shan Wang. Sharon: Secure and efficient cross-shard transaction processing via shard rotation. In *IEEE International Conference on Computer Communications (INFOCOM)*, 2024.

[43] R. Karp, C. Schindelhauer, S. Shenker, and B. Vocking. Randomized rumor spreading. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 565–574, 2000.

[44] Yeonsoo Kim, Seongho Jeong, Kamil Jezek, Bernd Burgstaller, and Bernhard Scholz. An off-the-chain execution environment for scalable testing and profiling

of smart contracts. In *2021 USENIX Annual Technical Conference (USENIX ATC 21)*, pages 565–579. USENIX Association, July 2021.

[45] Eleftherios Kokoris Kogias, Philipp Jovanovic, Nicolas Gailly, Ismail Khoffi, Linus Gasser, and Bryan Ford. Enhancing bitcoin security and performance with strong consistency via collective signing. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 279–296. USENIX Association, August 2016.

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

[47] Laphou Lao, Xiaohai Dai, Bin Xiao, and Songtao Guo. G-pbft: A location-based and scalable consensus protocol for iot-blockchain applications. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 664–673, 2020.

[48] Mingyu Li, Jinhao Zhu, Tianxu Zhang, Cheng Tan, Yubin Xia, Sebastian Angel, and Haibo Chen. Bringing decentralized search to decentralized services. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI 21)*, pages 331–347. USENIX Association, July 2021.

[49] Jinwen Liang, Zheng Qin, Sheng Xiao, Lu Ou, and Xiaodong Lin. Efficient and secure decision tree classification for cloud-assisted online diagnosis services. *IEEE Transactions on Dependable and Secure Computing*, 18(4):1632–1644, 2021.

[50] libp2p. The go implementation of the libp2p networking stack. `https://github.com/libp2p/go-libp2p`.

[51] Yu-Shan Lin, Ching Tsai, Tz-Yu Lin, Yun-Sheng Chang, and Shan-Hung Wu. Don't look back, look into the future: Prescient data partitioning and migration for deterministic database systems. In *Proceedings of the 2021 International*

Conference on Management of Data, SIGMOD/PODS '21, page 1156–1168. Association for Computing Machinery, 2021.

[52] Mengting Liu, F. Richard Yu, Yinglei Teng, Victor C. M. Leung, and Mei Song. Performance optimization for blockchain-enabled industrial internet of things (iiot) systems: A deep reinforcement learning approach. *IEEE Transactions on Industrial Informatics*, 15(6):3559–3570, 2019.

[53] Yi Lu, Xiangyao Yu, Lei Cao, and Samuel Madden. Aria: A fast and practical deterministic oltp database. *Proc. VLDB Endow.*, 13(12):2047–2060, 2020.

[54] Loi Luu, Viswesh Narayanan, Chaodong Zheng, Kunal Baweja, Seth Gilbert, and Prateek Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS 16)*. ACM, 2016.

[55] Loi Luu, Jason Teutsch, Raghav Kulkarni, and Prateek Saxena. Demystifying incentives in the consensus computer. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 706–719, New York, NY, USA, 2015. Association for Computing Machinery.

[56] Abdullah Al Mamun, Feng Yan, and Dongfang Zhao. Baash: Lightweight, efficient, and reliable blockchain-as-a-service for hpc systems. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21. Association for Computing Machinery, 2021.

[57] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th Annual Symposium on Foundations of Computer Science (Cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.

[58] Atsuki Momose and Ling Ren. Multi-threshold byzantine fault tolerance. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communica-*

tions Security, CCS '21, page 1686–1699. Association for Computing Machinery, 2021.

[59] Einar Mykletun, Maithili Narasimha, and Gene Tsudik. Authentication and integrity in outsourced databases. *ACM Trans. Storage*, 2(2):107–138, 2006.

[60] MySQL. Mysql 8.0 reference. `https://dev.mysql.com/doc/refman/8.0/en/sql-data-manipulation-statements.html`.

[61] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[62] L. N. Nguyen, T. D. T. Nguyen, T. N. Dinh, and M. T. Thai. Optchain: Optimal transactions placement for scalable blockchain sharding. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 525–535, 2019.

[63] Charalampos Papamanthou, Roberto Tamassia, and Nikos Triandopoulos. Optimal verification of operations on dynamic sets. In Phillip Rogaway, editor, *Advances in Cryptology – CRYPTO 2011*, pages 91–110. Springer Berlin Heidelberg, 2011.

[64] Q. Pei, E. Zhou, Y. Xiao, D. Zhang, and D. Zhao. An efficient query scheme for hybrid storage blockchains based on merkle semantic trie. In *2020 International Symposium on Reliable Distributed Systems (SRDS)*, pages 51–60, 2020.

[65] Yanqing Peng, Min Du, Feifei Li, Raymond Cheng, and Dawn Song. Falcondb: Blockchain-based collaborative database. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 637–652. Association for Computing Machinery, 2020.

[66] George Pîrlea, Amrit Kumar, and Ilya Sergey. *Practical Smart Contract Sharding with Ownership and Commutativity Analysis*, page 1327–1341. Association for Computing Machinery, 2021.

[67] Litecoin project development team. Litecoin - open source p2p digital currency. `https://litecoin.org/`.

[68] Xiaodong Qi. S-store: A scalable data store towards permissioned blockchain sharding. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 1978–1987, 2022.

[69] Xiaodong Qi and Yi Li. Lightcross: Sharding with lightweight cross-shard execution for smart contracts. In *IEEE International Conference on Computer Communications (INFOCOM)*, 2024.

[70] Raghu Ramakrishnan and Johannes Gehrke. *Database Management Systems*. McGraw-Hill, Inc., 2nd edition, 2000.

[71] Sukriti Ramesh, Odysseas Papapetrou, and Wolf Siberski. Optimizing distributed joins with bloom filters. In *Distributed Computing and Internet Technology*, pages 145–156. Springer Berlin Heidelberg, 2009.

[72] BitMEX Research. Bitcoin vs ethereum – blockchain size. `https://blog.bitmex.com/bitcoin-vs-ethereum-blockchain-size/`.

[73] Pingcheng Ruan, Gang Chen, Tien Tuan Anh Dinh, Qian Lin, Beng Chin Ooi, and Meihui Zhang. Fine-grained, secure and efficient data provenance on blockchain systems. *Proc. VLDB Endow.*, 12(9):975–988, may 2019.

[74] Pingcheng Ruan, Tien Tuan Anh Dinh, Dumitrel Loghin, Meihui Zhang, Gang Chen, Qian Lin, and Beng Chin Ooi. Blockchains vs. distributed databases: Dichotomy and fusion. In *Proceedings of the 2021 International Conference on Management of Data*, SIGMOD '21, page 1504–1517, New York, NY, USA, 2021. Association for Computing Machinery.

[75] SCIPRLab. libsnark: a c++ library for zksnark proofs. `https://github.com/scipr-lab/libsnark`.

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

[77] Rebecca Taft, Essam Mansour, Marco Serafini, Jennie Duggan, Aaron J. Elmore, Ashraf Aboulnaga, Andrew Pavlo, and Michael Stonebraker. E-store: Fine-grained elastic partitioning for distributed transaction processing systems. *Proc. VLDB Endow.*, 8(3):245–256, November 2014.

[78] Y. Tao, B. Li, J. Jiang, H. C. Ng, and B. Li C. Wang. On sharding open blockchains with smart contracts. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, 2020.

[79] Zilliqa team. Zilliqa. `https://www.zilliqa.com/`.

[80] Alexander Thomson, Thaddeus Diamond, Shu-Chun Weng, Kun Ren, Philip Shao, and Daniel J. Abadi. Calvin: Fast distributed transactions for partitioned database systems. In *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*, SIGMOD '12, page 1–12. Association for Computing Machinery, 2012.

[81] TPC. Tpc-h benchmark. `http://www.tpc.org/tpch/`.

[82] Gang Wang, Zhijie Jerry Shi, Mark Nixon, and Song Han. Sok: Sharding on blockchain. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*. ACM, 2019.

[83] Jiaping Wang and Hao Wang. Monoxide: Scale out blockchains with asynchronous consensus zones. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*. USENIX Association, 2019.

[84] Qin Wang, Rujia Li, Qi Wang, and Shiping Chen. Non-fungible token (nft): Overview, evaluation, opportunities and challenges, 2021.

[85] Xingda Wei, Sijie Shen, Rong Chen, and Haibo Chen. Replication-driven live reconfiguration for fast distributed transaction processing. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 335–347. USENIX Association, July 2017.

[86] Sam M. Werner, Daniel Perez, Lewis Gudgeon, Ariah Klages-Mundt, Dominik Harz, and William J. Knottenbelt. Sok: Decentralized finance (defi), 2021.

[87] Gavin Wood. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 2014.

[88] Cheng Xu, Ce Zhang, and Jianliang Xu. Vchain: Enabling verifiable boolean range queries over blockchain databases. In *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, page 141–158. Association for Computing Machinery, 2019.

[89] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS 18)*. ACM, 2018.

[90] Bo Zhang, Boxiang Dong, and Wendy Hui Wang. Integrity authentication for sql query evaluation on outsourced databases: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 33(4):1601–1618, 2021.

[91] Ce Zhang, Cheng Xu, Jianliang Xu, Yuzhe Tang, and Byron Choi. Gem2-tree: A gas-efficient structure for authenticated range queries in blockchain. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 842–853, 2019.

[92] Jianting Zhang, Wuhui Chen, Sifu Luo, Tiantian Gong, Zicong Hong, and Aniket Kate. Front-running attack in sharded blockchains and fair cross-shard consensus. In *NDSS*, 2024.

[93] Jianting Zhang, Zicong Hong, Xiaoyu Qiu, Yufeng Zhan, Song Guo, and Wuhui Chen. Skychain: A deep reinforcement learning-empowered dynamic blockchain sharding system. In *49th International Conference on Parallel Processing - ICPP*, ICPP '20. Association for Computing Machinery, 2020.

[94] Yupeng Zhang, Daniel Genkin, Jonathan Katz, Dimitrios Papadopoulos, and Charalampos Papamanthou. vsql: Verifying arbitrary sql queries over dynamic outsourced databases. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 863–880, 2017.

[95] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. Integridb: Verifiable sql for outsourced databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, page 1480–1491. Association for Computing Machinery, 2015.

[96] Peilin Zheng, Zibin Zheng, and Hong-ning Dai. Xblock-eth: Extracting and exploring blockchain data from ethereum. *Working Report*, 2019.

[97] Peilin Zheng, Zibin Zheng, Jiajing Wu, and Hong-ning Dai. Xblock-eth: Extracting and exploring blockchain data from ethereum. *IEEE Open Journal of the Computer Society*, 1:95–106, 2020.

[98] Yanchao Zhu, Zhao Zhang, Cheqing Jin, Aoying Zhou, and Ying Yan. Sebdb: Semantics empowered blockchain database. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1820–1831, 2019.