Porygon: Scaling Blockchain via 3D Parallelism

Wuhui Chen
Sun Yat-sen University
chenwuh@mail.sysu.edu.cn

Ding Xia
Sun Yat-sen University
xiad23@mail2.sysu.edu.cn

Zhongteng Cai
The Ohio State University
cai.1125@osu.edu

Hong-Ning Dai
Hong Kong Baptist University
henrydai@hkbu.edu.hk

Jianting Zhang
Purdue University
zhan4674@purdue.edu

Zicong Hong
The Hong Kong Polytechnic University
zicong.hong@connect.polyu.hk

Junyuan Liang
Sun Yat-sen University
liangjy53@mail2.sysu.edu.cn

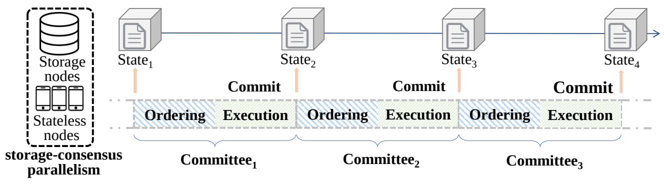
Zibin Zheng
Sun Yat-sen University
zhzibin@mail.sysu.edu.cn

***Abstract*—Recently, stateless blockchains have been proposed to alleviate the storage overhead for nodes. A stateless blockchain achieves storage-consensus parallelism, where storage workloads are offloaded from on-chain consensus, enabling more resource-constraint nodes to participate in the consensus. However, existing stateless blockchains still suffer from limited throughput.**

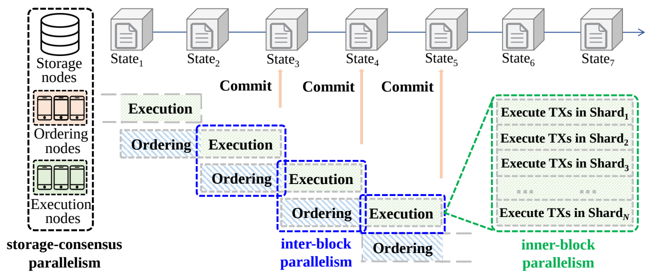
In this paper, we present Porygon, a novel stateless blockchain with three-dimensional (3D) parallelism. First, Porygon separates the storage and consensus of transactions as the stateless blockchain, achieving the storage-consensus parallelism. This first-dimensional parallelism divides the processing of transactions into several stages and scales the network by supporting more nodes in the system. Based on such a design, we then propose a pipeline mechanism to achieve second-dimensional inter-block parallelism, where relevant stages of processing transactions are pipelined efficiently, thereby reducing transaction latency. Finally, Porygon presents a sharding mechanism to achieve third-dimensional inner-block parallelism. By sharding the executions of transactions of a block and adopting a lightweight cross-shard coordination mechanism, Porygon can effectively execute both intra-shard and cross-shard transactions, consequently achieving outstanding transaction throughput. We evaluate the performance of Porygon by extensive experiments on an implemented prototype and large-scale simulations. Compared with existing blockchains, Porygon boosts throughput by up to 20×, reduces network usage by more than 50%, and simultaneously requires only 5MB of storage consumption per node.

I. Introduction

Blockchain, as a disruptive technology enabling the establishment of distributed, transparent, and secure ledgers, has witnessed progressive adoption across diverse sectors including trading systems [1], supply chains [2], healthcare [3], and business databases [4], [5], [6], along with its original application in cryptocurrencies [7]. Unfortunately, participants in traditional blockchain networks confront challenges posed by the escalated data volume. To participate in the consensus of a blockchain, a node must spend a long time synchronizing the complete blockchain state, associated with a large storage requirement. For instance, the data volume of account-based Ethereum had reached the terabyte scale [8]. This impedes the engagement of ordinary individuals using mobile devices with constrained storage and network resources in a blockchain system, limiting network scale and compromising the decentralization of the system.



(a) 1D parallelism: Each committee processes transactions round by round in a sequential manner.



(b) 3D parallelism (Porygon): Diverse committees concurrently process transactions through inter-block and inner-block parallelisms.

Fig. 1: Blockchain architectures: (a) and (b) depict the transaction processing workflows of stateless blockchain by 1D parallelism and 3D parallelisms, respectively.

In this context, *stateless blockchain* has been proposed to create a new transaction-committing protocol to alleviate the storage burden on nodes [9], [10], [11], [12], [8], [13]. This stateless mechanism can be conceptualized as establishing parallelism between on-chain *stateless nodes* and off-chain *storage nodes*. Storage nodes are responsible for storing states and providing data available for downloading when required, while stateless nodes run a consensus protocol for ordering and executing transactions with the relevant states derived from storage nodes. By parallelizing storage and consensus tasks, stateless blockchains reduce the storage overhead for resource-constraint stateless nodes and thus enable more participants in the network. Figure 1(a) provides a simplified depiction of this one-dimensional (1D) parallelized stateless blockchain. For illustration, here we simplify the procedure of committing transactions as the Ordering stage and Execution stage, while the complete procedure always involves more stages that will be detailed in § II-A.

Although allowing more resource-constraint nodes to engage in the system, current stateless blockchains still cannot be applied to a realistic scenario with thousands of real-time requests due to their limited performance. For instance, the state-of-the-art stateless blockchains [8], [11] can only process around 1,000 Transactions Per Second (TPS) while a common payment scenario, e.g., Visa, requires reaching 20,000 TPS. We observe that such poor performance arises from their two inherent factors: *sequential transaction processing* and *underutilized computational resources* (see § II-A for more details). Briefly speaking, since storage and consensus of transactions are separated and completed by different kinds of nodes, the existing stateless blockchains only allow a small group of stateless nodes to serially process transactions to ensure strong consistency among all nodes.

In this paper, we revisit and redesign the stateless blockchain to enhance its parallelism in processing transactions, thereby achieving an incredible improvement in performance while keeping its capability of supporting large-scale nodes. Specifically, we propose a new stateless blockchain system in the permissionless setting, called Porygon. As shown in Figure 1(b), Porygon introduces a three-dimensional (3D) parallelism based on the one-dimensional parallelism built by the original stateless blockchain. (1) Porygon builds on the stateless blockchain's 1D parallelism [11], [8] by separating data storage from consensus, enhancing network scalability by allowing more nodes to participate. (2) It introduces a new pipeline mechanism for the second dimension, namely *inter-block parallelism*, to construct *Ordering and Execution Committees* for processing transactions in parallel stages, thus improving throughput and reducing latency. (3) Lastly, Porygon implements sharding for the third dimension, namely inner-block parallelism, to divide transactions into sets for concurrent execution by *Execution Sub-Committees*, thereby optimizing computational efficiency.

However, it is non-trivial to implement such a three-dimensional parallelism due to the following three challenges.

Challenge 1: Ordering Committee Bottleneck. Due to the multi-dimensional parallelism, multiple Execution Sub-Committees may concurrently execute transactions in each round. As a result, the Ordering Committee needs to package a larger number of transactions and broadcast the complete block within the committee. The Ordering Committee's bandwidth can become a bottleneck. To tackle this challenge, we *propose a novel solution* to decouple broadcasting from ordering to better utilize bandwidth resources (§ IV-B). To achieve this, we decouple *transaction blocks* from *proposal blocks*. Proposal blocks only record the list of transaction blocks and hence are small in size. Then, we assign the task of broadcasting transaction blocks to storage nodes, while stateless nodes only need to broadcast smaller proposal blocks.

Challenge 2: Vulnerability to Unavailable Transactions. The pipelining mechanism within the stateless system poses challenges related to data availability and task allocation. Stateless nodes are vulnerable to unavailable transactions fabricated by malicious storage nodes, meaning that they can only download indices of transactions but cannot download transaction contents. However, requiring all committees to download complete transactions before execution to avoid such attacks will incur redundant overheads. To address this challenge, we *propose a solution* to introduce a separate Witness Phase to the pipeline, by assigning this phase to Execution Committees to ensure data availability (§ IV-C). This design eliminates the need for the Ordering Committee to download complete transactions.

Challenge 3: Sharding Coordination Complexity. Implementing the sharding mechanism within the stateless blockchain poses the challenge of cross-shard coordination. Particularly, the concurrent execution of transactions across different shards has to preserve the atomicity of cross-shard transactions and the consistency of blockchain states. To tackle this challenge, we *propose the solution* to designate the Ordering Committee as a coordinator to distribute transactions to different Execution Sub-Committees and aggregate execution results to update the global state. Moreover, Porygon proposes a lightweight mechanism for executing cross-shard transactions (§ IV-D). This mechanism effectively alleviates the transaction execution workload distributed across multiple shards while simultaneously ensuring the atomicity of cross-shard transactions.

In summary, this paper makes the following contributions:

- **The novel architecture of 3D parallelism.** We introduce Porygon, a novel stateless blockchain system that achieves a large network scale, low latency, and high transaction throughput by parallelizing the system from three distinct dimensions.
- **Transaction Processing Pipeline.** We design a fine-grained transaction-processing pipeline to achieve inter-block parallelism. The process of committing transactions is decoupled into sequential phases and then constitutes a pipeline to ensure performance as well as security.
- **Sharded Transaction Execution.** We distribute transactions to different shards for execution to achieve inner-block parallelism. We design a lightweight cross-shard coordination mechanism to ensure the atomicity of cross-shard transactions while preserving outstanding performance.
- **Experiment Evaluation.** We conduct extensive experiments to evaluate Porygon by both experiments on the implemented prototype and large-scale simulations (§ VI). Experimental results show that we can achieve a throughput of 21,090 transactions per second, which is nearly 20$\times$ of state-of-the-art stateless blockchain [11] while reducing network usage by more than 50% and requiring 5MB of storage consumption per node.

II. Background and Related Work

A. Background

The stateless blockchain concept aims to reduce storage burdens on full nodes by offloading state storage to off-chain storage nodes, enabling on-chain stateless nodes to process transactions without holding states [11], [8], [13]. This setup achieves 1D parallelism by allowing storage and stateless nodes to work in together.

Using Blockene as an example [11], illustrated in Figure 1(a), the workflow in a stateless blockchain involves *consensus rounds* for transaction commitment. Initially, a subset of stateless nodes is chosen each round for transaction processing to minimize communication overhead. These nodes, forming a committee, then download transactions from storage nodes and share download proofs to ensure data availability. Then, a block of verified transactions is proposed, and the committee conducts consensus to *order* and *execute* these transactions. Execution involves state and proof downloads from storage nodes for global state updates. Once a consensus on updates is reached, the block is certified and added to the blockchain.

Throughout the procedure of processing transactions, we observe two characteristics that constrain the performance of the existing stateless blockchain system.

Characteristic 1: Sequential Transaction Processing. The selected on-chain stateless nodes need to *engage in all stages of processing transactions in a serialized way*. Given limited bandwidth and computation resources, these stateless nodes will spend a longer time downloading, ordering, and executing transactions, thereby leading to a longer latency for transactions. Meanwhile, lightweight devices are prone to losing connectivity with the network and the prolonged transaction processing increases the likelihood of nodes getting disconnected from the network. This effect negatively impacts the overall system performance. This observation motivates us to establish parallelism in the stages of transaction processing.

Characteristic 2: Underutilized Computational Resources. Existing stateless blockchains assign *only one committee to process transactions per consensus round*. Although this design can easily ensure the consistency of the ledger, it also puts the vast majority of computation capacity untapped, thereby bringing limited improvement of transaction throughput even with a large number of nodes in the blockchain system. For instance, Blockene [11] selects about 2,000 stateless nodes as committee members to process transactions despite millions of available nodes. This observation motivates us to achieve further parallelism in executing transactions.

B. Related Work

Motivated by the above observations, this work targets designing a highly parallelized blockchain that can not only support large-scale nodes to participate in the system but also achieve high performance. Next, we will introduce some promising parallelized schemes for blockchains as well as their shortcomings compared with our solution.

Stateless blockchain system. Stateless blockchains aim to enable nodes to execute transactions without storing cumbersome states. Blockene [11] employs *Citizens* (smartphones with limited resources) and *Politicians* (servers) in its architecture. Single-committee-based citizens agree on blocks and update states, while Politicians only store states. Saguaro [13] designs a mobile consensus protocol to facilitate the participation of mobile devices. SlimChain [8] develops a stateless blockchain with smart contract support. It employs off-chain Trusted Execution Environments (TEEs) for executing smart contracts, but this approach may be exposed to a vulnerable trusted code base (TCB) susceptible to attacks [14], [15], [16]. However, as we discussed before, these works cannot scale performance, making it impractical to be applied to a realistic scenario with thousands of requests per second.

Decoupling blockchain functions. Function decoupling enhances blockchain concurrency and performance, as seen in systems like Hyperledger, BIDL, DispersedLedger, and NeuChain [17], [18], [19], [20]. BFT protocols, including Bullshark and DAG-Rider, realize this by separating transaction and proposal blocks [21], [22]. ResilientDB and RCC focus on consensus parallelization [23], [24], while Proof-of-Execution prioritizes pre-consensus transaction execution [25]. SChain introduces concurrent execution within and across blocks [26], and SEFrame, along with earlier SGX-based efforts, supports concurrent smart contract execution but lacks permissionless environment support [27], [28]. DispersedLedger separates consensus into two phases but faces security challenges in task allocation [19]. Our Porygon innovates with a Witness Phase for enhanced data availability and secured task allocation.

Blockchain sharding system. Sharding enhances blockchain performance by spreading workloads across multiple shards, reducing node overheads [14], [29], [30], [31], [32], [33], [34]. Cross-shard transactions ensure atomicity through various mechanisms: Elastico [35] uses a final committee for global results, OmniLedger [36] involves the client in shard coordination, RapidChain [37] simplifies verification by allowing direct user-committee communications while Byshard [38] applies database techniques to Byzantine environments. Unlike these blockchain-sharing systems, which burden all nodes with equivalent storage demands, Porygon implements a lightweight mechanism, facilitating sharding for stateless nodes, thus accommodating those with limited resources.

III. System and Adversary Model

A. System Model

Porygon is composed of storage nodes and stateless nodes, all of which have unique identities. Stateless nodes are periodically and randomly selected from the network to compose committees. We utilize an account-based model for the blockchain state, which facilitates basic transaction functionalities, such as transfers. Accounts are divided into different shards based on the last N digits of their IDs. Stateless nodes are categorized into two types of committees: the unique Ordering Committee and the multiple Execution Committees. The Ordering Committee groups transactions originating from distinct subsets of accounts. Execution Committees are subdivided into distinct shards, forming Execution Sub-Committees dedicated to the execution of transactions. During transaction execution, each execution node accesses state data from storage nodes. If a transaction only has to access states belonging to the same shard, it is regarded as an *intra-shard transaction*; it is regarded as a *cross-shard transaction* otherwise.

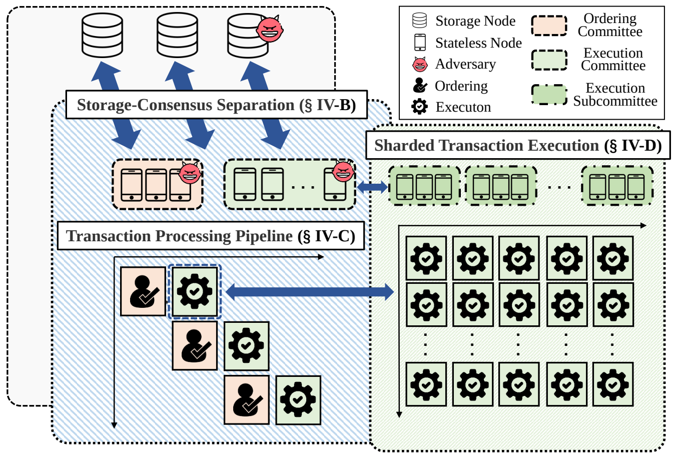

Storage Node
Stateless Node
Adversary
Ordering
Executon
Ordering Committee
Execution Committee
Execution Subcommittee
Storage-Consensus Separation (§ IV-B)
Sharded Transaction Execution (§ IV-D)
Transaction Processing Pipeline (§ IV-C)

Fig. 2: Overview of 3D parallelism in Porygon.

B. Adversary Model and Trust Assumption

We consider *honest* nodes that always follow the system protocol, while *malicious* nodes can deviate from the protocol in arbitrary ways. First, *malicious storage nodes* can discard messages, which need to be routed between stateless nodes or decline to broadcast locally received transactions to other storage nodes. Second, *malicious stateless nodes* can equivocate during the consensus. However, similar to previous stateless blockchains [11], [8], malicious nodes cannot forge identities or digital signatures. We assume that the adversary can control $\alpha = 1/4$ of stateless nodes and $\beta = 1/2$ of storage nodes. The security analysis under this assumption will be present in § V.

IV. Porygon Design

A. System Overview

Our Porygon achieves 3D parallelism through the following methods: *storage-consensus separation*, *transaction processing pipeline*, and *sharded transaction execution*, as shown in Figure 2. Firstly, it separates storage from consensus for parallel processing by utilizing i) stateless nodes for ordering and execution, and ii) storage nodes for data-keeping. Secondly, Porygon introduces a pipeline for parallel transaction execution, incorporating a Witness Phase for ensuring data availability. Lastly, it employs sharding in execution committees for parallel processing, with an Ordering Committee coordinating cross-shard transactions via a multi-phase commit protocol. Further details will be elaborated in subsequent sections.

B. Storage-Consensus Separation

We introduce the storage-consensus separation architecture in our design and elaborate on the key technical details that support further parallelism.

1) Stateless Architecture: In Porygon, storage and stateless nodes fulfill distinct functions. Storage nodes hold the entire blockchain state and facilitate communications among stateless nodes, which manage to compact proposal blocks and necessary public keys for consensus participation. After processing, stateless nodes remove transactions to conserve space. Each stateless node establishes connections with a randomly selected subset of storage nodes. These connections

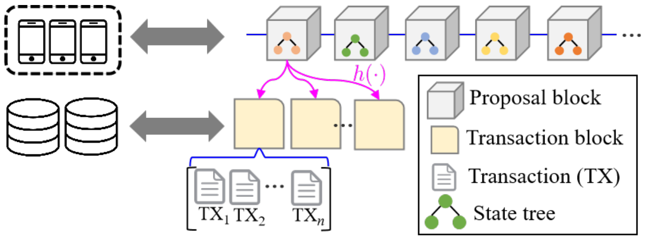


Fig. 3: Block structures in Porygon, including proposal block and transaction block.

enable stateless nodes to send proposal blocks, voting information, and other critical messages, which are then broadcasted among storage nodes for comprehensive network dissemination. Additionally, stateless nodes can request blockchain information or access messages from other stateless nodes through these connections, facilitating both data retrieval and message routing within the system.

To address the potential threat posed by malicious storage nodes, our protocol mandates stateless nodes to connect to multiple storage nodes. This redundancy ensures that the presence of at least one honest storage node is sufficient for the accurate transmission of messages [11]. This resilience is further confirmed by the analysis presented in § V (Lemma 1).

Committees are periodically selected from stateless nodes to process transactions [39]. Each message generated within a committee can be verified by both storage nodes and stateless nodes even if the lifecycle of this committee has ended. The uniqueness of the identification of each node can be enabled by trusted hardware (e.g., leveraging TrustZone in smartphones [40], [41], [11]).

2) Block Structure: To reduce transaction broadcast load, we introduce *proposal blocks* and *transaction blocks* for functional decoupling, as shown in Figure 3. Storage nodes create transaction blocks from users' submissions and broadcast them to stateless nodes. These blocks contain transaction indices and accessed states, which are pre-recorded using software tools for concurrency [42], [26]. Proposal blocks, formed by the Ordering Committee, chain these transaction blocks via backward hash links, by including committee membership information, transaction block lists, and the state tree root. Ordering nodes store only the *proposal block* to minimize storage needs, thereby simplifying consensus broadcasting and easing the Ordering Committee's load.

3) Committee Formation: Porygon periodically selects a set of stateless nodes from a stateless-node pool through *Verifiable Random Function* (VRF) [43] to form committees and process transactions. Specifically, each stateless node generates a random number locally once a round (i.e., each time the newest proposal block is generated) through VRF. The inputs of VRF include the hash value of the latest proposal block and the public key of the stateless node. In this way, it is nearly infeasible for adversaries to generate a biased VRF value in advance [39].

Stateless nodes are selected for the Ordering Committee based on their VRF-generated numbers. Particularly, those generating the smallest numbers form the committee, while the rest are allocated to Execution Committees. The allocation to 2^N Execution Committees is determined by the last N digits of their VRF values. Our system employs two thresholds: the *execution committee threshold* and the *ordering committee threshold*, which are documented in the latest proposal block. Each stateless node itself assesses its VRF result against these thresholds to ascertain its committee membership.

When forming the Ordering Committee, nodes submit their candidate proposal blocks to storage nodes, which then relay these blocks to all Ordering Committee members. This process ensures that each node is aware of others' proposals. The candidate proposal block that carries the lowest VRF value is deemed to be the valid proposal for that round, with its proposer acting as the round's leader. The Ordering Committee then collaborates to reach a consensus on this leader-proposed block, ensuring that all members agree on the block with the lowest VRF value as the basis for the next consensus round.

C. Transaction Processing Pipeline

We next describe how transactions are processed by different committees in parallel, including one Ordering Committee and multiple Execution Committees. First, we describe the successive phases of processing the same batch of transactions by only one committee (see § IV-C1). Then, we describe how to parallelize these phases by pipelining across different committees (§ IV-C2).

1) Decoupled Phases: Transaction blocks packaged by storage nodes are witnessed, ordered, executed, and finally committed by the committee. Details of these successive phases are as follows.

(a) *Witness Phase*. The Witness Phase starts each time a new proposal block is generated and aims to ensure that transaction blocks required by committee members are available. Otherwise, the system will produce empty blocks if committee members cannot download transaction blocks generated by potentially malicious storage nodes for execution. To preserve data availability, committee members have to download transactions before ordering them. Specifically, committee members require transaction blocks from storage nodes. If a committee member can successfully download all transactions in a transaction block, it generates a *witness proof* on this transaction block, i.e., a signature on the transaction block header, and uploads the proof to storage nodes. Besides this, it has to extract the states involved in each transaction and store them within the transaction block for concurrency control in § IV-D. A transaction block is eligible for the following Ordering Phase only when it has received witness proofs from more than T_w committee members, where T_w is set to be larger than the upper bound of the number of committee members that are deviated from the protocol (e.g., more than 1/2 of the committee members return the same result). Storage nodes gather witness proofs from multiple committee members to prove that its transaction blocks have been witnessed by enough committee members. Moreover, honest storage nodes broadcast their transaction blocks and corresponding witness proofs. Therefore, different committee members can download the same transaction block from different honest storage nodes and check whether it has received enough witness proofs.

(b) *Ordering Phase*. From storage nodes, committee members first download both the headers and witness proofs of all transaction blocks that have passed the Witness Phase and verify their witness proofs. The leader of the committee produces a proposal block containing a list of valid transaction blocks, broadcasts the list to all committee members, and runs the consensus algorithm to reach an agreement via storage nodes. All committee members can verify the eligibility of the ordered transaction blocks. In our implementation, the committee runs the BA$\star$ consensus protocol [39], [11]. The committee finally agrees on a unique proposal block, which gives the order of transactions.

(c) *Execution Phase*. After reaching an agreement on a proposal block, the committee then executes transactions and updates states according to the consensus result. The committee requires related states (i.e., key-value pairs) and corresponding integrity proofs (e.g., Merkle Tree paths) from storage nodes. Transactions are sequentially executed, and all failed transactions (e.g., duplicate transactions and double-spending transactions) are abandoned. Failed transactions are still recorded in the transaction block to preserve integrity. After updating all states, each committee member also calculates the latest Merkle tree root and sends the signed root to all committee members. If more than T_e committee members give the same execution result, then such a result is considered valid and will be recorded on-chain through the Commit Phase, where T_e is chosen to be larger than the number of malicious committee members (e.g., more than 1/3 of the committee members sign the same result).

(d) *Commit Phase*. In this phase, the updated Merkle Tree root is recorded on-chain through a new round of consensus protocol. Each committee member verifies the signed Merkle Tree roots it has received. If a member receives at least T_e consistent roots signed by the current committee members, it then agrees on the root. If no root receives enough votes, the output of the consensus algorithm will be the state tree root recorded in the last proposal block, indicating that the state tree remains unmodified. After a new proposal block is generated through the consensus algorithm and uploaded to storage nodes, storage nodes will update local states accordingly.

2) Pipeline Design: We parallelize the sequential processes mentioned in § IV-C1 through pipelining by one Ordering Committee (OC) and multiple Execution Committees (ECs), which are responsible for different phases to achieve inter-block parallelism. In particular, one OC is responsible for the Ordering Phase and the Commit Phase while ECs are responsible for the Witness Phase and the Execution Phase, alleviating the burden on the Ordering Committee.

We take Figure 4 as an example to elaborate on the transaction processing pipeline. The horizontal axis represents the sequence of *rounds*, and the vertical axis indicates diverse committees: one OC and multiple ECs, each of which is denoted by EC_i $(i = 1, 2, \cdots)$. When a new proposal block is generated by OC, a new round begins. It is depicted in Figure 4

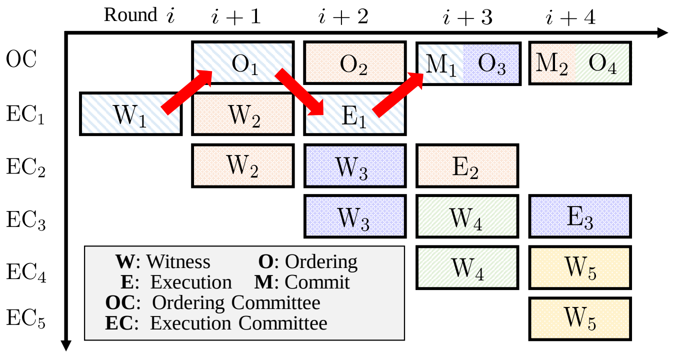


Fig. 4: Transaction processing pipeline.

that each EC lasts for three rounds. There are at most three ECs existing in the same round. We assume that the OC can have a relatively longer life cycle than ECs though the OC can be selected according to a round-robin scheme [14] without affecting the basic design of our pipeline. Each colored square describes the workload of each committee during the specific round. The letter in the square shows the corresponding phase and the following number i indicates that the committee is processing the i-th batch of transactions. We also use diverse colors to differentiate workloads related to different batches of transactions.

The red arrow in Figure 4 shows how the first batch of transactions is processed. The first EC (i.e., EC$_1$) downloads transaction blocks and signs witness proofs (i.e., W$_1$). The OC downloads from storage nodes the list of all transaction blocks that have received enough witness proofs without downloading complete transactions. Members of the OC then verify witness proofs and reach an agreement on the order of these transaction blocks. During the next round (i.e., round $i+2$), the OC sends the agreed list to EC$_1$ for execution. Members of EC$_1$ download the latest states and execute transactions. They do not have to download transactions that they have witnessed during the Witness Phase, thereby saving network usage. EC$_1$ then returns the execution results to the OC. The lifecycle of EC$_1$ then expires. Members of EC$_1$ will join the next round of the committee formation process. During round $(i+3)$, the OC agrees on the updated roots (i.e., M$_1$) and the latest list of transaction blocks (i.e., O$_3$) simultaneously. After the newest proposal block is agreed upon by the OC, storage nodes update local states according to the committed root and corresponding state tree. The first batch of transactions is finally committed to the blockchain ledger.

Cross-Batch Witness. In each round, the bandwidth of nodes in the EC may become a bottleneck, as they are responsible for witnessing all transactions within a block. To reduce the network usage of the Execution Committee in a consensus round, we design the *Cross-Batch Witness* mechanism. For example, during round $(i+1)$ in Figure 4, when the OC is reaching agreement on the order of transaction blocks in the first batch of transactions, EC$_1$ can continue to sign witness proofs for other transaction blocks (e.g., W$_2$). These witnessed transactions, along with the ones witnessed by EC$_2$, are considered as the second batch of transactions. Cross-

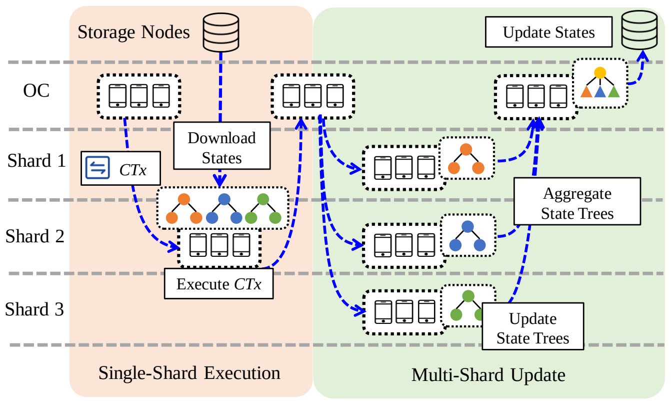


Fig. 5: Cross-shard Coordination.

Batch Witness does not deviate from the goal of the Witness Phase, which is to guarantee that valid transaction blocks can be downloaded by honest stateless nodes. Moreover, OC can verify whether a transaction block has received enough witness proofs in either EC$_1$ or EC$_2$. Generally, transaction blocks witnessed by the i-th EC can be ordered along with the $(i+1)$-th batch of transactions by the OC and can be executed by the $(i+1)$-th EC. Such a design also fills the gap between the Witness Phase and the Execution Phase, hence avoiding the waste of the computation capacity of ECs.

D. Sharded Transaction Execution

To achieve inner-block parallelism, we detail the process through which transactions are executed by Execution Sub-Committees (ESCs) within different shards. Moreover, we describe how stateless shards process cross-shard transactions with low overheads.

1) Parallelized Execution Sub-Committees: In § IV-C2, the system has one OC and three ECs in each round to support pipelining. Nodes in each Execution Committee (EC) are split into Execution Sub-Committees (ESCs) based on the last N digits of VRF numbers, forming shards. User accounts are divided into subsets by their last N digits, with each subset's transactions assigned to one shard for execution. Therefore, each transaction is only sent to one shard. Transactions, including both intra-shard transactions and cross-shard transactions, are ordered by the OC in each round. The OC also acts as the trusted coordinator between different shards to handle cross-shard transactions and perform concurrency control (e.g., impose locks on states that will be accessed by transactions and release locks after execution). After executing transactions, each shard (i.e. ESC) returns the execution results to the OC to update the state tree.

2) Cross-shard Coordination: For cross-shard transactions (CTx), shards coordinate to update states, preserving atomicity and avoiding conflicts, with the OC as the coordinator. Figure 5 shows this coordination process. It involves two phases: *Single-Shard Execution*, where the CTx is executed by one shard only, its result sent to the OC; and *Multi-Shard Update*, where CTx outcomes are distributed from the OC to all involved shards for state updates. Finally, the OC aggregates and commits the updated state trees. We further elaborate on them as follows.

After the Ordering Phase, the OC sends the cross-shard transactions (CTx) to specific shards (e.g., sending CTx to Shard 2 in Figure 5) for *pre-execution* based on where the initiating accounts reside. This approach ensures that the execution result does not modify the state tree until the Multi-Shard Update phase. Before sending CTx, the OC will download states that CTx will access in each shard, which are recorded by analysis tools in advance and stored in witnessed transaction blocks. It checks for potential conflicts between cross-shard transactions across different shards, discarding conflicting transactions to avoid issues while including them in the block for integrity, and notes their indexes. The OC also abandons all transactions submitted in the following rounds having conflicts with previous transactions that have not been committed. Conflicts within the same shard and in the same round do not have to be detected by the OC, since they can be handled by each ESC independently.

Each shard downloads states that are accessed by its transactions and executes these transactions through read-write operations. Some downloaded states may belong to accounts maintained by other shards (e.g., Shard 2 downloads states belonging to all three shards). The OC has avoided conflicts when assigning transactions to different shards. Thus, modifying states that are previously maintained by other shards causes no errors. Moreover, downloading all corresponding states will not lead to redundant overheads since nodes in each shard are stateless. ESCs in all shards finally return the execution results (i.e., the updated key-value pairs) to OC.

(b) *Multi-Shard Update*. During the second phase, the OC verifies whether ESC in each shard returns enough consistent execution results (e.g., more than 1/2 of the ESC members have returned the same result), similar to the workloads mentioned in § IV-C1. If the condition is satisfied, the OC will agree on a list of updated states that need to be updated by each shard. Those states will be recorded in the latest proposal block. The OC then sends it to all shards. Each shard receives the latest proposal block and directly updates states and Merkle Tree paths according to the given proposal block. If all related shards successfully update states and return enough consistent state tree roots, the OC can aggregate these states, calculate the latest state tree root, and finally record it on-chain. Cross-shard transactions are now considered as committed. If some ESCs fail to return enough consistent state tree roots in this round, i.e., some shards fail to commit cross-shard transactions, the OC will continually require the following ESCs of the same shard to update these states until success. The OC will also keep avoiding conflicts between the following transactions and uncommitted transactions. If these transactions still fail to be committed in a specific number of rounds (e.g., two rounds), The OC requires all related shards to roll back by triggering a cross-shard transaction, which updates all related states to its older version.

We now describe the complete process of *committing transactions* with pipelining and sharding. Both cross-shard transactions and intra-shard transactions require multiple phases to be committed although cross-shard transactions have a relatively longer latency than intra-shard ones. Specifically, intra-shard transactions can be committed within four rounds, from the Witness Phase to the final Commit Phase, since they do not require cross-shard coordination. Differently, cross-shard transactions need to be first executed by a single shard and then updated by multiple shards, hence requiring six rounds to be committed. One future work is to deterministically assign priorities to transactions to commit cross-shard transactions before intra-shard transactions.

Take Figure 6 as an example, which depicts the complete process of committing transactions within two shards (i.e., shards A and B). Each ESC is denoted by ESC_i^S, where i denotes the number of the round and S denotes the shard. In round i, the OC outputs a proposal block $\mathcal{B}_i$. We ignore the cross-batch witness and only describe the workloads of ESCs that are initiated from round 1 to round 3 to simplify this description. The arrowed lines between ESCs and the OC describe the complete process of handling the same batch of transactions, from being witnessed to being finally committed. This pipelining process is elaborated as follows.

❶ In round 1, ESC_1^A and ESC_1^B are initialized. They read the latest transaction blocks from storage nodes and sign witness proofs on them. Witnessed transactions are divided into intra-shard transactions and cross-shard transactions denoted by ITx and CTx, respectively. For example, we denote all transaction blocks that are witnessed by ESC_1^A and contain only intra-shard transactions by ITx_1^A. Similarly, those containing cross-shard transactions are denoted by CTx_1^A. Each ESC sends witness proofs to the OC.

❷ In round 2, OC verifies witness proofs and generates a list of ordered transaction blocks denoted by L_2. L_2 contains sub-lists specifically for each shard (e.g., $L_2[A]$ for shard A), which will only be sent to Shard A to save network usage. List L_2 and the latest state tree ($\mathcal{T}_2$) are recorded in proposal block $\mathcal{B}_2$ and are agreed through running the consensus protocol.

❸ In round 3, ESCs execute transactions according to the proposal block $\mathcal{B}_2$, which contains the list L_2 and the state tree $\mathcal{T}_2$ as the block contents. Both the list and the state tree are not completely sent to each shard, e.g., shard A only receives sub-list $L_2[A]$ and downloads subtree $\mathcal{T}_2^A$. $\mathcal{T}_2^A$ contains states required by transactions and their Merkle paths. Since shard A may be designated to execute some cross-shard transactions, $\mathcal{T}_2^A$ may include some states previously maintained by other shards. For intra-shard transactions in ITx, each ESC updates the subtree and output $\mathcal{T}_3^A$. For cross-shard transactions in CTx, each ESC merely returns a set of updated states S_3^A. $\mathcal{T}_3^A$ and S_3^A are returned to OC, and so does the remaining shards. ESCs initialized at round 1 then expire.

❹ In round 4, OC verifies whether enough consistent results $\mathcal{T}_3^d$ and S_3^d are received from each shard d ($d \in \{A, B\}$). Valid S_3^d is handled by the concurrency control function and transferred to the list U_4, indicating which states need to be updated by each shard. The state tree is updated to $\mathcal{T}_4$ according to the received subtrees. The Single-Shard Execution

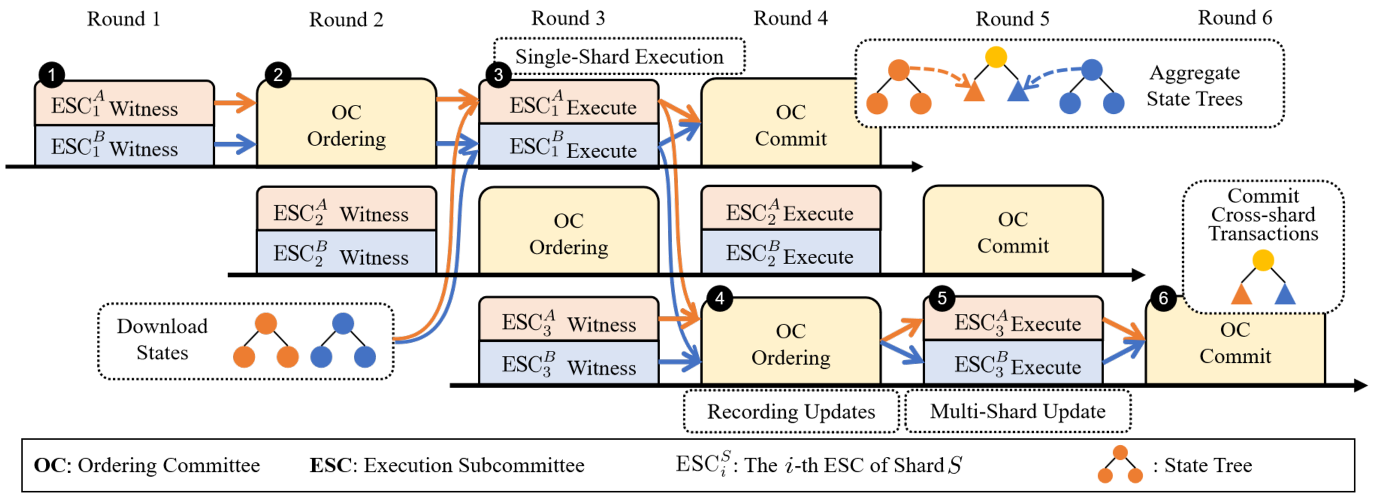


Fig. 6: The overview of integrating sharding into pipeline mechanism.

phase is accomplished.

❺ In round 5, ESC$_3^d$ belonging to shard d ($d \in \{A, B\}$) is in the Execution Phase. It downloads $\mathcal{B}_4$, and updates the state subtree according to both L_4 and U_4. Each ESC downloads states in U_4, which are maintained by itself and directly updates these key-value pairs and the state subtree. Similar to step ❸, ESC$_3^d$ outputs $\mathcal{T}_5^d$ and S_5^d.

❻ In round 6, the newest state tree root is calculated according to subtree roots $\mathcal{T}_5^A$ and $\mathcal{T}_5^B$. Since the state updated by cross-shard transactions has now been committed to the state tree recorded in $\mathcal{B}_6$, the Multi-Shard Update phase is finalized.

We now conclude that intra-shard transactions witnessed in round i are finally committed in round $(i+3)$, and cross-shard transactions witnessed in round i are finally committed in round $(i+5)$.

Workloads of different committees. The workloads of a specific ESC$_i^d$ in shard d are as follows:

- In round i, ESC$_i^d$ is in the Witness Phase. It downloads transaction blocks and returns witness proofs W_i^d.
- In round $(i+2)$, ESC$_i^d$ is in the Execution Phase. It updates the state subtree $\mathcal{T}_{i+1}^d$ according to the list of intra-shard transactions $L_{i+1}[d]$.ITx and the execution results of cross-shard transactions $U_{i+1}[d]$, both of which are included in $\mathcal{B}_{i+1}$. It returns updated subtree $\mathcal{T}_{i+2}^d$. It executes cross-shard transactions according to the list $L_{i+1}[d]$.CTx recorded in $\mathcal{B}_{i+1}$. It returns the set of updated key-value pairs S_{i+2}^d.

The workloads of the OC in any specific round i are described as follows:

- Receiving witness proofs W$_{i-1}^d$ from all shards (where $d \in \mathbb{Z}$) and ordering transactions into the list L_i, which distributes transactions to each shard for execution.
- Receiving sets S_{i-1}^d composed of states updated with cross-shard transactions from all shards and generates a new set U_i to distribute the workloads of update states to each shard.
- Receiving subtree $\mathcal{T}_{i-1}^d$ from all shards and calculating the new state tree $\mathcal{T}_i$.
- Running the consensus algorithm to commit a new block $\mathcal{B}_i$, which contains L_i, U_i and $\mathcal{T}_i$.

E. Performance Analysis

Complexity of Consensus. Without loss of generality, we calculate the complexity of committing a block in our proposed system. We denote the committee size, the total number of nodes in the network, and the block size by m, n, and b, respectively. Initially, a round of consensus is required to agree on the proposal block, contributing to a complexity of $\mathcal{O}(m^2)$. Subsequently, due to cross-shard transactions, the execution nodes in each shard must forward relevant cross-shard transaction information, including witness and proposal blocks, which is assumed to be of size w. Given that the number of shards is approximately equal to n/m, the communication between different shards contributes to the complexity of $\mathcal{O}(wn/m)$. Therefore, the overall complexity is $\mathcal{O}(m^2 + wn/m)$. In comparison, other sharding blockchains like RapidChain [37] exhibit a complexity of $\mathcal{O}(m^2 + bm\log n \cdot n/m) = \mathcal{O}(m^2 + bn\log n)$, while Elastico [35] and OmniLedger [36] show $\mathcal{O}(m^2 + bn)$, where $\mathcal{O}(m^2)$ is for shard consensus and the rest for cross-shard transaction communication.

Our Porygon has lower complexity due to the efficient routing of transactions by each shard. In RapidChain, all committee members need to forward transactions to other shards. Elastico uses a final committee to aggregate transactions and then send them to all nodes, and OmniLedger requires node-client interaction within all shards. In contrast, Porygon simplifies this process by requiring each shard to forward transactions only once at a time.

Complexity of Storage. We denote the total size of blockchain information as $|B|$. In RapidChain and OmniLedger, each shard shares this information, resulting in a complexity of $\mathcal{O}(m \cdot |B|/n)$. In contrast, in our system, stateless nodes only store messages necessary for verification, leading to a storage complexity of $\mathcal{O}(1)$.

V. Security Analysis

This section analyzes the security and liveness of our system. We present several lemmas to prove the correctness of separated phases in the pipeline. We first distinguish between *benign* committee members and *corrupted* committee members. Benign committee members are honest committee members who connect to at least one honest storage node. Corrupted committee members include (1) malicious committee members and (2) honest committee members, who nevertheless connect to malicious storage nodes only. We assume that the corrupted nodes are uniformly distributed across all the committees [8], [11]. Honest storage nodes will gossip all valid messages they have received to the whole network. Consequently, if a message is uploaded by a benign committee member to all connected storage nodes, all benign storage nodes can finally download the same message. However, honest-yet-corrupted nodes are separated from honest storage nodes. Malicious storage nodes can drop messages sent from honest storage nodes or other honest stateless nodes.

We first prove Lemma 1 that all committees, including the OC and ECs in each shard, have enough benign nodes to conduct the Ordering Phase and the Execution Phase. To prove Lemma 1, we use the Kullback-Leibler divergence [44] to measure the distance between different distributions by $D_{\mathrm{KL}}(p||q)$, i.e., $D_{\mathrm{KL}}(p||q) = p\ln(p/q)+(1-p)\ln((1-p)/(1-q))$. The total number of stateless nodes is M. The probability of each stateless node being selected as a member of a specific committee is p. The fraction of honest stateless nodes is α, and the fraction of malicious storage nodes is β. Each stateless node randomly connects to m storage nodes. Particularly, we set the average committee size $M_c = 3,500$, $\alpha = 0.75$, $\beta = 0.5$, and $m = 20$. Note that the committee size can be decreased to less than 100 in practice while still assuring security, utilizing the idea of safety-liveness dichotomy [29]. We say that a probability is negligible when the probability is less than $2^{-\kappa}$, with $\kappa = 30$.

Lemma 1. *Every committee has at least 2/3 benign nodes, except those with negligible probability.*

Proof. We denote the probability that a stateless node is a benign node by p_g. We have $p_g = (1-\beta^m)\alpha p$. According to the Chernoff bound [44], the number of benign stateless nodes in each committee $n_g \geq (p_g - \epsilon_g)M$, except those with probability $p_1 = \exp\{-D_{\mathrm{KL}}(p_g - \epsilon_g||p_g)M\}$, with $\epsilon_g \in [0, p_g]$. In particular, we choose ϵ_g to ensure that each committee has at least $\check{n}_g = 2,225$ benign nodes, except those with negligible probability. Similarly, we denote the probability that a stateless node is a corrupted node by p_c. We have $p_c = \beta^m\alpha p + (1-\alpha)p$. The number of corrupted stateless nodes in each committee $n_c \leq (p_c + \epsilon_c)M$, except those with probability $p_2 = \exp\{-D_{\mathrm{KL}}(p_c + \epsilon_c||p_c)M\}$, with $\epsilon_c \in [0, 1-p_c]$. We choose ϵ_c to ensure that each committee has at most $\hat{n}_c = 1,075$ nodes, except those with negligible probability. Since $\check{n}_g > 2\hat{n}_c$, we conclude that each committee has at least a 2/3 fraction of benign members, except those with negligible probability. □

Lemma 2. *(Ordering Phase): If a transaction block proposed by the leader of the OC receives at least T_w witness proofs, then the transaction block can be included in the output of the consensus algorithm, except those with negligible probability.*

Proof. The threshold T_w can be larger than the upper bound of the corrupted nodes, e.g., $T_w = \hat{n}_c + 1$. Therefore, the transaction block that receives at least T_w witness proofs must be downloaded by at least one benign node of EC. Benign nodes can upload transaction blocks they have downloaded to connected storage nodes in case these transaction blocks are generated by malicious storage nodes and have not been broadcast to honest storage nodes. As a result, all honest storage nodes can receive the transaction block, ensuring that benign nodes in the OC can receive and verify the witness proofs of the transaction block proposed by the leader. Moreover, Lemma 1 proves that there are at least 2/3 benign members in the OC. Hence, such transaction block can be included in the output of the consensus algorithm as long as the chosen consensus algorithm assures security. □

Lemma 3. *(Execution Phase): If OC agrees on a proposal block, then EC outputs at least $\check{n}_g$ identical roots consistent with the consensus result, except those with negligible probability.*

Proof. The OC agrees on the proposal block and uploads its commit message to connected storage nodes. Consequently, all benign nodes in the EC can verify the consensus results and download the same proposal block and transaction blocks from honest storage nodes. According to Lemma 1, at least $\check{n}_g$ benign nodes in the EC can download the complete contents of transaction blocks referred by the proposal block and produce identical updated Merkle Tree roots through the deterministic execution process. Note that EC has a higher fault tolerance of 1/2 after decoupling the Execution Phase from the Ordering Phase [45]. Hence, $\check{n}_g$ benign nodes are enough to assure the security of the Execution Phase. □

Lemma 4. *(Commit Phase): If the leader of the OC proposes a state tree root signed by at least T_e EC members, then such a root will finally be committed through the consensus algorithm, except those with negligible probability.*

Proof. When threshold T_e is $\hat{n}_c + 1$, at least one benign node has signed the state tree root. Therefore, such a root is the correct execution result, since all benign nodes are honest and hence always follow the system protocol. Moreover, such a root can be successfully downloaded and agreed upon by at least 2/3 benign members in the OC, which is similar to the case in Lemma 2. □

We then have the following theorems for the security and liveness of the proposed Porygon.

Theorem 1 (Security)**.** *If the system maintains a consistent state at the beginning of each round, then the blockchain state* *will be updated consistently and correctly after committing the state tree root.*

Proof. A committed state tree root has to satisfy the following requirements: all related transaction blocks are witnessed by at least T_w EC members, confirmed by the OC during the Ordering Phase, signed by at least T_e EC members, and finally agreed upon during the Commit Phase. According to Lemma 1, each committee satisfies the Byzantine fault tolerance threshold of 2/3. Therefore, the OC can output the unique consensus result in each round. According to Lemma 3, the choice of T_e ensures that such a state tree root is the correct execution result. Hence, there are no two valid-yet-different roots. The committed root maps to the correct blockchain states, which are consistent among all benign nodes. □

Theorem 2 (Liveness). *A transaction block generated by honest storage nodes will finally be committed.*

Proof. If a transaction block is generated by an honest storage node, it can be downloaded by all benign nodes in ECs and hence receives at least T_w witness proofs. If the leader of the OC is benign, it will package the transaction block in its proposal block. According to Lemmas 2-4, such a proposal block can be forwarded from the Ordering Phase to the Commit Phase. As a consequence, the system outputs an empty block only when a corrupted node is selected as the leader of the OC. The probability that a consensus leader is corrupted is 0.25. Hence, the probability that empty blocks are committed in more than 15 successive rounds is negligible. When the system selects a benign node as the leader of the OC, valid transaction blocks can be committed. □

VI. Evaluation

We conduct extensive experiments on an implemented prototype of Porygon and large-scale simulations with 3D parallelism to evaluate the effectiveness of our system in comparison with other representative blockchain systems with only 1D parallelism.

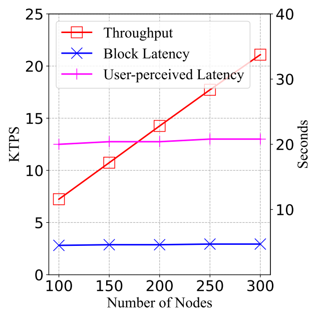
Implementation and Setup. We implement a prototype of Porygon on top of six cloud servers, each of which is equipped with Intel Cascade Lake 3.0GHz 24vCPUs and 48GB RAMs. This prototype contains instances of two storage nodes and up to 300 stateless nodes in Golang, with a total codebase of approximately 8,000 lines. Each storage node is hosted on a dedicated cloud server, and WebSocket connections facilitate data synchronization between storage nodes. Transaction data is stored in MySQL within each storage node. Stateless nodes are implemented with Docker containers distributed on four cloud servers. Each stateless node has 500M memory and 1 MB/s bandwidth, which is compatible with resource-limited mobile devices [11]. Stateless nodes connect with storage nodes through HTTP links to run the BA$\star$ consensus protocol and process transactions. Each transaction is about 112 bytes in size. Each transaction block contains about 2,000 transactions and has to receive at least 10 witness signatures to be considered valid.

To validate Porygon's performance with a large number of nodes, we design Python-based simulations involving up to 100,000 nodes. In the simulations, we specifically focus on the design of 3D parallelism, omitting the intricate engineering aspects of distributed architecture between nodes. Based on empirical observations from our prototype system, we exclude the network restructuring component for simplicity, substituting it by a fixed interval of 2 seconds plus random numerical values to model the rapid committee formation process. Additionally, we fix the latency between storage nodes and stateless nodes at approximately 0.5 milliseconds, intended to represent the baseline transmission time for messages.

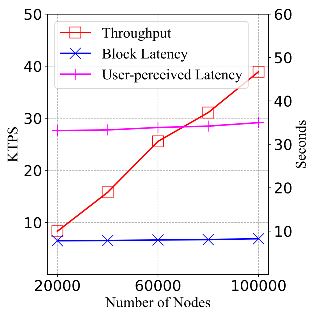
Comparisons. We implement Blockene [11] based on our codebase and ByShard [38] based on Tendermint [46]. Blockene is a representative of stateless blockchain systems with storage-consensus parallelism that still sequentially processes all blocks. ByShard is a representative of sharding systems, employing a two-phase cross-shard protocol that appoints the sender shard as the coordinator for cross-shard transactions. Specifically, our implementation of ByShard leverages a distributed version of the protocol, which exhibits superior performance compared to linear or centralized protocols, thus rendering it more symbolic. Byshard features inter-block parallelism where transactions are executed by multiple groups in parallel but still require nodes to store the ever-growing states. For a fair comparison, we choose the same settings for our Porygon, Blockene, and Porygon to evaluate performance scalability. We let the bandwidth and memory setting of nodes in ByShard be the same as that of Porygon and Blockene, consequently implementing a lightweight version of ByShard, namely *lightweight ByShard*.

A. Scalability Performance

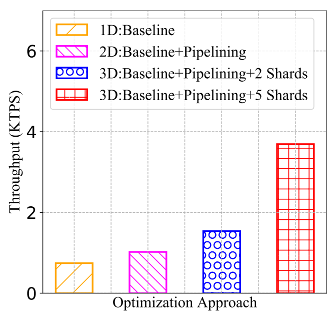
We evaluate the performance scalability of Porygon by evaluating the throughput and latency of the system when the number of nodes increases. In prototype experiments, each shard contains 10 nodes. We then increase the number of shards from 10 to 30. Consequently, the network scale increases from 100 nodes to 300 nodes. Figure 7(a) shows a linearly increased throughput, from about 7,240 Transactions Per Second (TPS) to 21,090 TPS. This is because inner-block parallelism can effectively distribute workloads to different shards. Meanwhile, the latency of creating a new block only slightly increases from 4.5 seconds to 4.7 seconds when the network scale increases from 100 nodes to 300 nodes. The average latency of committing transactions remains stable at about 13 seconds. This is because the OC only consumes a stable time interval to run the consensus algorithm to create new blocks with our new storage-consensus parallelism design. The *user-perceived latency*, defined as the duration from the time when a user sends a message until the time when they receive confirmation of its inclusion in the blockchain increases from 20 seconds to 21 seconds. We also evaluate the system performance through simulations with up to 100,000 stateless nodes. The number of shards increases from 10 shards to 50 shards, each containing 2,000 stateless nodes. Figure 7(b)



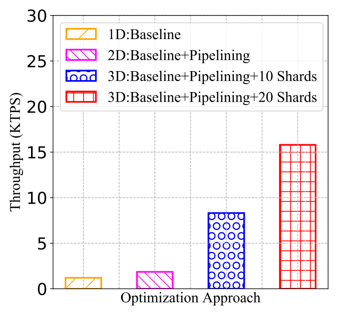
(a) Throughput and latency in prototype experiments



(b) Throughput and latency in simulations

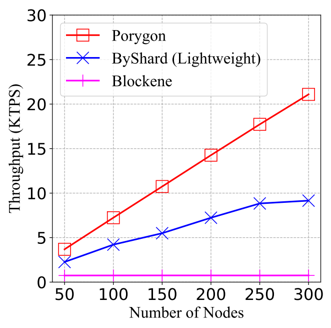


(c) Optimization effect in prototype experiments

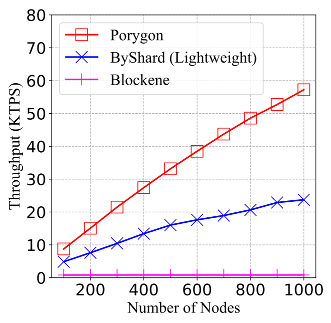


(d) Optimization effect in simulations

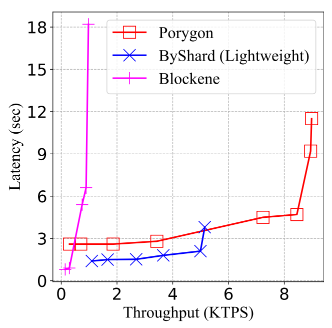
Fig. 7: Performance of Porygon under different settings



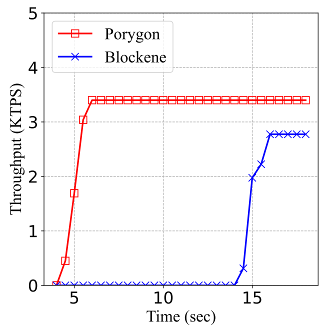
(a) Throughput comparison in prototype experiments



(b) Throughput comparison in simulations



(c) Throughput versus Latency (# of nodes = 100)



(d) Throughput under varied participating time of nodes

Fig. 8: Performance comparison between different systems

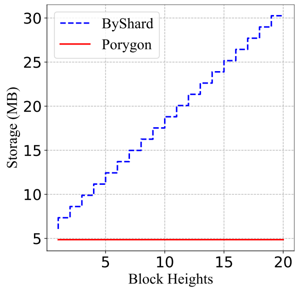
depicts the throughput and the latency of simulations. The throughput increases from 8,310 TPS to 38,940 TPS, while the latency slightly increases from 7.8 seconds to 8.3 seconds, and the user-perceived latency increases from 33 seconds to 35 seconds. The results demonstrate Porygon's high scalability. Its throughput increases with the increased network scale (i.e., an increased number of nodes) even when each shard has a large size to assure security.

We evaluate the effectiveness of the parallelization mechanisms (i.e., pipelining and sharding) of our Porygon with two more dimensional parallelisms. Figure 7(c) demonstrates the throughput of our implementation when applying different parallelization approaches. We consider a simplified version of Porygon as the 1D parallelism baseline without pipelining and sharding functions. The baseline is composed of 2 storage nodes and 10 stateless nodes. The baseline system achieves 740 TPS. We then introduce pipelining and sharding functions to the baseline system, in which each shard contains 10 more nodes. Results show that pipelining can increase the throughput of the baseline to 1,020 TPS. This is because inter-block parallelism can increase the number of transactions that the system can commit each time (as indicated in § IV-C2). Moreover, inner-block parallelism further speeds up the process by reducing the time of executing transactions with appropriate cross-shard coordination (as shown in § IV-D). Figure 7(d) demonstrates the effectiveness of inter-block and inner-block parallelism in a larger scale network by simulations. Particularly, similar to prototype experiments, better performance is achieved by applying the pipelining function and adding more shards. In summary, both experiments and simulations further confirm the effectiveness of our design.

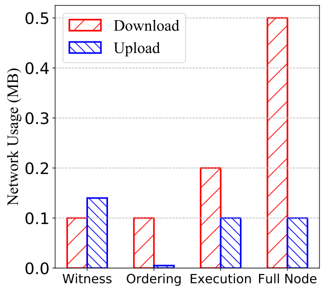
B. Performance Comparison

We compare Porygon with representative blockchain systems only with 1D parallelism: ByShard [38] and Blockene [11]. We first conduct prototype experiments to evaluate Porygon, ByShard, and Blockene in terms of the throughput when the number of nodes increases from 50 nodes to 300 nodes. Each shard in both ByShard and Porygon has 10 nodes. Figure 8(a) shows that the throughput of ByShard increases from about 2,260 TPS to 9,150 TPS. Meanwhile, the throughput of Blockene remains at about 750 TPS, due to the lack of further parallelism among participating nodes of Blockene. By contrast, our Porygon achieves a throughput of more than 21,090 TPS with 300 nodes, which is much higher than ByShard and Blockene. Figure 8(b) also compares our Porygon with ByShard and Blockene in large-scale simulations. The number of nodes increases from 100 nodes to 1,000 nodes. When the network scale grows, our Porygon can achieve the fastest throughput increment, i.e., increasing from 8,760 TPS to 57,220 TPS, further demonstrating the best scalability of our Porygon among all the methods.

Figure 8(c) compares Porygon with ByShard and Blockene in terms of throughput versus latency. We submit transactions to the implemented prototype of Porygon, ByShard, and Blockene with varied frequencies. We then record the corresponding throughput and latency. All the three systems are tested with 100 nodes. Both Porygon and ByShard are



(a) Storage consumption of different systems



(b) Network usage of different phases/nodes

Fig. 9: Resource consumption of different systems (where # of nodes = 100)

assigned with 10 shards. We observe that Porygon achieves a higher capacity for processing transactions than ByShard and Blockene even though Porygon has longer latency at first, mainly caused by the communication latency between storage nodes and stateless nodes. This can be attributed to the 3D parallelism of our Porygon, thereby achieving high throughput while maintaining moderate latency. Moreover, Figure 8(d) plots the throughput of Porygon and Blockene with the varied time of nodes staying in the network. Committee members in Blockene have to sequentially process 50 blocks before reconfiguration. In Porygon, committee members stay in the network for 3 rounds. When stateless nodes only stay in the blockchain system for a short period, Blockene encounters a higher probability of execution failure as nodes in Blockene have a longer transaction processing cycle. As a result, committees have to commit empty blocks, thereby compromising system performance. In comparison, Porygon assigns a shorter lifecycle for participating nodes due to our inter-block parallelism design. Hence, Porygon is quite robust to the scenario when the network demands frequent reconfigurations (e.g., nodes frequently joining or leaving).

Building upon insights from previous studies [47], [48], we conducted additional simulations to assess how different cross-shard transaction ratios affect system throughput and latency. Table I shows that throughput decreases from 9,179 to 8,810 TPS, and latency slightly increases from 7.6 to 7.9 seconds in a 10-shard setting when the cross-shard transaction ratio increases from 0.5 to 1.0. These findings showcase our system's adaptability to various datasets in cross-shard systems.

C. Resource Consumption

We next evaluate the resource consumption of stateless nodes. In particular, we evaluate the storage and network consumption of stateless nodes in comparison to a full-node scheme (i.e., ByShard). We first compare the increment of storage consumption between ByShard and Porygon as the block height increases, as depicted in Figure 9(a). Blocks in both systems contain about 1,000 transactions. In ByShard, full nodes must store complete block contents. Consequently, the storage consumption significantly increases with constantly increased new blocks. By comparison, stateless nodes in Porygon only store messages necessary for verification, such as the block header and the public keys of committee members, which remain at about 5 MB.

TABLE I: Performance comparison under different cross-shard transaction ratios

Ratio	0.5	0.7	0.9	0.95	1.0
TPS	9179	9015	8911	8867	8810
Latency(s)	7.60	7.71	7.83	7.84	7.89

We also measure the network usage of different phases in Porygon in comparison to the ByShard full node. The result is reported in Figure 9(b). Both Porygon and ByShard are tested with 10 shards and 100 nodes. A full node in ByShard has to spend bandwidth resources on downloading transactions and blocks within each round, which takes about 1.7 seconds. In comparison, Porygon can achieve lower per-node overhead by distributing network usage among different phases and different committees. The time interval for each phase in Porygon is 1.7 seconds on average. In the Witness Phase, stateless nodes in the EC download transactions and upload corresponding signatures. In the Ordering Phase, stateless nodes in the OC replicate signatures of EC and lists of transaction blocks. In the Execution Phase, stateless nodes in EC download the latest block and required states, and upload updated states. Note that transactions downloaded in the Witness Phase do not have to be downloaded twice. The network usage for each phase drops by 50% to 80% compared with full nodes. The result verifies that our 3D parallelism does not impose additional overhead on the stateless nodes.

VII. Conclusion

We present Porygon, a novel stateless blockchain that enhances performance through three-dimensional parallelism. Specifically, Porygon comprises stateless nodes and storage nodes, responsible for transaction execution and state storage, respectively. Secondly, we devise a fine-grained transaction-processing pipeline to achieve inter-block parallelism. Finally, we introduce a sharding mechanism to parallelize transaction execution for inner-block parallelism. To prove the performance improvement brought by our system, we conduct extensive experiments and compare it with other related blockchain systems focusing on scaling blockchain in both realistic network environments and simulations. Experimental results demonstrate that our system achieves higher throughput i.e., 2.3× improvement of sharding systems and 20× improvement of stateless systems, while also maintaining low storage consumption and network usage.

References

[1] P. Sangha, V. Pureswaran, , and S. Soman, "Advancing global trade with blockchain," *IBM Research Insights*, 2020. [Online]. Available: https://www.ibm.com/downloads/cas/WVDE0MXG

[2] V. Gaur and A. Gaiha, "Building a Transparent Supply Chain," *Harvard Business Review*, May 2020. [Online]. Available: https://hbr.org/2020/05/building-a-transparent-supply-chain

[3] J. Morey, "The Future Of Blockchain In Healthcare," *Forbes Innovation*, Oct 2021. [Online]. Available: https://www.forbes.com/sites/forbestechcouncil/2021/10/25/the-future-of-blockchain-in-healthcare/

[4] M. J. Amiri, D. Agrawal, and A. E. Abbadi, “CAPER: A cross-application permissioned blockchain,” *Proc. VLDB Endow.*, vol. 12, no. 11, pp. 1385–1398, 2019. [Online]. Available: http://www.vldb.org/pvldb/vol12/p1385-amiri.pdf

[5] M. El-Hindi, C. Binnig, A. Arasu, D. Kossmann, and R. Ramamurthy, “Blockchaindb - A shared database on blockchains,” *Proc. VLDB Endow.*, vol. 12, no. 11, pp. 1597–1609, 2019. [Online]. Available: http://www.vldb.org/pvldb/vol12/p1597-el-hindi.pdf

[6] S. Nathan, C. Govindarajan, A. Saraf, M. Sethi, and P. Jayachandran, “Blockchain meets database: Design and implementation of a blockchain relational database,” *Proc. VLDB Endow.*, vol. 12, no. 11, pp. 1539–1552, 2019. [Online]. Available: http://www.vldb.org/pvldb/vol12/p1539-nathan.pdf

[7] K. Babel, P. Daian, M. Kelkar, and A. Juels, “Clockwork finance: Automated analysis of economic security in smart contracts,” in *2023 IEEE Symposium on Security and Privacy (SP)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2023, pp. 622–639. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/SP46215.2023.00036

[8] C. Xu, C. Zhang, J. Xu, and J. Pei, “Slimchain: Scaling blockchain transactions through off-chain storage and parallel processing,” *Proceedings of the VLDB Endowment*, vol. 14, no. 11, pp. 2314–2326, 2021.

[9] B. Bunz, L. Kiffer, L. Luu, and M. Zamani, “Flyclient: Super-light clients for cryptocurrencies,” in *2020 IEEE Symposium on Security and Privacy (SP)*, 2020, pp. 928–946.

[10] T. Xie, J. Zhang, Z. Cheng, F. Zhang, Y. Zhang, Y. Jia, D. Boneh, and D. Song, “ZkBridge: Trustless Cross-Chain Bridges Made Practical,” in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’22, 2022, p. 3003–3017.

[11] S. Satija, A. Mehra, S. Singanamalla, K. Grover, M. Sivathanu, N. Chandran, D. Gupta, and S. Lokam, “Blockene: A high-throughput blockchain over mobile devices,” in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*. USENIX Association, Nov. 2020, pp. 567–582. [Online]. Available: https://www.usenix.org/conference/osdi20/presentation/satija

[12] M. Campanelli, A. Nitulescu, C. Ràfols, A. Zacharakis, and A. Zapico, “Linear-map vector commitments and their practical applications,” in *Advances in Cryptology–ASIACRYPT 2022: 28th International Conference on the Theory and Application of Cryptology and Information Security, Taipei, Taiwan, December 5–9, 2022, Proceedings, Part IV*. Springer, 2023, pp. 189–219.

[13] M. J. Amiri, Z. Lai, L. Patel, B. T. Loo, E. Lo, and W. Zhou, “Saguaro: An edge computing-enabled hierarchical permissioned blockchain,” in *39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, CA, USA, April 3-7, 2023*. IEEE, 2023, pp. 259–272. [Online]. Available: https://doi.org/10.1109/ICDE55515.2023.00027

[14] 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*, ser. SIGMOD ’19. New York, NY, USA: Association for Computing Machinery, 2019, p. 123–140. [Online]. Available: https://doi.org/10.1145/3299869.3319889

[15] W. Wang, S. Deng, J. Niu, M. K. Reiter, and Y. Zhang, “ENGRAFT: Enclave-guarded Raft on Byzantine Faulty Nodes,” in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2841–2855.

[16] Y. Yan, C. Wei, X. Guo, X. Lu, X. Zheng, Q. Liu, C. Zhou, X. Song, B. Zhao, H. Zhang, and G. Jiang, “Confidentiality support over financial grade consortium blockchain,” in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD ’20. New York, NY, USA: Association for Computing Machinery, 2020, p. 2227–2240. [Online]. Available: https://doi.org/10.1145/3318464.3386127

[17] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. D. Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich, S. Muralidharan, C. Murthy, B. Nguyen, M. Sethi, G. Singh, K. Smith, A. Sorniotti, C. Stathakopoulou, M. Vukolić, S. W. Cocco, and J. Yellick, “Hyperledger fabric: a distributed operating system for permissioned blockchains,” in *Proceedings of the Thirteenth EuroSys Conference on*, 2018, p. 30.

[18] J. Qi, X. Chen, Y. Jiang, J. Jiang, T. Shen, S. Zhao, S. Wang, G. Zhang, L. Chen, M. H. Au *et al.*, “BIDL: A high-throughput, low-latency permissioned blockchain framework for datacenter networks,” in *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, 2021, pp. 18–34.

[19] L. Yang, S. J. Park, M. Alizadeh, S. Kannan, and D. Tse, “DispersedLedger: High-Throughput Byzantine Consensus on Variable Bandwidth Networks,” in *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, 2022, pp. 493–512.

[20] Z. Peng, Y. Zhang, Q. Xu, H. Liu, Y. Gao, X. Li, and G. Yu, “Neuchain: A fast permissioned blockchain system with deterministic ordering,” *Proc. VLDB Endow.*, vol. 15, no. 11, p. 2585–2598, jul 2022. [Online]. Available: https://doi.org/10.14778/3551793.3551816

[21] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias, “Bullshark: Dag bft protocols made practical,” in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS ’22. New York, NY, USA: Association for Computing Machinery, 2022, p. 2705–2718. [Online]. Available: https://doi.org/10.1145/3548606.3559361

[22] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, “All you need is dag,” in *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*, 2021, pp. 165–175.

[23] S. Gupta, S. Rahnama, J. Hellings, and M. Sadoghi, “Resilientdb: global scale resilient blockchain fabric,” *Proc. VLDB Endow.*, vol. 13, no. 6, p. 868–883, feb 2020. [Online]. Available: https://doi.org/10.14778/3380750.3380757

[24] 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)*. Los Alamitos, CA, USA: IEEE Computer Society, apr 2021, pp. 1392–1403. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/ICDE51399.2021.00124

[25] S. Gupta, J. Hellings, S. Rahnama, and M. Sadoghi, “Proof-of-execution: Reaching consensus through fault-tolerant speculation,” in *Proceedings of the 24th International Conference on Extending Database Technology, EDBT 2021, Nicosia, Cyprus, March 23 - 26, 2021*, Y. Velegrakis, D. Zeinalipour-Yazti, P. K. Chrysanthis, and F. Guerra, Eds. OpenProceedings.org, 2021, pp. 301–312. [Online]. Available: https://doi.org/10.5441/002/edbt.2021.27

[26] X. Qi, Z. Chen, H. Zhuo, Q. Xu, C. Zhu, Z. Zhang, C. Jin, A. Zhou, Y. Yan, and H. Zhang, “Schain: Scalable concurrency over flexible permissioned blockchain,” in *39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, CA, USA, April 3-7, 2023*. IEEE, 2023, pp. 1901–1913. [Online]. Available: https://doi.org/10.1109/ICDE55515.2023.00148

[27] M. Fang, X. Zhou, Z. Zhang, C. Jin, and A. Zhou, “Seframe: An sgx-enhanced smart contract execution framework for permissioned blockchain,” in *38th IEEE International Conference on Data Engineering, ICDE 2022, Kuala Lumpur, Malaysia, May 9-12, 2022*. IEEE, 2022, pp. 3166–3169. [Online]. Available: https://doi.org/10.1109/ICDE53745.2022.00289

[28] M. Fang, Z. Zhang, C. Jin, and A. Zhou, “High-performance smart contracts concurrent execution for permissioned blockchain using sgx,” in *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, 2021, pp. 1907–1912.

[29] B. David, B. Magri, C. Matt, J. B. Nielsen, and D. Tschudi, “Gearbox: Optimal-size shard committees by leveraging the safety-liveness dichotomy,” in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 683–696.

[30] Z. Cai, J. Liang, W. Chen, Z. Hong, H.-N. Dai, J. Zhang, and Z. Zheng, “Benzene: Scaling blockchain with cooperation-based sharding,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 2, pp. 639–654, 2022.

[31] J. Zhang, Z. Hong, X. Qiu, Y. Zhan, S. Guo, and W. Chen, “Skychain: A deep reinforcement learning-empowered dynamic blockchain sharding system,” in *Proceedings of the 49th International Conference on Parallel Processing*, 2020, pp. 1–11.

[32] M. J. Amiri, D. Agrawal, and A. El Abbadi, *SharPer: Sharding Permissioned Blockchains Over Network Clusters*. New York, NY, USA: Association for Computing Machinery, 2021, p. 76–88. [Online]. Available: https://doi.org/10.1145/3448016.3452807

[33] Y. Zhang, S. Pan, and J. Yu, “Txallo: Dynamic transaction allocation in sharded blockchain systems,” in *39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, CA, USA, April 3-7, 2023*. IEEE, 2023, pp. 721–733. [Online]. Available: https://doi.org/10.1109/ICDE55515.2023.00390

[34] P. Zheng, Q. Xu, Z. Zheng, Z. Zhou, Y. Yan, and H. Zhang, “Meepo: Sharded consortium blockchain,” in *37th IEEE International Conference on Data Engineering, ICDE 2021, Chania, Greece, April*

19-22, 2021. IEEE, 2021, pp. 1847–1852. [Online]. Available: https://doi.org/10.1109/ICDE51399.2021.00165

[35] 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*, 2016, pp. 17–30.

[36] 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, pp. 583–598.

[37] M. Zamani, M. Movahedi, and M. Raykova, "RapidChain: Scaling Blockchain via Full Sharding," in *CCS '18 Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, 2018, pp. 931–948.

[38] J. Hellings and M. Sadoghi, "ByShard: Sharding in a Byzantine Environment," *Proc. VLDB Endow*, vol. 14, no. 11, p. 2230–2243, jul 2021.

[39] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 51–68.

[40] T. Alves, "TrustZone : Integrated Hardware and Software Security," *White Paper*, 2004.

[41] S. Pinto and N. Santos, "Demystifying ARM TrustZone: A Comprehensive Survey," *ACM Comput. Surv.*, vol. 51, no. 6, jan 2019. [Online]. Available: https://doi.org/10.1145/3291047

[42] 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*, ser. PLDI 2021. New York, NY, USA: Association for Computing Machinery, 2021, p. 1327–1341. [Online]. Available: https://doi.org/10.1145/3453483.3454112

[43] S. Micali, M. Rabin, and S. Vadhan, "Verifiable random functions," in *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, 1999, pp. 120–130.

[44] W. Mulzer, "Five proofs of chernoff's bound with applications," *arXiv preprint arXiv:1801.03365*, 2018.

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

[46] D. Cason, E. Fynn, N. Milosevic, Z. Milosevic, E. Buchman, and F. Pedone, "The design, architecture and performance of the tendermint blockchain network," in *2021 40th International Symposium on Reliable Distributed Systems (SRDS)*, 2021, pp. 23–33.

[47] J. Zhang, W. Chen, S. Luo, T. Gong, Z. Hong, and A. Kate, "Front-running attack in sharded blockchains and fair cross-shard consensus." NDSS, 2024.

[48] H. Huang, X. Peng, J. Zhan, S. Zhang, Y. Lin, Z. Zheng, and S. Guo, "Brokerchain: A cross-shard blockchain protocol for account/balance-based state sharding," in *IEEE INFOCOM 2022 - IEEE Conference on Computer Communications*, 2022, pp. 1968–1977.