CoChain: High Concurrency Blockchain Sharding via Consensus on Consensus

Mingzhe Li*†◇, You Lin*, Jin Zhang*✉, Wei Wang†✉

*Guangdong Provincial Key Laboratory of Brain-inspired Intelligent Computation, Research Institute of Trustworthy Autonomous Systems, Computer Science and Engineering Department, Southern University of Science and Technology

†Computer Science and Engineering Department, Hong Kong University of Science and Technology

◇Institute of High Performance Computing, and Agency for Science, Technology and Research, Singapore

mlibn@connect.ust.hk, liny2021@mail.sustech.edu.cn, zhangj4@sustech.edu.cn, weiwa@cse.ust.hk

***Abstract*—Sharding is an effective technique to improve the scalability of blockchain. It splits nodes into multiple groups so that they can process transactions in parallel. To achieve higher parallelism and concurrency at large scales, it is desirable to maintain a large number of small shards. However, simply configuring small shards easily results in a higher fraction of malicious nodes inside shards, causing shard corruption and compromising system security. Existing sharding techniques hence demand *large shards*, at the expense of limited concurrency. To address this limitation, we propose CoChain: a blockchain sharding system that can securely configure small shards for enhanced concurrency. CoChain allows some shards to be corrupted. For security, each shard is monitored by multiple other shards. The latter reach a cross-shard Consensus on the Consensus results of their monitored shard. Once a corrupted shard is found, its subsequent consensus will be taken over by another shard, hence recovering the system. Via Consensus on Consensus, CoChain allows the existence of shards with more fraction of malicious nodes (<2/3) while securing the system, thus reducing the shard size safely. We implement CoChain based on Harmony and conduct extensive experiments. Compared with Harmony, CoChain achieves 35x throughput gain with 6,000+ nodes.**

I. Introduction

Blockchain has played an important role for enabling decentralized digital currencies [24]. However, traditional blockchain comes at the price of throughput scalability [24], [36]. Sharding [20] is one of the most promising approaches to increase blockchain scalability. Its main idea is to split the nodes in the network into multiple smaller committees (shards), so they can process incoming transactions and reach consensus in parallel [19]. Generally, under the same network scale, it is desirable to configure a large amount of small shards for better transaction concurrency and throughput [10], [25].

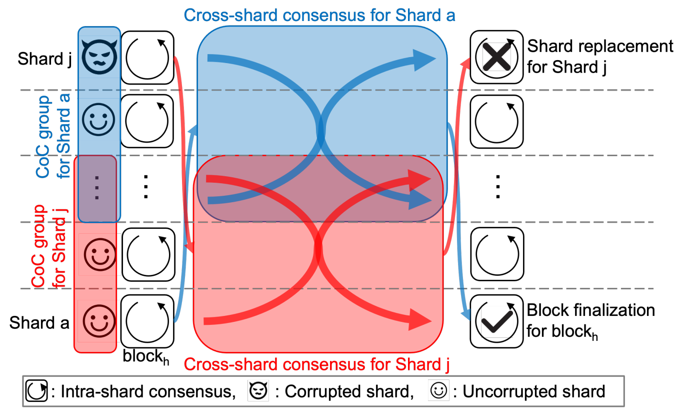
However, simply configuring a large number of small shards tends to cause the corruption of some shards, which *compromises system security* [20]. Therefore, existing works typically configure *large shards*, which severely *limits the transaction concurrency* of the large-scale blockchain sharding systems [2], [17], [33]. Specifically, nodes in blockchain sharding systems are usually *randomly assigned* to individual shards [18], [20], [39]. Due to such randomness, simply configuring small shards can easily over-proportion malicious nodes in some shards and lead to shard corruption (e.g., $\geq 1/3$ within a shard for BFT-typed consensus) [10], [25]. In the existing blockchain sharding systems, any corrupted shards can compromise system security. As a result, existing works typically configure large shard sizes to guarantee, with an extremely large probability, that each shard is not corrupted. For example, in OmniLedger [17], when the whole system can tolerate <1/4 fraction of malicious nodes, each shard needs to be configured with 600 nodes to guarantee no corruption (i.e., <1/3 fraction of malicious nodes per shard). Such large shard sizes slow down the intra-shard consensus speed and reduce the number of shards in the whole network, significantly degrading the transaction concurrency in existing large-scale blockchain sharding systems.

Some previous studies propose large-scale blockchain sharding systems with reduced shard sizes, yet their solutions have various limitations. For example, some works reduce the shard size at the expense of reducing the fault resiliency of the whole system [15], [17], [18]. For instance, in Pyramid [15], the system resiliency is reduced from 1/4 to 1/8, meaning the whole system can only tolerate <1/8 fraction of malicious nodes, reducing system security. More related works are in Section II.

In this paper, our target is to build a blockchain sharding system with *enhanced transaction concurrency without compromising system security*. To *improve concurrency*, unlike previous systems where every shard is required to be uncorrupted, we *allow the existence of some corrupted shards* due to the randomness of node assignment. As a result, the size of each shard can be reduced. However, without judicious design, system security would be compromised by the presence of corrupted shards. This leads to our first challenge.

As mentioned, the ***first challenge*** is how to guarantee system security in the presence of some corrupted shards. To address this, our core idea is that for each shard, there are multiple other shards that monitor it and conduct cross-shard Consensus on its Consensus results. Moreover, when a corrupted shard is found, replace it with another shard to recover system security. An illustration is shown in Figure 1. Specifically, due to the randomness of node assignment, which shards are corrupted is unknown in advance. Therefore, every shard needs to be monitored by multiple other shards. Those shards form a consensus group (named CoC group) and reach a cross-shard consensus on the consensus results of their monitored shard (hence named Consensus on Consensus, ***CoC***). For the CoC, our intuition is to *analogize each shard to a node*, and design the CoC as a PBFT-typed (Practical Byzantine Fault Tolerance [5]) cross-shard consensus. To ensure the security of CoC, we judiciously fine-tune the size of CoC groups and keep the fraction of corrupted shards in each CoC group to be less than 1/3. Normally, the CoC helps the uncorrupted shards to finalize their produced blocks. If a corrupted shard is detected (e.g., it launches attacks), another shard will replace the corrupted one by taking over its subsequent consensus, thus recovering the system.

The ***second challenge*** is how to monitor other shards with


Cross-shard consensus for Shard a
Shard j
CoC group for Shard a
CoC group for Shard j
Shard a
block$_h$
Cross-shard consensus for Shard j
Shard replacement for Shard j
Block finalization for block$_h$
: Intra-shard consensus, : Corrupted shard, : Uncorrupted shard
Fig. 1: Illustration of CoChain's main idea.

low overhead. To detect whether other shards launch attacks (whether corrupted), a naive design is to require each shard store other shards' states and verify every transaction packaged by them. However, this imposes huge storage, communication, and computation overhead to each shard. To reduce the overhead during CoC, our main idea is to *leverage the intra-shard consensus to guarantee the validity of specific transactions, and leverage the cross-shard CoC to detect other typical attacks.* To achieve this, we observe that if there are <2/3 fraction of malicious nodes in a corrupted shard, a block containing invalid transactions will not pass the intra-shard consensus, because it cannot collect enough votes. However, such a corrupted shard can still launch other kinds of attacks (e.g., equivocation, silence attack) [29]. Fortunately, those attacks can be detected via some metadata (e.g., block headers). Based on the observations, our solution is that we configure any corrupted shard to have <2/3 fraction of malicious nodes through rigorous theoretical calculations. As a result, the shards in a CoC group do not require to store extra states, obtain transactions, or verify transactions of their monitored shard. They only need to obtain the block headers from the monitored shard and detect other attacks. This design allows each shard to conduct CoC with low overhead.

The ***third challenge*** is how to maintain high efficiency in CoC while ensuring security. To maintain high CoC efficiency, we mainly propose the *pipelining mechanism.* Each shard optimistically produces new blocks while waiting for others to conduct CoC for it. However, to ensure security, we require that an optimistically produced block can be considered finalized only if it passes the CoC. Moreover, the cross-shard transactions will be sent only after a block is finalized to prevent rollback attacks on multiple shards.

With the above challenges addressed, we propose CoChain, a high concurrency blockchain sharding system with recovery ability. CoChain allows the existence of some corrupted shards with a larger fraction of malicious nodes (i.e., <2/3). To ensure security, we design an efficient and secure CoC protocol. Through mutual monitoring between shards, CoC enables timely shard replacement and system recovery when a corrupted shard is detected. Therefore, CoChain can safely reduce the shard size without compromising the overall fault resiliency of the system, thus improving concurrency.

We implement a prototype of CoChain based on Harmony [33], a well-known public blockchain sharding project. We conduct extensive and large-scale experiments on Amazon EC2. Experimental results demonstrate that in a large-scale network of 6,100 nodes, CoChain improves 35× of the system throughput and 5× of the throughput per shard compared to Harmony.

II. Background and Related Work

A. Blockchain and Sharding

Blockchain has drawn significantly attentions from research and industry areas [11], [24], [36]. However, traditional blockchain cannot scale its transaction processing capacity. One promising approach to scale blockchain is via sharding, which has been an active research topic in industry and academia [20], [27], [32], [33]. Its main idea is to split the set of nodes into a number of smaller committees (shards). Each shard maintains a disjoint subset of states, processes different transactions in parallel, and reaches intra-shard consensus in parallel.

B. Shard Size, Concurrency and Security

If security is not a concern, a large number of small shards will usually enhance the system concurrency under the same network scale. This is because, first, more shards bring higher parallelism in transaction processing. Second, the smaller shard size reduces the communication overhead of intra-shard consensus, hence speeding up the intra-shard consensus process.

However, security is a crucial concern in blockchain (especially permissionless) sharding [30]. Simply configuring small shards in large-scale network can easily compromise system security. This is because the nodes are usually randomly assigned to each shard in blockchain sharding. Due to such randomness, smaller shards easily cause some shards to be assigned with a larger fraction of malicious nodes, corrupting the shards and compromising system security. Existing works hence configure very large shard size to guarantee, with extremely high probability, that *each shard will not be corrupted* [8], [15], [17], [39] (e.g., 600 nodes per shard in OmniLedger [17]).

Several works aim to improve the system concurrency by reducing the shard size, yet they have various drawbacks. Some solutions reduce the shard size with the price of reducing the number of malicious nodes that the system can tolerate and compromising system security [15], [17], [18]. For example, Pyramid [15] reduces the system resiliency to 1/8 (from 1/4).

Some other attempts to reduce the shard size are, however, less practical [8], [16], [39], [40]. For example, RapidChain [39] increases the shard resiliency to 1/2 and reduces the shard size by assuming a *synchronous network* inside each shard. However, this assumption is less practical, especially in public blockchain with large-scale network [29]. In [8], they reduce the shard size by using trusted execution environment (TEE)

[31]. However, their design requires *additional hardwares* on each node, limiting their generality. Some works can configure very small shard sizes [1], [9], [14]. However, their overall network scale is very small. Therefore, in real blockchain systems (usually with thousands of nodes), their systems still need to configure large shard sizes for system security.

Contrary to the previous studies, we allow the existence of some corrupted shards (with <2/3 fraction of malicious nodes), and propose the CoC protocol to ensure security. Therefore, CoChain reduces the shard sizes in large-scale network for boosted concurrency, without compromising system security (tolerating 1/4 fraction of malicious nodes in the whole system). Moreover, CoChain requires no additional hardware and is adaptable to practical network environments.

C. Sharding with Corrupted Shards

Some blockchain sharding works allow the existence of corrupted shards. However, they are with various limitations. For example, works like [25], [28] require synchronous network, which is a less practical assumption in real-world large-scale blockchain network. Moreover, Free2Shard [28] adopts PoW consensus, which does not guarantee deterministic finality and is prone to forking. In [10], it requires all the nodes to run a network-wide consensus for security, which could easily be the performance and security bottleneck of the system. In Near [32], nodes can play as challengers to challenge the corrupted shards. However, the system security might be compromised when a malicious challenger spam with invalid challenges. In CoChain, each shard is monitored jointly by multiple shards via the CoC protocol, which ensures security.

D. Strengthened Fault Tolerance

A few non-sharding research has studied the problem of increasing fault resiliency [21], [37]. However, it is not proven whether their protocols can guarantee safety and liveness in blockchain sharding systems. Moreover, our intuition is different. In CoChain, *the corrupted shard itself is not recovered.* Instead, we leverage other uncorrupted shards to monitor and replace the corrupted shard to *recover the system.*

III. System and Threat Model

A. System Model

In CoChain, there are N nodes and S shards in the system, each shard thus has $n = N/S$ nodes. The CoC size in the system is $m < S$, meaning that for any shard i, there are m shards conducting CoC for it. Those m shards monitoring shard i form a CoC group, denoted as C_i. Like most practical blockchain systems [2], [6], [15], [17], [20], the nodes in CoChain are connected by a *partially synchronous* peer-to-peer network, in which there exists an unknown global stabilization time after which all messages sent are delivered in less than a fixed amount of time [19]. Like existing systems, each node has a unique public/secret key pair given by a Public-Key Infrastructure (PKI). A public key represents the identity of a node. It will be broadcast through the network and recorded once a node joins [15].

CoChain adopts the account model (similar to Ethereum) [36] to represent the ledger state, in which each account has its own states and nodes in different shards record the states (e.g., balance) for different accounts. The states of an account are assigned to one shard for maintenance based on the hash of its account address [15], [33], [35]. Therefore, a transaction in the network will be routed to the corresponding shard based on its associated account address.

B. Threat Model

There are two kinds of nodes in CoChain: honest and malicious. The honest nodes obey all the protocols. However, malicious (Byzantine) nodes may corrupt the protocols in arbitrary manners, such as arbitrarily packing invalid transactions into blocks (e.g., transaction manipulation), sending messages with different values to different nodes (e.g., equivocation attack), or failing to send any or all messages (e.g., silence attack). The fraction of total malicious nodes in the system is denoted as F, meaning FN nodes are controlled by Byzantine adversaries in the whole system. The fraction of malicious nodes in each shard is denoted as f, meaning fn nodes are Byzantine in each shard.

Similar to most existing blockchain sharding systems, we assume that the Byzantine adversaries are *slowly-adaptive*, i.e., the set of malicious nodes and honest nodes are fixed during each epoch and can be changed only between epochs [19], [26]. Also, all nodes have access to an external random oracle H which is collision-resistant, like other works [39], [40].

IV. Architecture of CoChain

CoChain is a high concurrency blockchain sharding system with recovery ability. The running of CoChain proceeds in fixed time periods called *epochs*. The length of the epoch can usually be tuned according to the system requirements (e.g., one day, as many existing blockchain sharding systems adopt [8], [15], [33], [39].).

During each epoch, each shard produces blocks via ***intra-shard consensus*** in parallel and commits the blocks that pass the consensus to its own shard chain. In CoChain, we choose the leaderless BFT consensus protocol proposed by Red Belly [6] as our intra-shard consensus protocol. In the protocol, each node in a shard proposes a micro-block, which is then merged into a complete block via consensus. The protocol is proven to guarantee safety and liveness in partially-synchronous network. Readers can refer to [6] for specific descriptions. We make such a choice mainly to eliminate the influence of a single leader on the intra-shard consensus results. This is because a malicious leader in a shard may cause the uncorrupted shard to temporarily fail, preventing the CoC from being reached efficiently. Alternatively, leader-based consensus protocols [27], [38] can also be adopted.

In a blockchain sharding system, each shard also needs to handle ***cross-shard communications***. In CoChain, the cross-shard communications include cross-shard transactions and cross-shard messages generated during CoC (details in Section V). All the cross-shard communications are sent by all the nodes (for security) in their source shards and are routed (e.g., via Kademlia [22], [39]) to their destination shards for processing, like many existing works [15], [16], [33], [39]. The cross-shard transactions are those transactions sent from one shard to another, which are processed via the traditional cross-shard transaction relay scheme (e.g., [33], [35]) for efficiency and atomicity. Specifically, a cross-shard transaction is split into two parts and processed successively by the source shard and the destination shard. However, in CoChain, to ensure security, only when a block is finalized, the cross-shard transactions in it can be sent (details in Section V-D).

Between any two epochs, CoChain has a ***shard reconfiguration*** process [15], [33], [39], [40] to determine which nodes should be reassigned to which shards in the next epoch. This is mainly designed to sustain attacks from the slowly-adaptive adversaries. For security, each node is *randomly assigned* to a specific shard based on its identity and a randomness. We leverage a combination of verifiable random function (VRF) [23] and verifiable delay function (VDF) [3] to generate verifiable, unbiased, and unpredictable distributed randomness. Moreover, there is also a *beacon chain* in CoChain to perform tasks such as recording node's identity and assisting the process of shard reconfiguration. However, unlike previous works which usually assume a trusted beacon chain, CoChain allows the beacon chain to be corrupted and be replaced to recover the system. We also assume a trusted ***bootstrapping process*** before the first epoch, which is widely adopted in research and industrial areas [17], [33].

V. Protocol Design of CoC

The main design difference between our system and previous systems is the Consensus on Consensus protocol. During each epoch, each shard i continuously sends its intra-shard consensus result to multiple other shards, those shards reach a cross-shard BFT consensus to determine the validity of the intra-shard consensus result (Section V-A, V-B). If the intra-shard consensus result is valid, the CoC protocol helps to finalize it. Otherwise, the corrupted shard i is replaced by another shard to recover the safety (all honest nodes agree on the same value in the same order) and liveness (system continuously makes progress) of the system (Section V-C).

A. Basic Design of CoC

In CoChain, each shard i not only processes transactions, but also continuously sends intra-shard consensus result (details in Section V-B) to m shards in order to be monitored by them. Those m shards form a CoC group C_i that monitor shard i. Meanwhile, each shard also belongs to m different CoC groups that monitor different shards, it receives consensus result from m different shards in order to monitor them. The m shards belonging to the same CoC group C_i perform CoC on shard i to reach a cross-shard consensus on shard i's intra-shard consensus results.

The CoC protocol includes three main phases: pre-prepare, prepare and commit, as illustrated in Figure 2.

Pre-prepare. In this phase, the shards participating in the same CoC group C_i receive the consensus result from the monitored shard i. Each of them first reaches intra-shard consensus on whether the result is valid (i.e., cross-shard verification, see Section V-B). If the result is judged to be valid, the `Prepare` message is generated for it. The `Prepare` message is then sent to the other shards in C_i and the protocol enters the Prepare phase. If the intra-shard consensus determines that the received result is not valid, it generates a `Complain` message and sends it to the other shards in C_i. The shard replacement is then triggered later (see Section V-C) to recover the system.

Prepare. In this phase, the shards participating in the same CoC group C_i receive each other's voting messages in the Pre-prepare phase. For each of them, after receiving more than $2m/3$ consistent `Prepare` messages from different shards, it reaches an intra-shard consensus and generate the `Commit` message accordingly. After that, the `Commit` message is sent to other shards in C_i.

Commit. In this phase, the shards participating in the same CoC group C_i receive each other's voting messages in the Prepare phase. For each of them, after receiving more than $2m/3$ consistent `Commit` messages from different shards, it first reaches intra-shard consensus and generate the `Finalized` message. After that, the `Finalized` message is sent back to the monitored shard i.

CoC leads to two consensus outcomes. First, the monitored shard i is uncorrupted. In this case, after receiving the CoC consensus result (`Finalized`), shard i will attach that message to the block header as a proof of block being finalized, and continue with the subsequent consensus. Second, shard i is corrupted (i.e., $f \geq 1/3$). In this case, the shard replacement mechanism will be triggered. Among the shards in C_i, one shard will be elected to replace the corrupted shard i for subsequent consensus (see Section V-C).

Designs on Security. A *suitable CoC size m* needs to be decided to ensure the security of CoC. CoC can be seen as a cross-shard PBFT consensus. Therefore, it needs to ensure that, for each CoC group with size m, more than $2m/3$ of the participated shards are honest. For this reason, we fine-tune m as the minimum value that guarantees the security of CoC with extremely large probability. The specific equations and proofs are given in Section VI-A. By configuring appropriate m, the CoC can be ensured not to fail with extremely large probability (e.g., one failure in hundreds of years, like existing studies), thus not compromising system security.

We exploit the randomness generated during the shard reconfiguration (Section IV) to determine *which shards should be in each CoC group.* Each shard combines this randomness with the ID of its own shard (e.g., shard i) to generate a new random number. Shard i uses the new random number as a seed to map it (via the random oracle) to m random shards forming the same CoC group C_i that monitors it.

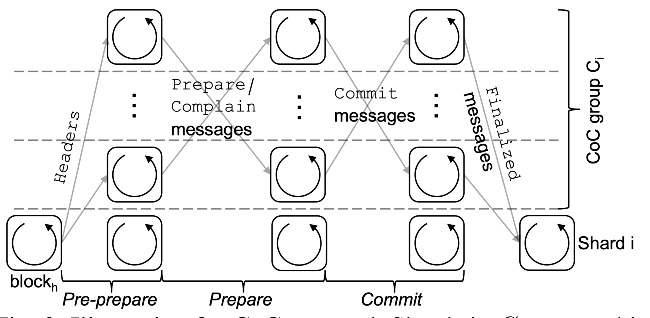


Fig. 2: Illustration for CoC protocol. Shards in C_i are reaching CoC for the block in shard i with height h. If attacks are found, shards send `Complain` and trigger Shard Replacement.

B. *Efficient Cross-Shard Verification*

How to monitor other shards during CoC with low overhead is a critical issue. A naive design requires each shard to store the states of other shards, receive transactions from other shards, and use the states to verify received transactions. While this design can detect whether other shards are launching attacks (whether corrupted), it imposes huge storage, communication, and computation overhead to each shard.

To address this issue, we first observe that when the fraction of malicious nodes in a shard is controlled to be less than 2/3 (i.e., $f < 2/3$), those attacks against specific transactions (e.g., replay attacks, transactions with invalid contents, data unavailability, etc.) will not be successfully launched. This is because honest nodes within the shard can always detect the attacks against specific transactions during intra-shard consensus and will not sign the blocks with such attacks for confirmation. Therefore, in the case of $f < 2/3$, any block with such attacks will not pass the intra-shard consensus due to insufficient signatures, i.e., blocks that successfully pass the intra-shard consensus only contain valid transactions.

However, a corrupted with $f < 2/3$ can still launch other typical attacks. For instance, a corrupted shard may launch equivocation attacks to fork the shard chain by producing multiple blocks containing valid transactions with the same height, or it can remain silent to compromise liveness. Those attacks need to be detected by CoC. Fortunately, detecting such attacks does not require the state information, but only require the block headers, which is with low overhead.

Therefore, we conduct rigorous theoretical and empirical calculations (details in Section VI-A), and fine-tune the size of each shard to ensure that if a shard is corrupted, it will has less than 2/3 fraction of malicious nodes with extremely large probability. As a result, a monitored shard i only needs to send each block header (as the consensus result) that have passed the intra-shard consensus to the CoC group C_i. The shards in C_i then use the header to detect possible attacks (e.g., equivocation) and reach consensus.

In case shard i launches silence attack, we adopt a timeout scheme to ensure liveness. Similar idea is used in most BFT-typed consensus to guarantee liveness under partial-synchronous network. However, in CoC, each node in any CoC group C_i maintains a local timer for shard i. When more than 2/3 fraction of the nodes in a shard of C_i have their timers timed out, the shard reaches intra-shard consensus and generates the `Complain` message. When more than $2m/3$ of the shards in C_i generate `Complain` messages, shard replacement is triggered. The timeout value increases as the system runs and eventually stabilizes. Readers may refer to [5] for details on configuring timeout. Although this scheme may affect uncorrupted shards in the early stages of the system, the effect is usually temporary (e.g., one epoch) and does not affect system security. To sum up, the scheme does not require synchronization between nodes and works properly as the network stabilizes, like existing works [5], [6]. We consider refined schemes (e.g., in [4], [12]) for our future work.

C. *Shard Replacement*

When a CoC group detects a corrupted shard, how to recover the safety and liveliness of the system is an important issue. We address this issue once and for all by leveraging another shard to take over the corrupted shard's subsequent consensus until the start of the next epoch.

During cross-shard consensus, if the monitored shard i is found to be corrupted, one of the shards in CoC group C_i will replace the corrupted shard and restores the system. Specifically, when a shard in C_i receives more than $2m/3$ `Complain` messages from different shards, it sends these `Complain` messages together as a proof to shard i and stops subsequent CoC consensus work for shard i (as shard i is corrupted). After the shard i receives the proof formed by the `Complain` information, each node within that shard sends the state information that is latest finalized by CoC, the transactions (including rollback transactions), along with a proof (i.e., `Finalized` information) to the shard in C_i who is responsible for replacing shard i. The latter after receiving those messages, takes over the subsequent transactions processing from the corrupted shard. Those subsequent transactions will be routed to the new shard that replacing shard i. The shard replacing shard i will process the transactions from both shard i and itself. The corrupted shard, in turn, stops subsequent consensus until the start of the next epoch.

Designs on Security. We exploit distributed randomness to determine *which shard in the CoC group C_i is responsible for replacing the corrupted shard i*. We use the randomness generated during the shard reconfiguration, and randomly map it to one of the shards participated in C_i. The chosen shard is then responsible for taking over the work of the corrupted shard.

We need to *guarantee the validity of the state* information passed by the corrupted shard in order to allow the successor shard to process subsequent transactions safely. For this purpose, the block header generated by each shard will contain the root of the Merkle Tree [7] generated from its latest state [34]. Since the CoC reaches a consensus on the block header, the state information in the block header finalized by the CoC can be considered valid. Therefore, each node in the corrupted shard needs to send the state information that is latest finalized by the CoC to the shard that replaces it, along with the header and the CoC's `Finalized` information on that header. The received shard can then verify the validity of the state and prevent attackers from tampering with the state information.

D. Pipelining Mechanism

Since each block proposed by each shard depends on CoC to finalize, maintaining high efficiency for CoC is important. For this reason, we propose the pipelining mechanism.

In the pipelining mechanism, we parallelize the production of blocks with the execution of CoC. While each shard i waits for other shards to conduct CoC for it, the shard continues to *produce blocks optimistically.* Specifically, after a shard produces a block of height h, the block needs to wait for CoC to be finalized. Meanwhile, the shard keeps producing new blocks optimistically while waiting for other shards to reach CoC for its block of height h (as illustrated in Figure 2, Shard i keeps producing blocks while waiting for CoC). As a concern of security, we further propose the following designs.

Designs on Security. For security, when shard i is detected as corrupted, the blocks produced by the shard during the waiting process are considered invalid. The state of that shard will be rolled back to the latest one finalized by CoC. However, there are not only intra-shard transactions but also cross-shard transactions in blockchain sharding systems. Therefore, the rollback of a shard state usually involves several other shards, which greatly compromises the security and efficiency of the system.

To *prevent rollback behavior involving multiple shards*, we propose a mechanism for *pessimistically sending cross-shard transactions.* Only when a block within a shard is finalized by the CoC, the cross-shard transactions contained in that block are then sent to other shards along with the proof (i.e., the `Finalized` messages).

Note that optimistically producing blocks involves *processing transactions as well as conducting CoC for other shards.* Therefore, even if the state of a corrupted shard is rolled back, the blocks produced during this period are kept as orphan blocks. The reason behind such design is to leave a history of it conducting CoC for other shards, i.e., the information about each shard's participation in CoC is not rolled back.

E. Discussion of CoC Protocol

In this part, we discuss and show that our CoC protocol is feasible in practice. First, we analyze that our CoC protocol does *not increase too much cross-shard communication overhead* and still has superior performance (see Section VII-C). Since the messages transmitted across shards during CoC are meta messages (signed headers or voting messages), they only account for little communication overhead (typically few hundred bytes). Additionally, for security, we sacrifice some communication efficiency by requiring all nodes in a shard to send cross-shard communication, as in many previous works [15], [16], [33], [39]. We make it our future work to reduce the number of nodes sending cross-shard communications while maintaining security, as in [18]. Second, there are *not too many corrupted shards* in the system that need to be replaced in each epoch. We verify this through extensive experiments (see Section VII-B). Third, since each shard stores only a subset of the whole state, it is *efficient to transmit state information between shards*, as verified in Section VII-D through experiments. In fact, transmitting state between shards is extremely common in blockchain sharding (during shard reconfiguration), and we apply the idea of state transmission to replace corrupted shards. Moreover, the system can further reduce the overhead of state storage by using techniques such as state pruning [17], [39] for better feasibility.

VI. Security Analysis

A. Epoch Security

We now bound the failure probability for the whole system during each epoch. We first analyze the probability for a single shard under different cases (e.g., $f < 1/3$, $1/3 \leq f < 2/3$, $f \geq 2/3$) in each epoch, as the nodes will be reshuffled before each new epoch during the shard reconfiguration. Similar to previous research [8], [15], [39], we use the hypergeometric distribution function to calculate those probabilities. Specifically, let X be a random variable representing the number of Byzantine nodes assigned to a shard of size $n = N/S$. The probability for a shard to be uncorrupted in each epoch can be thus computed by:

$$p_{(f<1/3)} = \sum_{x=0}^{\lfloor n/3 \rfloor - 1} \frac{\binom{FN}{x}\binom{N-FN}{n-x}}{\binom{N}{n}}. \tag{1}$$

The probability for a shard to have $1/3 \leq f < 2/3$ and $f \geq 2/3$ fraction of malicious nodes in each epoch can be thus computed by:

$$p_{(1/3\leq f<2/3)} = \sum_{x=\lfloor n/3 \rfloor}^{\lfloor 2n/3 \rfloor - 1} \frac{\binom{FN}{x}\binom{N-FN}{n-x}}{\binom{N}{n}} \tag{2}$$

and

$$p_{(f\geq 2/3)} = \sum_{x=\lfloor 2n/3 \rfloor}^{n} \frac{\binom{FN}{x}\binom{N-FN}{n-x}}{\binom{N}{n}} \tag{3}$$

respectively.

Based on the above probabilities, we then analyze the failure probability of the system within an epoch. CoChain will fail in two cases: 1), There exists shards with $\geq 2/3$ fraction of malicious nodes. 2), The corruption ratio in each shard is $<$ $2/3$, yet the system still cannot be recovered after CoC. That is, there exists CoC groups whose corrupted shards is $\geq m/3$. In this case, the CoC is not secure, which may finally lead to the system failure.

When calculating the upper bound failure probability, we make the same assumption as the previous studies that the failure probability of each shard is independent of each other [15], [19], [39]. We first calculate the union bound to bound the failure probability when there exist shards with $\geq 2/3$ fraction of malicious nodes, $p_{(\exists f \geq 2/3\, and\, fail)}$. Under this case (the first case), the system will definitely fail. Thus, similar to previous studies [8], [15], [39], the equation is as follow:

$$p_{(\exists f\geq 2/3\, and\, fail)} = 1 \cdot S \cdot p_{(f\geq 2/3)}. \tag{4}$$

We now bound the probability when $f < 2/3$ in each shard and the system cannot be safely recovered after CoC (the second case):

$$p_{(\exists f<2/3\, and\, fail)} = p_{(fail\, when\, f<2/3)} \cdot p_{(f<2/3)}. \quad (5)$$

Here we calculate $p_{(fail\, when\, \exists f<2/3)}$ (the probability that the system cannot be safely recovered after CoC under the condition that $f < 2/3$ in each shard) as the upper bound, since obviously $p_{(fail\, when\, \exists f<2/3)} \geq p_{(\exists f<2/3\, and\, fail)}$. Let Y and Z be random variables representing the number of corrupted shards in the system and in one CoC group, respectively. To calculate $p_{(fail\, when\, \exists f<2/3)}$, we first estimate the probability that the system has $Y = y$ corrupted shards in the case that $f < 2/3$ in each shard, denoted as $p_{(Y=y)}$:

$$p_{(Y=y)} = \binom{S}{y}\left(\frac{p_{(1/3\leq f<2/3)}}{1-p_{(f\geq 2/3)}}\right)^{y} \cdot \left(\frac{p_{(f<1/3)}}{1-p_{(f\geq 2/3)}}\right)^{(S-y)}. \quad (6)$$

We then compute the union bound to bound the probability that the CoC cannot safely reach consensus ($z \geq \lfloor m/3 \rfloor$) given that the system has y corrupted shards, $p_{(fail\, when\, Y=y)}$:

$$p_{(fail\, when\, Y=y)} = S \cdot \sum_{z=\lfloor m/3 \rfloor}^{m} \frac{\binom{y}{z}\binom{S-y}{m-z}}{\binom{S}{m}}. \quad (7)$$

Based on Equation 6 and 7, we can derive $p_{(fail\, when\, \exists f<2/3)}$:

$$p_{(fail|\exists X<\lfloor 2n/3 \rfloor)} = \sum_{y=0}^{S} p_{(Y=y)} \cdot p_{(fail\, when\, Y=y)}. \quad (8)$$

Combining the probabilities under the above two cases (Equation 4 and 8), we can finally derive an upper bound failure probability of CoChain within each epoch as follow:

$$p_{(fail)} = p_{(\exists f\geq 2/3\, and\, fail)} + p_{(fail\, when\, \exists f<2/3)}. \quad (9)$$

We empirically verify in Section VII-E that our estimated theoretical upper bound failure probability can bound the true failure probability. Moreover, by adjusting the shard size n and CoC size m in different system scales, we can limit the upper bound failure probability of the system to be negligible, the specific values of the probability are shown in Section VII-B.

B. Protocol Security Analysis

Under negligible epoch failure probability, we now analyze the security of our main designs.

Theorem 1. *For any shard i, if there are less than* $2/3$ *fraction of malicious nodes (*$f < 2/3$*), then the honest shards in the CoC group who monitors it (*C_i*) can detect typical attacks.*

Proof. As explained in Section V-B, for any corrupted shard with $f < 2/3$, attackers can still fork a shard chain by generating multiple valid blocks with the same height. For any honest shard that monitor shard i, it can determine whether such attacks have occurred via the block header. Since each block header is sent by all the nodes in shard i and there are always honest nodes in it, the header always can be sent by the honest nodes. For silence attacks, we adopt a timeout scheme to ensure liveness, as mentioned in V-B. □

Theorem 2. *For any shard i, if* $f < 2/3$*, and there are less than* $1/3$ *fraction of corrupted shards in CoC group* C_i*, then our CoC protocol is secure.*

Proof. In CoC, the shards in C_i reach cross-shard BFT consensus on the consensus results of shard i. The intuition is to analogize the shards in CoC to the nodes in PBFT consensus. Specifically, when shard i is uncorrupted, the shards in C_i will execute the Pre-prepare, Prepare, and Commit phase accordingly. Since there are $< 1/3$ fraction of corrupted shards in C_i, those shards can reach a consensus and finalize the block for shard i. When more than $2m/3$ `Complain` messages from different shards in C_i are generated, then shard i is detected as corrupted (the security of the cross-shard verification is proved in Theorem 1). Because $< 1/3$ fraction of shards in C_i are corrupted. The protocol can then safely move to the shard replacement phase.

□

Theorem 3. *In CoChain, our shard replacement mechanism can replace corrupted shards, thus recovering the safety and liveness of the system.*

Proof. First, when the shards participating in CoC group C_i receive more than $2m/3$ of the `Complain` messages, they determine that shard i is corrupted and stop conducting subsequent CoC for shard i. Therefore, even if the corrupted shard i still performs subsequent consensus, no one will recognize its subsequent consensus results since its intra-shard consensus results are not certified by CoC (no `Finalized` message). Second, for any shard i, since there are always honest nodes and the cross-shard communication is sent by each node, valid state information can always be routed to the shard responsible for replacing shard i. The successor shard can use the proof information to verify the validity of the sent state and take over from shard i for subsequent consensus. Finally, if the shard responsible for taking over is also a corrupted shard, it can still be taken over by other shards. □

VII. Evaluation

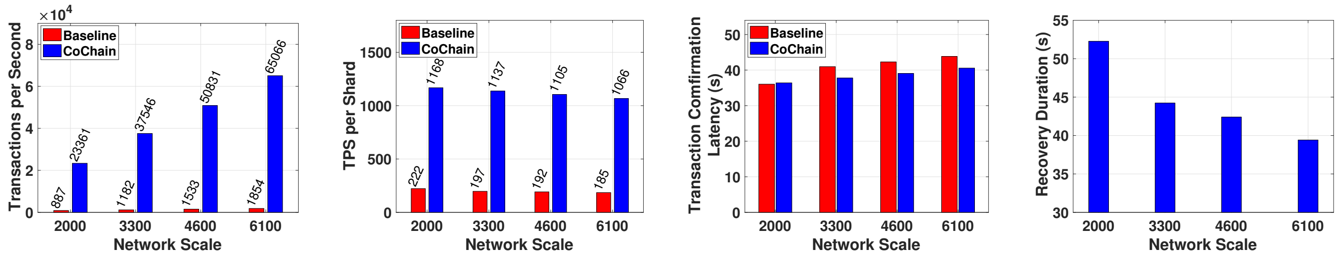
A. Experimental Setup

We implement a prototype of CoChain in Golang based on Harmony [33], a well-known permissionless blockchain sharding project. For fair comparison, we choose Harmony as the baseline protocol to compare the performance with CoChain. The main protocols in CoChain can be easily applied to most existing sharding systems for improved performance.

We simulate a large-scale network of up to 6,100 nodes by oversubscribing up to 61 Amazon EC2 instances. Each instance has a 96-core processor and a 25-Gbps communication link. Like existing research, we consider a latency of 100 ms for every message. The bandwidth for each node is set as 50 Mbps. We set each transaction size to 512 bytes, and each complete block can contain up to 4,096 transactions. The fraction of total malicious nodes F is set as 1/4.

B. Choice of Parameters

Choice of Shard Size and CoC Size. The shard size and the CoC size should be adjusted to limit the system failure probability to be negligible, as described in Section VI-A. We choose to adjust the shard and CoC size to bound the



(a) Total TPS comparison (b) Per-shard TPS comparison (c) Confirmation latency comparison (d) Time duration for system recovery

Fig. 3: Main performance results under various network scales.

TABLE I: Choice of parameters in baseline and CoChain.

Network Scale	2000	3300	4600	6100
# of Shards in Baseline	4	6	8	10
Shard Size in Baseline	500	550	575	610
Failure Probability in Baseline ($\cdot 10^{-6}$)	2.4	3.8	6.6	5.0
# of Shards in CoChain	20	33	46	61
Shard Size in CoChain	100	100	100	100
CoC Size in CoChain	15	18	21	24
Failure Probability in CoChain ($\cdot 10^{-6}$)	6.1	6.5	2.4	6.9
Avg. # of Failed Shards	0.5	0.8	1.2	1.6

failure probability to be less than $2^{-17} \approx 7.6 \cdot 10^{-6}$ [15]. This probability guarantees that one failure will occur in about 359 years if the system reconfigures in one-day epochs. We determine the shard size and CoC size in CoChain based on Equation 9. The shard size in the baseline protocol is determined based on the classical equation used in [15], as they do not have the CoC protocol.

As shown in Table I, the choice of shard size and CoC size makes the failure probability in both CoChain and the baseline less than $7.6 \cdot 10^{-6}$, ensuring security. In addition, the results show that traditional sharding systems require large shard sizes to secure the system at each epoch. This is confirmed in previous studies [17]. The results also show that CoChain reduces the shard size significantly due to the CoC protocol, which allows CoChain to have more shards for the same network scale as well.

Reasoning of Shard Size in CoChain. In our experiments, we choose the shard size of 100 for CoChain, as some previous studies have also set their shard sizes to around 100 [8], [15]–[17], [39], which is practical. However, as mentioned in Section II-B, previous solutions have various limitations when reducing the shard size.

The shard size in CoChain is adjustable under the same network scale. This is because in CoChain, the shard size and the CoC size together determine the system failure probability. Therefore, one can choose a smaller shard size (with a correspondingly larger CoC size) in CoChain to achieve potentially higher concurrency while maintaining the same failure probability. This brings better flexibility to CoChain.

Number of Failed Shards. We need to choose the appropriate number of corrupted shards for experiments. We conduct 10 million simulations of the shard reconfiguration results and compute the average number of corrupted shards. As shown in Table I, we use this average value rounded upward as the number of corrupted shards in the following experiments. Moreover, the results show that there are few corrupted shards in the system, so our shard replacement mechanism does not impact the system performance too much.

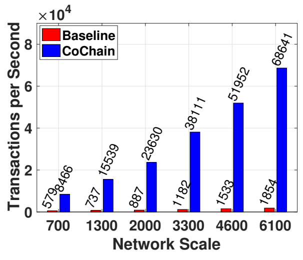
C. Throughput and Latency

We first compare the average throughput (TPS) of CoChain (including throughput during recovery) and the baseline protocol at different network scales. As shown in Figure 3a, CoChain improves throughput by up to 35 times over baseline in a network with 6100 nodes. The reason is mainly that there are more shards in CoChain and the shard size is smaller. Figure 3b shows the throughput for a single shard. The results show that CoChain also achieves more than 5 times of throughput per shard compared to the baseline. This is mainly due to the smaller shard size in our system. Our pipelining mechanism also helps with the performance improvement.

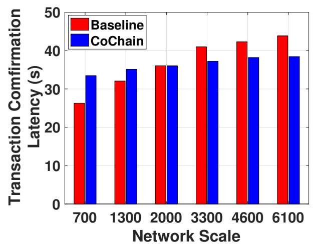
Figure 3c shows the comparative results of the transaction confirmation latency (the latency between the time that a transaction starts to be processed until the transaction is finalized, similar to previous works [15], [39]). CoChain and the baseline protocol have similar latency. *Compared to the huge improvement in throughput, CoChain does not improve the latency significantly.* The main reason is that, in CoChain, even though the reduced shard size speeds up the intra-shard consensus, each transaction needs to wait for a round of CoC to finish before it can be finalized, and the cross-shard transactions need to wait longer. The baseline has such a long latency due to its large shard size, causing slow consensus speed. We make it our future work to further reduce the latency for CoChain. It is also shown that the latency of the baseline grows faster as the network size increases. This is mainly caused by the rapid increase in its shard size.

D. Recovery Duration

We evaluate the time required to recover the system by replacing corrupted shards in CoChain after replaying 1 million Ethereum transactions [41]. As shown in Figure 3d, since each shard in the sharding system only stores a subset of the whole

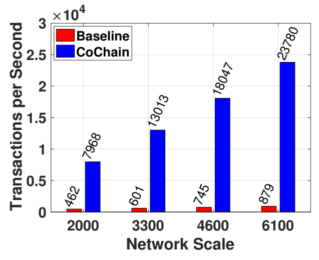


(a) Total TPS comparison

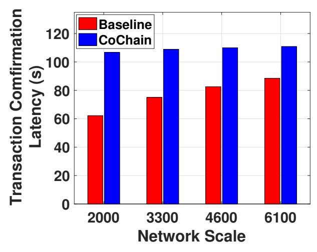


(b) Latency comparison

Fig. 4: Performance under empirical failure probability.



(a) Total TPS comparison



(b) Latency comparison

Fig. 5: Performance under practical environments.

state, the corrupted shard can be replaced in a few tens of seconds. Moreover, as the number of shards in the system increases, the amount of state information stored in each shard is reduced. Therefore, the time required to recover the system is also reduced.

TABLE II: Settings under empirical failure probability.

Network Scale	700	1300	2000	3300	4600	6100
# of Shards in Baseline	2	3	4	6	8	10
# of Shards in CoChain	7	13	20	33	46	61
Reduced CoC Size	6	9	12	15	18	18
Empirical Failure Probability ($\cdot 10^{-6}$)	1.0	5.0	2.5	3.7	1.3	3.3

E. Performance under Empirical Failure Probability

We now evaluate the performance of CoChain under the empirical failure probability. Since the theoretical failure probability is an upper bound, the real failure probability may be smaller than the theoretical computed value. Therefore, by bounding the empirical failure probability to below $7.6 \cdot 10^{-6}$, CoChain can achieve better performance. Specifically, we adjust the shard size and CoC size and repeat 10 million simulations of the shard reconfiguration process to calculate the empirical failure probability, and the results are shown in Table II.

The results show the following points: First, the CoC size can be reduced under the empirical failure probability. Second, we obtain smaller failure probabilities than the theoretical failure probability with smaller CoC sizes (compared with Table I). This implicitly confirms that the theoretically derived failure probability bounds the true failure probability, since it is obvious that an increase in the CoC size leads to a decrease in the failure probability. Therefore, with same CoC size, the real failure probability will be much smaller than the theoretical failure probability upper bound. Third, as the size of the CoC is reduced, CoChain is able to run in smaller network scales with empirical failure probability.

The comparison results for throughput and latency are shown in Figure 4. CoChain improves throughput by up to 37 times over baseline with 6100 nodes. Moreover, compared to baseline, CoChain reduces the latency by 14% with 6100 nodes. However, CoChain achieves an increased latency of $1.27\times$ in a small-scale network with 700 nodes. The reasons for the results are similar to those in Section VII-C.

F. Performance under Practical Environments

To better simulate the geographically-distributed large-scale network environment, we randomize the delay for each message between different nodes (1-1000ms, which is practical [13]). The throughput and latency comparison results are shown in Figure 5. The performance of both CoChain and baseline protocols gets degraded (compared with Figure 3a and 3c) due to the randomly increased message delay between nodes. CoChain improves throughput by up to 27 times over baseline with 6100 nodes. However, the latency in CoChain is longer than that in the baseline protocol. We speculate the main reason is that the increased message delay has a larger impact on CoChain: it decreases both the speed of intra-shard consensus and the speed of reaching CoC. However, it only reduce the speed of reaching intra-shard consensus in baseline.

VIII. Conclusions

In this paper, we propose CoChain, a high concurrency blockchain sharding system. In CoChain, individual shards monitor each other's consensus results and recover the system by shard replacement when a corrupted shard is found. To ensure the security and efficiency of cross-shard monitoring and replacement, we propose a cross-shard Consensus on Consensus (CoC) protocol and elaborate the designs of each part in the protocol. Based on the intuition of corruption-and-recovery and the above designs, CoChain securely configures small shard sizes and significantly enhances the system concurrency. Finally, we implement CoChain and conduct large-scale experiments to verify the superiority of CoChain. Empirical evaluation shows that CoChain achieves tens of times higher throughput compared to the advanced baseline protocol.

Acknowledgement

This work was supported by National Natural Science Foundation of China (Grant No. 61701216), Guangdong Provincial Key Laboratory (Grant No. 2020B121201001), Guangdong Innovative and Entrepreneurial Research Team Program (Grant No. 2016ZT06G587), Shenzhen Sci-Tech Fund (Grant No. KYTDPT20181011104007, JCYJ20180507181527806), and RGC GRF Grant 16202121.

References

[1] M. J. Amiri, D. Agrawal, and A. El Abbadi. Sharper: Sharding permissioned blockchains over network clusters. In *Proceedings of the 2021 International Conference on Management of Data*, pages 76–88, 2021.

[2] P. Barrett. Zilliqa technical whitepaper, 2017.

[3] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788. Springer, 2018.

[4] Y. Buchnik and R. Friedman. Fireledger: a high throughput blockchain consensus protocol. *arXiv preprint arXiv:1901.03279*, 2019.

[5] M. Castro, B. Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, volume 99, pages 173–186, 1999.

[6] T. Crain, C. Natoli, and V. Gramoli. Red belly: a secure, fair and scalable open blockchain. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 466–483. IEEE, 2021.

[7] R. Dahlberg, T. Pulls, and R. Peeters. Efficient sparse merkle trees. In *Nordic Conference on Secure IT Systems*, pages 199–215. Springer, 2016.

[8] H. Dang, T. T. A. Dinh, D. Loghin, E.-C. Chang, Q. Lin, and B. C. Ooi. Towards scaling blockchain systems via sharding. In *Proceedings of the 2019 international conference on management of data*, pages 123–140, 2019.

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

[10] B. David, B. Magri, C. Matt, J. B. Nielsen, and D. Tschudi. Gearbox: Optimal-size shard committees by leveraging the safety-liveness dichotomy. 2022.

[11] H. Duan, J. Li, S. Fan, Z. Lin, X. Wu, and W. Cai. Metaverse for social good: A university campus prototype. In *Proceedings of the 29th ACM International Conference on Multimedia*, pages 153–161, 2021.

[12] S. Gupta, J. Hellings, and M. Sadoghi. Rcc: Resilient concurrent consensus for high-throughput secure transaction processing. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 1392–1403. IEEE, 2021.

[13] S. Gupta, S. Rahnama, J. Hellings, and M. Sadoghi. Resilientdb: Global scale resilient blockchain fabric. *arXiv preprint arXiv:2002.00160*, 2020.

[14] J. Hellings and M. Sadoghi. Byshard: Sharding in a byzantine environment. *Proceedings of the VLDB Endowment*, 14(11):2230–2243, 2021.

[15] Z. Hong, S. Guo, P. Li, and W. Chen. Pyramid: A layered sharding blockchain system. In *IEEE INFOCOM 2021-IEEE Conference on Computer Communications*. IEEE, 2021.

[16] C. Huang, Z. Wang, H. Chen, Q. Hu, Q. Zhang, W. Wang, and X. Guan. Repchain: A reputation-based secure, fast, and high incentive blockchain system via sharding. *IEEE Internet of Things Journal*, 8(6):4291–4304, 2020.

[17] 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)*, pages 583–598. IEEE, 2018.

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

[19] Y. Liu, J. Liu, M. A. V. Salles, Z. Zhang, T. Li, B. Hu, F. Henglein, and R. Lu. Building blocks of sharding blockchain systems: Concepts, approaches, and open problems. *arXiv preprint arXiv:2102.13364*, 2021.

[20] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena. A secure sharding protocol for open blockchains. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 17–30, 2016.

[21] D. Malkhi, K. Nayak, and L. Ren. Flexible byzantine fault tolerance. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 1041–1053, 2019.

[22] P. Maymounkov and D. Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. In *International Workshop on Peer-to-Peer Systems*, pages 53–65. Springer, 2002.

[23] S. Micali, M. Rabin, and S. Vadhan. Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.

[24] S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. *Decentralized Business Review*, page 21260, 2008.

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

[26] R. Pass and E. Shi. Hybrid consensus: Efficient consensus in the permissionless model. *Cryptology ePrint Archive*, 2016.

[27] G. Pîrlea, A. Kumar, and I. Sergey. Practical smart contract sharding with ownership and commutativity analysis. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, pages 1327–1341, 2021.

[28] R. Rana, S. Kannan, D. Tse, and P. Viswanath. Free2shard: Adversary-resistant distributed resource allocation for blockchains. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 6(1):1–38, 2022.

[29] A. Ranchal-Pedrosa and V. Gramoli. Blockchain is dead, long live blockchain! accountable state machine replication for longlasting blockchain. *arXiv preprint arXiv:2007.10541*, 2020.

[30] P. Ruan, T. T. A. Dinh, D. Loghin, M. Zhang, G. Chen, Q. Lin, and B. C. Ooi. Blockchains vs. distributed databases: Dichotomy and fusion. In *Proceedings of the 2021 International Conference on Management of Data*, pages 1504–1517, 2021.

[31] M. Sabt, M. Achemlal, and A. Bouabdallah. Trusted execution environment: what it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, volume 1, pages 57–64. IEEE, 2015.

[32] A. Skidanov and I. Polosukhin. Nightshade: Near protocol sharding design. *URL: https://nearprotocol. com/downloads/Nightshade. pdf*, page 39, 2019.

[33] H. Team. Harmony: Technical whitepaper, 2018.

[34] D. Vujičić, D. Jagodić, and S. Ranđić. Blockchain technology, bitcoin, and ethereum: A brief overview. In *2018 17th international symposium infoteh-jahorina (infoteh)*, pages 1–6. IEEE, 2018.

[35] J. Wang and H. Wang. Monoxide: Scale out blockchains with asynchronous consensus zones. In *16th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 19)*, pages 95–112, 2019.

[36] G. Wood et al. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper*, 151(2014):1–32, 2014.

[37] Z. Xiang, D. Malkhi, K. Nayak, and L. Ren. Strengthened fault tolerance in byzantine fault tolerant replication. *arXiv preprint arXiv:2101.03715*, 2021.

[38] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 347–356, 2019.

[39] M. Zamani, M. Movahedi, and M. Raykova. Rapidchain: Scaling blockchain via full sharding. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 931–948, 2018.

[40] M. Zhang, J. Li, Z. Chen, H. Chen, and X. Deng. Cycledger: A scalable and secure parallel protocol for distributed ledger via sharding. In *2020 IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 358–367. IEEE, 2020.

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