SharDAG: Scaling DAG-based Blockchains via Adaptive Sharding

Feng Cheng*, Jiang Xiao*, Cunyang Liu*, Shijie Zhang*, Yifan Zhou*, Bo Li†, Baochun Li‡, Hai Jin*
*National Engineering Research Center for Big Data Technology and System
*Services Computing Technology and System Lab, Cluster and Grid Computing Lab
*School of Computer Science and Technology, Huazhong University of Science and Technology, China
†Hong Kong University of Science and Technology, Hong Kong
‡University of Toronto, Canada
Email: jiangxiao@hust.edu.cn

***Abstract*—*Directed Acyclic Graph* (DAG)-based blockchain (a.k.a distributed ledger) has become prevalent for supporting highly concurrent applications. Its inherent parallel data structure accelerates block generation significantly, shifting the bottleneck from performance to storage scalability. An intuitive solution is to apply state sharding that divides the entire ledger (i.e., transactions and states) into multiple shards. While each node only stores proportional transactions, it suffers from the challenges of storing and ensuring the processing consistency of cross-shard transactions. In this paper, we propose *SharDAG*, a new mechanism that leverages adaptive sharding for DAG-based blockchains to achieve high performance and strong consistency. The key idea of *SharDAG* is to exploit unique characteristics — silent assets — and design a lightweight processing mechanism based on avatar account caching. Furthermore, we design a Byzantine resilient cross-shard verification mechanism with a theoretically optimal number of participating nodes, which guarantees the consistency and security of avatar account aggregation. Our comprehensive evaluations on real-world workloads demonstrate that SharDAG presents up to 3.8× throughput improvement compared to the state-of-the-art and reduces the storage overhead of cross-shard transactions.**

***Index Terms*—DAG-based blockchain, sharding, cross-shard transaction processing**

I. Introduction

Directed Acyclic Graph (DAG)-based blockchain, which generates and appends blocks of transactions to the tamper-proof distributed ledger concurrently, has gained extensive interests in both industry [1]–[3] and academia [4]–[7]. Compared to traditional chain-based blockchains [8], [9], DAG-based blockchains exhibit distinctive advantages of throughput improvement owing to their inherent parallel structure. For example, Conflux [4], a prevailing DAG-based blockchain, achieves a throughput of over 3.4K *transactions per second* (TPS), which is several orders of magnitude higher than Bitcoin's 7 TPS [8]. We have witnessed the growing demands of developing DAG-based blockchains in supporting highly concurrent applications [10]–[12].

DAG-based blockchain with parallel topology has placed an unprecedented requirement on its underlying storage. This is because concurrent block generation significantly accelerates data expansion, leading to severe storage overhead. Moreover, existing DAG-based blockchains still adopt the full replication storage approach, in which nodes need to store and process the complete ledger (Fig. 1(a)). For instance, Conflux [4] can achieve a throughput of 3.4K TPS, leading to a rapid growth of the overall storage overhead of about 2.3 TB per year. The ever-growing DAG ledger will substantially increase the data volume and hinder its development. Thus, it is essential and desirable to improve the storage scalability of DAG-based blockchains toward widespread adoption.

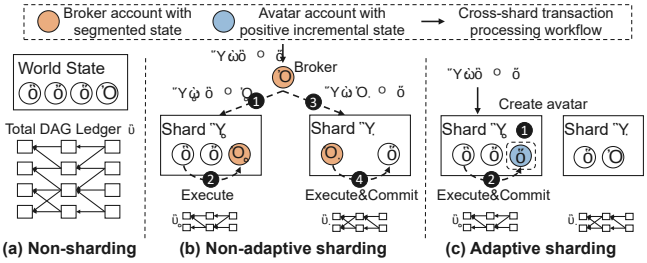


Fig. 1. Illustration for DAG-based blockchains using different sharding mechanisms. (a) Non-sharding. (b) Non-adaptive sharding (e.g., BrokerChain [13]). Broker accounts are segmented and placed to multiple shards to coordinate cross-shard transaction processing, which introduces complex workflows and fails to adapt to actual cross-shard workloads. (c) Adaptive sharding (SharDAG). Shards adaptively create avatar accounts to maintain the incremental states and efficiently process cross-shard transactions.

Sharding, as a common scale-out technique in the distributed database community [14]–[17], has been employed in chain-based blockchains to improve storage scalability and system performance [18]–[25]. It divides the nodes into small groups called shards. All shards collaboratively maintain the entire ledger and state, and each node only needs to process and store a part of it, thus alleviating the storage costs per node [26]. Some recent works propose cross-shard transaction processing methods such as transaction splitting [27], transaction relaying [28], and *two-phase commit* (2PC) methods [18], [29]–[32]. However, they all depend on a fixed account assignment strategy that assigns one account to one shard (i.e., one-to-one assignment), which leads to an increase in transactions involving accounts across shards, aggravating the overhead of cross-shard transaction processing.

Recently, Pyramid [33] leverages a bridge shard that stores account states from other shards (i.e., one-to-many assignment) to shorten the process of cross-shard transactions at the cost of limited storage scalability. To guarantee the consistency of cross-shard transactions, shards must cooperate and

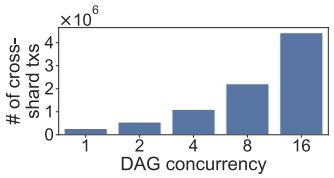


Fig. 2. An empirical study of the number of cross-shard transactions in an intuitive sharding DAG-based blockchain based on the one-to-one account assignment mode used in Monoxide [28]. DAG concurrency refers to the number of blocks generated concurrently in a DAG shard. We use 8 shards, 2000 transactions per block, and 20 rounds.

transmit cross-shard messages (mainly messages generated in intra-shard consensus) [34], [35]. Many works [26], [34], [36], [37] delegate the transmission and appending of cross-shard messages to the leader node (i.e., the leader of intra-shard consensus). When the leader behaves maliciously across shards, it requires complicated and long-delay remote and local recovery to ensure liveness, which degrades system performance. Periodic state reconfiguration that reassigns accounts is necessary to reduce the number of cross-shard transactions and load imbalance [13], [38], [39]. However, each shard typically utilizes a monolithic *Merkle Patricia Trie* (MPT) or its variants [13], [40], [41] to store account states, which is inefficient in migrating account states across shards due to complex hash operations on trie nodes.

Unfortunately, it falls short when sharding DAG-based blockchains and poses three fundamental challenges.

Challenge 1: How to design an adaptive sharding mechanism to achieve efficient cross-shard transaction processing? It is non-trivial to directly apply sharding on DAG-based blockchains since the concurrent block generation nature leads to an explosion in the number of cross-shard transactions. We reveal this through an empirical study via workloads from Ethereum, the most widely used blockchain platform [9], [42] (Fig. 2). The overwhelming number of cross-shard transactions brings frequent cross-shard communication, which limits the parallelism of shards and weakens the performance advantages of DAG-based blockchains. The intuitive sharding solutions applying a fixed account assignment strategy, e.g., one-to-one assignment (e.g., Monoxide [28]) or one-to-many assignment (e.g., BrokerChain [13]), are not adaptive to actual cross-shard workloads (Fig. 1(b)) and fail to provide efficient processing.

Challenge 2: How to design a Byzantine resilient cross-shard verification mechanism to ensure state consistency among multiple DAG shards? Cross-shard verification is vital for ensuring state consistency in a sharding DAG-based blockchain under an untrusted Byzantine environment. It involves two critical phases: cross-shard message transmission and appending, both requiring guaranteed safety and liveness. The traditional single leader-based transmission approach fails to ensure liveness when suffering from cross-shard censorship attacks [34] if the receiver shard does not know when the sender shard sends messages to it. Besides, in the receiver shard, each receiver node can append messages to the local DAG in parallel. The traditional single leader-based appending method [34] does not take advantage of this concurrent block generation nature and may suffer from long appending delays.

Challenge 3: How to design scalable state storage to support efficient state reconfiguration? In the periodic state reconfiguration scheme, the system proceeds in epochs, each including a consensus phase (containing multiple consensus rounds and processing a certain number of transactions) and a reconfiguration phase [13], [41]. DAG-based blockchain sharding requires more frequent state reconfiguration than its chain-based blockchain counterpart since the high-performance DAG-based consensus greatly shortens the consensus phase. However, using the traditional monolithic MPT to store account states may lead to performance bottlenecks in state reconfiguration. Specifically, when migrating account states across shards, it is necessary to delete nodes in the monolithic MPT and generate corresponding proofs, which involves hash operations of all nodes in the path from the leaf node to the root, resulting in long state reconfiguration delays.

In this paper, we propose SharDAG, the first adaptive sharding mechanism specialized for DAG-based blockchain, to tackle the three challenges above. We explore the impact of realistic workloads on sharding DAG-based blockchains to drive our design (Section III). For Challenge 1, we observe *silent assets*, which can potentially enhance cross-shard transaction efficiency. Specifically, newly received assets often have a prolonged silence period before being spent. This indicates that we can create an avatar account for the receiver of the cross-shard transaction to cache its received assets in the local shard temporarily. Thus, subsequent transactions involving the account can be processed locally. By leveraging this insight, we present an adaptive account assignment strategy based on cross-shard avatar account caching to achieve processing efficiency. As shown in Fig. 1(c), SharDAG adaptively creates an avatar account for the cross-shard receiver based on actual cross-shard workloads, caching incremental states rather than redundantly storing the entire shard state. Hence, cross-shard transactions can be locally executed in the sender shard without complex cross-shard interactions. In addition to simple payments, our account caching mechanism can support contract-based token transfers, which is suitable for most current contract-based applications [43].

For Challenge 2, we design a Byzantine resilient cross-shard verification scheme through theoretical analysis. Specifically, in the transmission phase, $2f + 1$ sender and $f + 1$ receiver nodes are sufficient to guarantee safety and liveness without complex remote recovery. In the appending phase, we leverage the concurrent block generation nature and design a dual-mode scheme to provide safety and liveness. Moreover, a cross-shard message can be appended to the DAG ledger after at most one failure, thus avoiding large appending delays. Based on such designs, we develop a cross-shard avatar account aggregation mechanism to aggregate avatar account states periodically, thereby ensuring state consistency.

For Challenge 3, we observe that the migrated accounts during state reconfiguration are *active accounts*, i.e., accounts appearing in recent transactions. The key insight is that there is no need to operate on dormant accounts in the MPT during

state reconfiguration. Therefore, we separate the state storage of active and dormant accounts and design a two-tier state storage model, which stores active and dormant accounts in two MPTs, *Active-Trie* and *Full-Trie*, respectively. Consequently, most operations are completed on the lightweight *Active-Trie* rather than a vast MPT containing active and numerous dormant accounts, thus accelerating state reconfiguration.

We implement SharDAG and evaluate it in WANs using real-world workloads. Results show that SharDAG achieves superior performance than several non-sharding/non-adaptive sharding DAG-based blockchains, i.e., a non-sharding scheme based on BullShark [7] and two sharding schemes using state-of-the-art sharding protocols, BrokerChain [13] and Monoxide [28]. Specifically, SharDAG with 16 shards achieves 7.5× lower storage overhead and 40× higher end-to-end throughput than the non-sharding scheme under the same network scale of 64 nodes. Moreover, SharDAG outperforms the two sharding schemes by up to 3.8× higher end-to-end throughput and 18× lower end-to-end latency with 16 shards. Further, SharDAG reduces the state reconfiguration delay by 35.9%.

This paper makes the following contributions:

- We introduce SharDAG, an adaptive scalable and efficient sharding mechanism for DAG-based blockchains.
- SharDAG comprises three key ingredients: a cross-shard avatar account caching scheme for efficient cross-shard transaction processing, a Byzantine cross-shard verification mechanism for consistent cross-shard avatar account aggregation, and a two-tier state storage model to support efficient state reconfiguration.
- We implement and evaluate SharDAG to demonstrate that SharDAG outperforms the state-of-the-art in both latency and throughput and provides storage scalability.

II. Background and Related Work

A. DAG-based Blockchains

DAG-based blockchains [1]–[7] use the *Directed Acyclic Graph* (DAG) data structure as the underlying storage model for concurrent transaction appending. It offers potential performance gains over conventional chain-based blockchains.

Different consensus protocols have been proposed to promote the performance of DAG-based blockchains, categorized as probabilistic and deterministic consensus. Probabilistic protocols, such as IOTA [1], Conflux [4], OHIE [5], and Prism [44], determine transaction confirmation based on transaction's depth or cumulative weight in the DAG ledger. However, they suffer from high confirmation latency and security risks. The latter category, e.g., BullShark [7], Tusk [45], and DAG-Rider [46], takes the merits of determinism in *Byzantine fault-tolerant* (BFT) consensus and decouples transaction dissemination from ordering logic, achieving high throughput and fast confirmation. Nezha [47] explores address-based dependency for conflict detection and develops a novel concurrent transaction processing mechanism for DAG-based blockchains. Until recently, LDV [10] investigates social relationships in *Vehicular Social Networks* (VSNs) and designs a topic group-enabled DAG storage reduction model.

Implications. As shown in Table I, existing DAG-based blockchains hardly scale out with the increasing throughput and storage demands. Most of them still adopt the full replication storage strategy, where each node stores a complete ledger copy. Our *SharDAG* serves as the first attempt to apply state sharding to design a storage scalable DAG-based blockchain.

B. Sharding Mechanisms in Blockchains

State sharding strategy. Many sharding chain-based blockchains, e.g., ByShard [18], [27], Monoxide [28], and OmniLedger [29], utilize the classic hash-based account allocation strategy, yielding massive cross-shard transactions due to ignoring interactions between accounts across shards.

Recent works have explored the one-to-one account assignment problem to reduce the number of cross-shard transactions. Some of them construct an account graph to capture the historical transaction patterns and place frequently interacting accounts in the same shard, e.g., TxAllo [38], Transformers [41], [48], and [49]. Shard Scheduler [50] adopts a transaction-level assignment method that determines the shard location of involved accounts when each transaction comes. Sliver [51] explores the transaction distribution problem in the sharding relay chain in a cross-chain scenario, which is not applicable to the general-purpose blockchains we are concerned with. In the one-to-one account assignment method, processing cross-shard transactions requires cooperation and multiple consensus rounds among shards [52], resulting in processing inefficiency.

To further improve the efficiency of cross-shard transaction processing, some studies [13], [33], [53], [54] adopt a one-to-many assignment method for partial accounts, i.e., some accounts can be replicated or fragmented and deployed to multiple shards for concurrent processing. For instance, Pyramid [33] proposes a layered sharding by leveraging bridge shards for storing redundant states and ledgers of multiple other shards for handling cross-shard transactions, which brings in a severe storage burden [55]. BrokerChain [13] introduces trusted *Brokers* accounts whose state is partitioned into many segments and placed into multiple shards for simultaneous execution. However, it relies on the strong assumption that *Brokers* always have sufficient tokens and requires an extra incentive mechanism to recruit sufficient *Brokers*.

Cross-shard verification. To reduce cross-shard communication complexity, some approaches [26], [34], [36], [37] use a single leader to transmit and append cross-shard messages. However, they need a cross-shard view change to resist censorship attacks [34], which does not work in the situation where the receiver shard does not know when the sender shard sends cross-shard messages to it. Moreover, the cross-shard message may suffer from long appending delays in the receiver shard in the case of cascading leader failures. Furthermore, Fynn et al. [56] present a client-driven protocol that allows smart contracts to move between shards. However, they do not clearly discuss how to guarantee liveness when the client is malicious.

State storage. Some studies explore state storage for sharding blockchains yet focus on problems different from ours. S-Store [40] proposes a scalable state data storage based on an

TABLE I
COMPARISON OF STATE-OF-THE-ART BLOCKCHAIN SHARDING MECHANISMS

Sharding mechanism	Features			Key components		Performance	
	Storage model	Workload-aware	No trust assumption	Account assignment	Cross-shard transaction processing	Scalability	Efficiency
Monoxide [28]	Chain-based	✖	✔	One-to-one	Multiple shards	✔	✖
BrokerChain [13]	Chain-based	✖*	✖	One-to-many	Multiple shards	✔	✔
Pyramid [33]	Chain-based	✖	✔	One-to-many	Multiple shards	Limited	✔
TxAllo [38]	Chain-based	✖*	✔	One-to-one	N/A	✔	✖
FS [34]	Chain-based	✖	✔	One-to-one	Multiple shards	✔	✖
Conflux [4]	DAG-based	N/A	N/A	N/A	N/A	✖	N/A
BullShark [7]	DAG-based	N/A	N/A	N/A	N/A	✖	N/A
SharDAG (Ours)	DAG-based	✔	✔	One-to-many	Single shard	✔	✔

* The account assignment is only adjusted in the reconfiguration phase but cannot be adaptively adjusted according to actual workloads in the consensus phase.

Aggregate Merkle B+ tree for dynamic sharding scaling. tMPT [57] proposes a trimmed *Merkle Patricia Trie* (MPT) to enable the newly arrived node to synchronize the shard state data quickly during shard reconfiguration. In contrast, we target scalable state storage to support efficient state reconfiguration.

Summary. Table I presents a comparison of some representative sharding mechanisms with our work. All studies mentioned above point out the importance of sharding for chain-based blockchains, but no effective solutions have been proposed towards scalable sharding for DAG-based blockchains. SharDAG differs from them in several aspects: (1) SharDAG conducts an in-depth analysis of realistic blockchain workloads and proposes an adaptive cross-shard avatar account caching mechanism to improve processing efficiency without extra trust assumptions; (2) SharDAG guarantees data consistency and security by a novel Byzantine resilient cross-shard verification mechanism; (3) SharDAG provides a novel two-tier state storage design for efficient state reconfiguration.

III. OBSERVATION AND MOTIVATION

Measurement setup. Our measurement study uses two time periods of Ethereum transactions[1] with block heights of 12M to 12.2M (Dataset1, Mar-Apr 2021) and 14M to 14.2M (Dataset2, Jan-Feb 2022), as accounts were relatively active during these periods [58]. Our statistics show that over 70% of the transactions are asset transfers, including normal transfers and contract-based token transfers. We further analyze the latest 50,000 blocks and confirm that asset transfer transactions still dominate, exceeding 50%. Hence, in this section, we focus on analyzing the characteristics of asset transfer transactions. Each transaction in our dataset contains the transaction details and the associated account states before and after execution, obtained by our *replay* tool.

Most sharding systems [28], [30] handle cross-shard transactions based on a one-to-one account assignment mode. All transactions updating the state of the same account need to be validated and committed serially in a particular shard, resulting in high confirmation latency and frequent cross-shard communication, as shown in Fig. 3(a). This one-to-one mode does not take full advantage of the parallelism of sharding.

[1]Existing DAG-based and sharded blockchains still employ Ethereum workloads for test because of lacking publicly available transaction data.

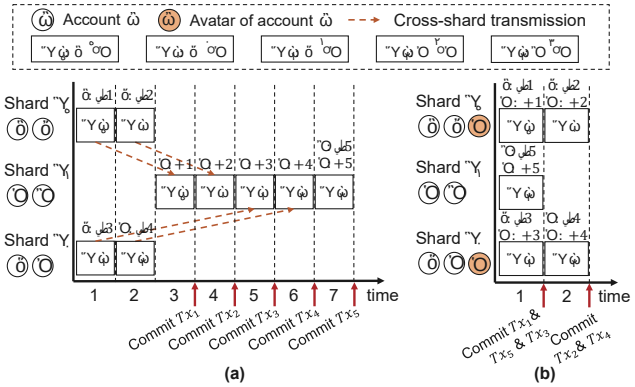


Fig. 3. Cross-shard transaction processing (a) without vs. (b) with cross-shard avatar account caching. (a) Serial commit in a single shard (Monoxide [28]). (b) Parallel commit in multiple shards (SharDAG).

To explore a feasible parallel execution and commit scheme, we characterize the transaction behavior of accounts. We track the asset received by an account from transaction activities and analyze the time interval between when it is received and spent, termed *silence before spent*. Fig. 4 illustrates the distribution of *silence before spent*. Each block contains 1000 transactions. An interesting observation is that only a small proportion of assets is spent soon, with the majority remaining silent for a considerable period before being spent. For instance, in Dataset2, less than 3.5% of assets are spent within 100 blocks, with over 94.1% remaining silent for 200 blocks and over 84.0% remaining silent for 300 blocks.

Observation 1: *Silent assets: Most of the received assets will undergo a long silence before being spent.*

Implication 1: *During the silence, an asset is dispensable because the transactions initiated by the account do not depend on it. If we allow the transferred asset in a cross-shard transaction to be temporarily cached in an account avatar, then transactions can be committed in parallel, thus fully leveraging the parallelism of sharding and significantly reducing cross-shard communication (Fig. 3(b)). In addition, as long as the avatars are withdrawn and aggregated with the account securely before the silence ends, the account states will not be lost, and transaction abortion can also be avoided.*

Periodic state reconfiguration has gradually become an essential component in improving the performance of sharded

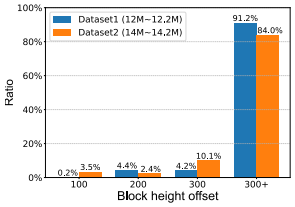


Fig. 4. Distribution of silence before spent

TABLE II
RATIO OF ACTIVE AND MIGRATED ACTIVE ACCOUNTS UNDER R-METIS-BASED STATE RECONFIGURATION METHOD

Dateset	Epoch	Active (Ratio)	Active migrated (Ratio)
Dataset1 (12M-12.2M)	0	90223 (100%)	78996 (100%)
	9	91665 (15.2%)	77126 (100%)
	19	93831 (8.6%)	81639 (100%)
	29	102044 (6.4%)	87548 (100%)
Dataset2 (14M-14.2M)	0	97135 (100%)	84960 (100%)
	9	97606 (16.5%)	85021 (100%)
	19	97345 (9.5%)	86694 (100%)
	29	97446 (6.6%)	84884 (100%)

blockchains [13], [38], [41], [48]. Typically, during the reconfiguration phase, a trusted third party or distributed shard committee generates a reconfiguration strategy (containing state roots of all shards for verification) and notifies all shards. Each shard then reconstructs its underlying state storage, i.e., cleans up the outgoing accounts, informs the target shard of their states and Merkle proofs actively or passively, and inserts the incoming accounts after verification [41].

The state storage organization affects state reconfiguration efficiency. Typically, each shard organizes state using a monolithic MPT or its variants [13], [40], [41]. When migrating account states across shards, it is necessary to delete nodes in the monolithic MPT and generate proofs, which involves hash operations of all nodes in the path from the leaf node to the root, resulting in long delays. To explore a more reasonable way to organize states, we investigate the characteristics of the accounts migrated during the reconfiguration phase. Many approaches [38], [48], [50] only reassign accounts that appear in recent transactions, i.e., active accounts. This avoids migrating dormant accounts that are not beneficial to the future and reduces reconfiguration costs [38], [50]. We measure the number and ratio of average active and migrated active accounts per shard using the classic R-METIS [48] [2] under a setting of 8 shards and 1M transactions per epoch (Table II). The results show that after running 30 epochs, active accounts only account for 6.4% (or 6.6%) of the total accounts, and all the migrated accounts in each epoch are from active accounts.

Observation 2: *Active accounts: Active accounts in each epoch are only a small fraction of the total accounts, and most migrated accounts during state reconfiguration are active.*

Implication 2: *Organizing the active accounts into a separate MPT is a better approach toward efficient state reconfiguration because most operations fall on the lightweight MPT.*

IV. SHARDAG OVERVIEW

A. System and Threat Model

System model. In SharDAG, there are N nodes in the system. According to an unbiased and unpredictable distributed randomness [27], these nodes are evenly assigned to S shards, each containing n nodes. Like other systems [29], [33], we assume the partially synchronous communication model with optimistic, exponentially increasing time-outs. As for cross-shard transmission, we assume that cross-shard communication occurs randomly, and the receiver shard does not know when the sender shard will send cross-shard messages.

[2]R-METIS is a variant of the classic graph partitioning algorithm METIS. It only uses transactions in the current epoch to construct the transaction graph.

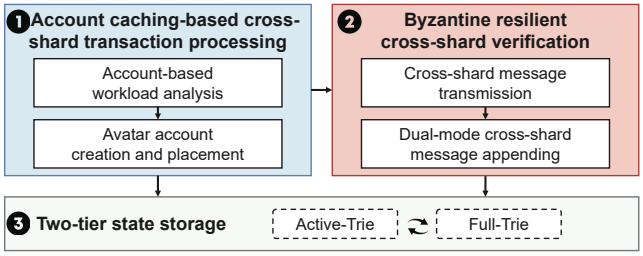


Fig. 5. Architecture of SharDAG

Threat model. The system resilience is denoted as t. In other words, tN nodes are controlled by a Byzantine adversary, and the rest are honest nodes. Note that t is less than $1/3$ to meet the assumption that the malicious nodes in each shard $f < n/3$ after shard formation. Honest nodes always behave loyally, while malicious nodes behave arbitrarily, e.g., sending tampered messages, refusing to respond, or conspiring in an attack. However, the Byzantine adversary is computationally bounded and cannot impersonate honest nodes.

B. Goals

The overall goal of SharDAG is to improve the storage scalability of DAG-based blockchains through adaptive sharding. Nodes only need to store a part of the whole DAG ledger and state. Moreover, SharDAG achieves the following goals with respect to performance and security.

Efficient cross-shard transaction processing. The sharding mechanism should adapt to actual cross-shard workloads to achieve low cross-shard frequency and high cross-shard transaction processing efficiency.

Byzantine resilient cross-shard verification. Cross-shard verification needs to guarantee safety, robust liveness, and low appending delays.

Efficient state storage. The state storage should be well organized to support efficient state reconfiguration.

C. SharDAG Architecture

SharDAG is adapted to asset transfer scenarios, and we assume that the workload is characterized by the two critical features observed in Section III. Fig. 5 illustrates the architecture of SharDAG, which contains three key modules:

Account caching-based cross-shard transaction processing. Based on the silent assets feature of real-world workloads, we propose a cross-shard avatar account caching mechanism that extends the traditional one-to-one account assignment mode to a one-to-many mode. Thus, there are two kinds of accounts in SharDAG: 1) *Primary account.* Each entity has a unique primary account. All primary accounts are assigned to shards using a one-to-one strategy, and each shard stores only a subset of them. 2) *Avatar account.* An entity may have multiple avatar accounts spread across different shards except for the shard where its primary account is located. All accounts

from an entity are identified by an identical address. SharDAG analyzes the accounts involved in cross-shard transactions and creates avatars for the asset receiver to cache its positive incremental states, i.e., the assets transferred to it. Then the avatar participates in cross-shard transactions on behalf of its primary account. Thus, all cross-shard transactions can be processed in one shard without cross-shard communication.

Byzantine resilient cross-shard verification. The incremental states of an entity are scattered across avatar accounts in multiple shards. Avatar accounts are periodically withdrawn and aggregated to their primary accounts via cross-shard messages, i.e., avatar-epoch. To ensure state consistency, we devise a Byzantine resilient cross-shard verification mechanism in two steps. SharDAG utilizes multiple nodes to ensure strong liveness during cross-shard message transmission without complex remote recovery. In addition, we leverage the concurrent block generation nature of DAG-based consensus and design a dual-mode cross-shard message appending to reduce appending delays and ensure liveness.

Two-tier state storage. Since most migrated accounts are active, we devise a two-tier state storage to support efficient state reconfiguration. Specifically, two-tier state storage maintains local primary accounts and is organized into two MPTs, a *Full-Trie* and an in-memory *Active-Trie*. *Full-Trie* stores all accounts, and *Active-Trie* only caches active accounts appearing in recent transactions. *Active-Trie* is lightweight since active accounts are only a fraction of total accounts. Hence, most account migration operations are performed on the lightweight *Active-Trie* rather than the vast *Full-Trie*, thus improving state reconfiguration efficiency.

Similar to [13], we divide the system runtime into consecutive epochs, each containing a consensus and a reconfiguration phase. In the consensus phase, user transactions are sent to the shard where the sender's primary account is located for processing to prevent replay attacks. Each shard runs a DAG-based BFT consensus [7] independently to agree on the block order in the local DAG ledger. The sorted blocks are then executed serially by the execution engine, yielding the same state updates among all honest nodes. Cross-shard transactions are processed using our account caching design, and the adaptively-created avatar accounts are cleared and aggregated with their primary accounts every avatar-epoch. In the reconfiguration phase, the primary accounts are redistributed between shards via the R-METIS method [48], and nodes reach an agreement on the decisions and migrate accounts through the state reconfiguration protocol proposed in Transformers [41]. During migration, nodes reconstruct the two-tier state storage for the next epoch. In addition, dormant accounts are cleared from *Active-Trie* to keep it lightweight.

V. System Design

A. Cross-shard Avatar Account Caching

SharDAG creates avatar accounts adaptively according to actual cross-shard transactions and commits cross-shard transactions in different shards in parallel. Our cross-shard account caching supports the following two types of transactions.

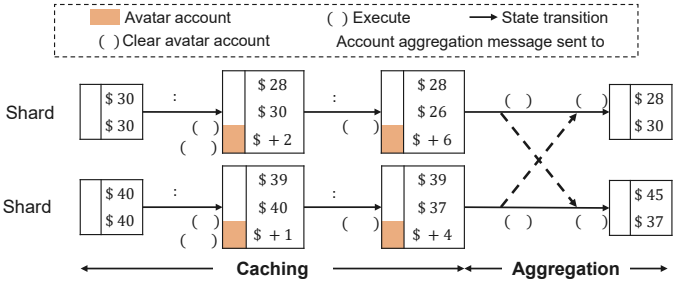


Fig. 6. Running example of cross-shard account caching and aggregation

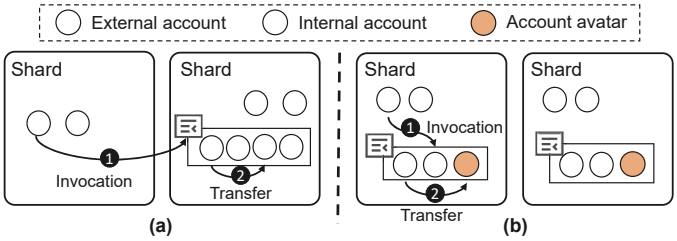


Fig. 7. Processing contract-based single-step transfer transactions via (a) traditional method vs. (b) cross-shard avatar account caching (SharDAG)

Normal transfer transactions. When processing an original cross-shard normal transfer transaction, if the receiver is unavailable locally, the node first creates an avatar account for it with an initial balance of 0 to maintain the receiver's incremental states within the current avatar-epoch. For instance, in the example in Fig. 6, S_1 creates an avatar C' for account C when processing Tx_1. Subsequent transactions involving the receiver C can be executed as intra-shard transactions without cross-shard interaction. Since the Byzantine resilient cross-shard verification mechanism guarantees the security of avatar account aggregation, the original cross-shard transactions can be committed securely within a single execution with shorter confirmation latency. Meanwhile, transactions issued by the receiver in the current avatar-epoch (e.g., Tx_3) can be executed and committed successfully with a high probability, based on the feature of silent assets observed in Section III.

Contract-based transfer transactions. In a contract-based single-step transfer transaction, an external account invokes a contract to transfer tokens from its internal account to another internal account. In the traditional one-to-one assignment mode, the entire contract account, encompassing contract code and all internal account states, is stored in a specific shard. When the external and contract account are located in different shards, the transaction cannot be committed in a single shard and requires a single cross-shard invocation (Fig. 7(a)). In SharDAG, we devise a fine-grained allocation for contract accounts. For a contract account, each shard stores an avatar of it, which includes the contract code and the states of the internal accounts corresponding to the external accounts in the local shard. Similarly, we create avatars adaptively for internal accounts belonging to other shards according to actual cross-shard workloads. Thus, contract-based single-step cross-shard transfer transactions can be executed and committed entirely within a shard without cross-shard invocations (Fig. 7(b)).

Moreover, avatar accounts are created adaptively and can be aggregated asynchronously since they only keep incremental states, which is different from Pyramid [33], in which the bridge shard redundantly stores the entire state of other shards. Thus, SharDAG is efficient in terms of storage overhead.

B. Byzantine Resilient Cross-shard Verification

At the end of the avatar-epoch, the state of avatar accounts must be securely aggregated with their primary accounts to ensure state consistency across shards. Specifically, if a sender shard has cached avatar accounts, it should clear them, generate cross-shard account aggregation messages carrying the state of avatar accounts, and send them to the shards where the primary accounts belong. After receiving an aggregation message, the receiver shard should verify the message and append it to the DAG only when making sure that the sender shard indeed agrees with it. Finally, the receiver shard reaches a consensus and performs account aggregation. An example can be obtained in the right part of Fig. 6.

The above process can be abstracted as a generic cross-shard verification in sharding DAG-based blockchains, i.e., a sender shard sends an agreed message to a receiver shard, which appends it to its local DAG ledger for further consensus and processing. To ensure safety and liveness in the presence of malicious nodes, the cross-shard verification mechanism must be elaborately designed. Thus, before detailing the cross-shard avatar account aggregation workflow, we first introduce its underlying Byzantine resilient cross-shard verification mechanism, which consists of the following two critical steps:

Cross-shard message transmission. The traditional single leader-based method [34] cannot ensure liveness when suffering from cross-shard transaction censorship attacks since nodes cannot detect the malicious sender by setting timers. In SharDAG, we choose $2f+1$ senders and $f+1$ receivers for cross-shard message transmission, sufficient to ensure safety and liveness while avoiding redundant transmission. The reasons are given in Section VI. These nodes are randomly selected based on the digest of unpredictable cross-shard messages to defend against attacks and balance loads.

Dual-mode cross-shard message appending. Through cross-shard message transmission, $f+1$ receivers receive the valid message, among which at least one honest node. Once an honest node appends the message to its local DAG, the intra-shard DAG-based consensus ensures that the message will eventually appear in the local DAGs of all honest nodes.

In SharDAG, the single leader-based method [34] is not suitable due to its inability to leverage the concurrent block generation nature and potential long appending delays in case of cascading leader failures. Allowing all receivers to append the message provides strong liveness and low delay but leads to duplicate appending by honest nodes, resulting in redundant transactions in the DAG ledger and affecting throughput. To balance the appending delay and throughput, we design a dual-mode solution containing optimistic and pessimistic appending. In the optimistic appending, we choose multiple nodes to perform appending to improve success probability. In the pessimistic appending, other receivers are responsible for appending to ensure liveness when optimistic appending fails. Thus, in the worst case, the message can be appended after one failure, thus shortening appending delays. Then we determine the optimal number of nodes for optimistic appending to be 2 by theoretical analysis, as detailed in Section VI.

Let $\mathrm{Digest}(m)$ be the digest of message m. The shard S_j appends the received valid cross-shard message m_j to local DAG using the following dual-mode scheme:

a) Optimistic appending: Two nodes are elected for optimistic appending based on $\mathrm{Digest}(m_j)$ from all receivers. Then the honest node packs m_j into a block b_j and appends it to its local DAG. Other receivers confirm that m_j is appended by tracking their own DAG ledger. Eventually, m_j will appear in the same location in the DAG of each honest node in S_j.

b) Pessimistic appending: To detect failures and recover quickly, each honest receiver sets a timer for m_j. If the timer expires before the node observes a valid m_j in its local DAG, which means optimistic appending fails, the honest receiver immediately performs appending m_j, ensuring liveness. The timeout can be set according to the actual network environment. According to [59], an empirically recommended setting is to ensure at least 20 times the one-way network latency.

Fast abort. Although dual-mode appending selects an optimal number of nodes, it cannot completely avoid redundant appending. To further reduce redundancy, we introduce *fast abort*. Due to network transmission delays, the progress of each receiver may differ. Some honest receivers may have attached m_j to DAG after a timeout, while others may still collect messages from S_1 (verify the validity of m_j) or wait for a timeout (check if the optimistic appending succeeds). When slow honest receivers observe that m_j has been attached correctly, they can immediately abort the processing of m_j. This scheme prevent some honest nodes from repeatedly appending m_j to the DAG, thus alleviating performance degradation.

C. Cross-shard Avatar Account Aggregation

After processing the last block b_i of avatar-epoch e, shard S_i initiates aggregation for locally-cached avatar accounts. Let A_i be the set of avatar accounts created in e by S_i. Let $\mathrm{Shard}(a)$ denote the shard of the primary account of avatar a and $\mathrm{State}(a)$ denote the state of avatar a. Let $S = \{\mathrm{Shard}(a) | a \in A_i\}$. S_i is the sender shard, and each shard $S_j \in S$ is a receiver shard. The cross-shard account aggregation procedure involves four steps, as shown in Fig 8.

Step 1: Generate cross-shard account aggregation messages in the sender shard. For each S_j in S, each node in S_i generates an aggregation message m_j based on the set $\{\mathrm{State}(a) | a \in A_i \wedge \mathrm{Shard}(a) = S_j\}$ and an incremental aggregation serial number (e.g., avatar-epoch id) to resist replay attacks. Then each node in S_i clears all avatar accounts.

Step 2: Send each cross-shard account aggregation message from the sender shard to its receiver shard. For each m_j, $2f+1$ nodes in S_i are elected as senders, and $f+1$ nodes in S_j are elected as receivers according to $\mathrm{Digest}(m_j)$. Then each sender signs and sends m_j to its receivers.

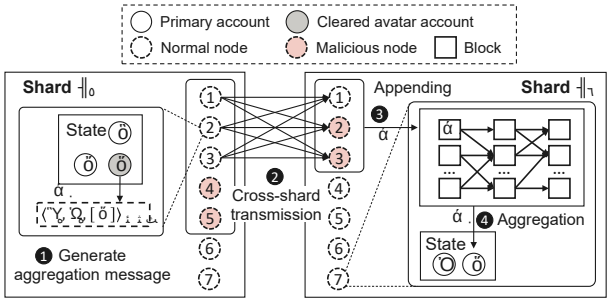


Fig. 8. Illustration of cross-shard avatar account aggregation workflow at the end of avatar-epoch e of Shard S_1. Shard size: $n = 3f + 1$, $f = 2$.

Step 3: Cross-shard account aggregation message verification and appending in the receiver shard. Once a node in S_j receives m_j, it first verifies the correctness of the signature. After receiving $f + 1$ identical valid m_j, the node confirms that the honest nodes in S_i have agreed on m_j. Then it constructs an aggregated signature for m_j and appends m_j to the local DAG ledger using the dual-mode cross-shard message appending mechanism mentioned above. After that, m_j will appear in a block b_j in the local DAG of S_j.

Step 4: Perform account aggregation in the receiver shard. Each node in S_j executes m_j deterministically after consensus. For each avatar account a that appears in m_j, each node aggregates it into its primary account.

Noting that cross-shard account aggregation is asynchronous. After the current avatar-epoch e, a shard can advance to avatar-epoch $e+1$ to process new transactions without waiting for aggregation messages belonging to avatar-epoch e from other shards. Moreover, since **Step 2** does not affect the execution of subsequent blocks, it can be performed in parallel with the execution of the first block in avatar-epoch $e + 1$.

Aggregation interval. A long account aggregation interval yields lower cross-shard communication but also accumulates account assets in avatars scattered across multiple shards. Due to such asset dispersion, the primary account's balance may not be sufficient to cover the transfer amount in new transactions, resulting in transaction abortion. Hence, an appropriate cross-shard account aggregation interval is essential for balancing cross-shard overhead and transaction abortion. In SharDAG, it is set to 100 blocks based on the observations from realistic workloads in Section III.

D. Two-tier State Storage

Two-tier state storage can accelerate state reconfiguration by organizing active accounts into a lightweight *Active-Trie*. We then discuss critical operations on it. During the consensus phase, the execution engine accesses the two-tier storage to obtain the primary account's state. If the account is found in *Active-Trie*, which means it is active, the two-tier storage returns its state in *Active-Trie*; otherwise, it returns the state in *Full-Trie* and inserts it into *Active-Trie*. Moreover, all updates are directly written to *Active-Trie*. Hence, *Active-Trie* always holds the latest states of active accounts in the current epoch.

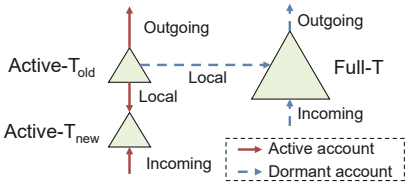


Fig. 9. Updating the two-tier state storage during the reconfiguration phase

During the reconfiguration phase, nodes migrate accounts across shards and reconstruct the state storage for the next epoch. In addition, *Active-Trie* will gradually expand as the system runs since some accounts become dormant. To keep *Active-Trie* lightweight, SharDAG cleans up dormant accounts from it every epoch and moves them back to *Full-Trie*. Considering that the removing operation is more expensive than the getting and inserting operations [60], SharDAG creates a new empty *Active-Trie* for the next epoch. Fig. 9 shows all operations on the two-tier storage in the reconfiguration phase. Based on the verifiable reconfiguration results received from the main shard [13], [41], each shard can identify whether the outgoing (incoming) account is active or dormant. SharDAG first obtains the states and proofs for outgoing active and dormant accounts from *Active-Trie*$_{old}$ and *Full-Trie*, respectively. Note that a small number of outgoing dormant accounts may need to be obtained from *Active-Trie*$_{old}$ when it has just become dormant in the current epoch. Then the outgoing dormant accounts and old snapshots of the outgoing active accounts are removed from *Full-Trie*. Incoming active and dormant accounts can be inserted into *Active-Trie*$_{new}$ and *Full-Trie* simultaneously after verification. In addition, SharDAG fetches the states of local accounts from *Active-Trie*$_{old}$ and inserts those active in the current epoch into *Active-Trie*$_{new}$ and the dormant ones into *Full-Trie*. *Active-Trie*$_{old}$ is safely deleted when all states in it are fetched. Since most of the operations fall on the lightweight *Active-Trie*, it takes less time to migrate out and in accounts, and less data is transmitted across shards, thus enabling an efficient state reconfiguration.

VI. Security Analysis

A. Correctness of SharDAG

This section provides the safety and liveness analysis of SharDAG. We do not discuss the shard formation phase here, as it is orthogonal to the core design of this paper. The proof builds upon the following postulate: based on a random and bias-resistant shard formation protocol, SharDAG system has a negligible failure probability, that is, the probability that the fraction of malicious nodes in each shard exceeds the intra-shard consensus threshold at any time is negligible.

Theorem 1. *Let S_1 and S_2 be two shards. S_1 synchronizes its cached avatar accounts to S_2 using the cross-shard account aggregation mechanism via an account aggregation message m. This mechanism provides the following properties:*

- *Safety: If an honest node in S_2 executes message m, then m contains the signature of at least one honest node in S_1 (Validity). Honest nodes in S_2 will receive the same*

message m and perform account aggregation, resulting in the same state updates (Consistency).

- *Liveness: Honest nodes in S_2 will eventually receive message m and complete account aggregation (Termination).*

Proof. Validity. The malicious senders may behave arbitrarily, such as tampering with m, intentionally not sending m. However, they cannot forge signatures. Since the aggregated signature of message m is generated according to $f+1$ distinct signatures and there are at most f malicious nodes in S_1, at least one signature comes from an honest node in S_1. Thus, cross-shard avatar account aggregation satisfies validity.

Consistency. According to the liveness property proved below, at least one honest receiver appends the valid message m to the local DAG. Since the intra-shard DAG-based consensus satisfies censorship resistance and safety, the message m can be committed, and all honest nodes in S_2 will execute m deterministically, leading to the same state update. Hence, cross-shard avatar account aggregation satisfies consistency.

Liveness. Given less than $1/3$ malicious nodes in each shard, the intra-shard DAG-based BFT consensus can provide safety. In other words, the honest nodes in S_1 will execute blocks in the same order, so the state of the cached avatar account is identical. Thus, m generated by honest nodes in S_1 are identical. Since there are at most f malicious nodes in S_1, at least $f+1$ nodes correctly send the same m to the pre-selected $f+1$ receivers in S_2, ensuring the sending liveness. Moreover, there is at least one honest node among $f+1$ receivers. Hence, S_2 will eventually receive m and confirm its validity. If nodes in optimistic appending fail to pack m due to malicious behavior, at least one honest node in pessimistic appending will append m to the local DAG after a timeout, ensuring the appending liveness. As the intra-shard DAG-based BFT consensus can achieve liveness, m will eventually appear in the DAG ledger of all honest nodes in S_2 and be processed. Hence, cross-shard avatar account aggregation satisfies liveness. □

B. Analysis of Dual-mode Cross-shard Message Appending

To make a trade-off between appending delay and throughput, we theoretically analyze the dual-mode cross-shard message appending process. In the following analysis, we assume that honest nodes in a shard are relatively synchronized, i.e., the time difference between their appending messages is less than the network propagation delay. Hence, a message may be appended by multiple honest nodes repeatedly.

Table III summarizes the relevant variables used in the analysis. To perform the analysis, we model two steps: 1) randomly select r nodes from n nodes in a shard as cross-shard message receivers; 2) randomly select l nodes from r receivers for optimistic appending. We establish the failure probability of optimistic appending as a function of l. Higher failure probability implies a higher likelihood of turning to pessimistic appending, leading to longer appending delays. In addition, we analyze the relationship between the redundant appending probability and l. Higher redundant append probability implies that duplicate messages are appended, affecting throughput.

TABLE III
NOTATION TABLE

Notation	Description	Requirements
f	Number of Byzantine nodes in each shard	
n	Number of nodes in each shard	$n = 3f+1$
r	Number of cross-shard receivers	$r = f+1$
k	Number of Byzantine receivers	$k = 0, 1, \cdots f$
l	Number of nodes in optimistic appending	$l = 1, 2, \cdots r$
j	Number of Byzantine nodes in optimistic appending	
$(l, r-l)$	Parameter settings of the dual-mode appending mechanism	

We only consider honest receivers since we cannot prevent malicious receivers from adding any messages to the DAG. We aim to find an optimal l that minimizes both probabilities.

Since the content of a cross-shard message is unpredictable, SharDAG uses its digest as a source of randomness for both steps. Hence, the number of Byzantine receivers, k, is a random variable following hypergeometric distribution $k \sim H(n, f, r)$. Let X be a random variable. Then:

$$\Pr[X = k] = \frac{\binom{f}{k}\binom{n-f}{r-k}}{\binom{n}{r}} \tag{1}$$

Similarly, given k, the number of malicious nodes in optimistic appending also follows hypergeometric distribution $j \sim H(r, k, l)$. Let Y be a random variable. We have:

$$\Pr[Y = j] = \frac{\binom{k}{j}\binom{r-k}{l-j}}{\binom{r}{l}} \tag{2}$$

The optimistic appending fails when all nodes in optimistic appending are Byzantine. Therefore, the failure probability of optimistic appending is given by:

$$\Pr[\text{failure}] = \Pr[X = k] \cdot \Pr[Y = l] \tag{3}$$

When there is more than one honest node in optimistic appending, optimistic appending succeeds redundantly. The redundant appending probability of optimistic appending is:

$$\begin{aligned}\Pr\left[\text{redundancy}_{\text{opt}}\right] = \Pr[X = k] \cdot \\ (1 - \Pr[Y = l] - \Pr[Y = l-1])\end{aligned} \tag{4}$$

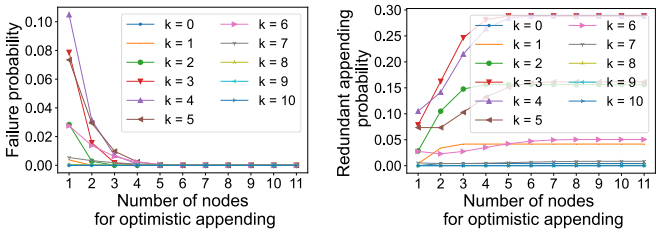
If optimistic appending fails, turn to pessimistic appending. If there are at least two honest nodes in pessimistic appending, then pessimistic appending succeeds redundantly. Thus, the redundant appending probability of pessimistic appending is:

$$\Pr\left[\text{redundancy}_{\text{pes}}\right] = \begin{cases} \Pr[\text{failure}] & r-k \geq 2 \\ 0 & \text{others} \end{cases} \tag{5}$$

Therefore, the redundant appending probability of the dual-mode cross-shard message appending mechanism is:

$$\begin{aligned}\Pr[\text{redundancy}] = &\Pr\left[\text{redundancy}_{\text{opt}}\right] \\ &+ \Pr\left[\text{redundancy}_{\text{pes}}\right]\end{aligned} \tag{6}$$

Based on equations (3) and (6), Fig. 10 visualizes the relationship between the two probabilities and the number of nodes in optimistic appending, l. We observe that the failure



(a) Failure probability of optimistic appending (b) Probability of redundant appending

Fig. 10. Effect of the node number in the optimistic appending on the failure probability and the redundant appending probability when $f = 10$

probability of optimistic appending is inversely proportional to l, while the redundant appending probability is the opposite. Both probabilities maintain low when $l = 2$. Hence, we claim that using a $(2, f - 1)$ dual-mode scheme is an appropriate choice, achieving both low appending delay and avoiding throughput degradation.

Note that the above analysis focuses on a relatively synchronous case. Actually, the progress of nodes may be different due to network delay. A more comprehensive analysis is to consider the above difference and the fast abort scheme we proposed in Section V. We consider this as our future work.

VII. Evaluation

A. Experimental Setup

Implementation. For performance evaluation, we implement a prototype of SharDAG in Rust based on BullShark [7], the state-of-the-art DAG-based BFT consensus protocol. Specifically, SharDAG uses BullShark as the intra-shard consensus and executes transactions in a shard serially.

Testbed. Our testbed consists of 17 virtual machines on Alibaba Cloud, including a VM located in Hong Kong serving as the benchmark client and 16 VMs evenly distributed among four different regions (i.e., Tokyo, Frankfurt, Singapore, and Seoul) to run sharding nodes. These VMs are connected via 200 Mbps network links. Each VM is equipped with a 16-core 3.5-GHz Intel Xeon Platinum 8369B CPU with 32 GB RAM, running Ubuntu 20.04 LTS. We deploy up to 16 shards using up to 160 sharding nodes. By default, we run 10 nodes in each VM. We simulate the cross-shard transaction censorship attack by randomly selecting f nodes in each shard as cross-shard Byzantine nodes, which behave honestly within the shard but maliciously across shards [34], i.e., do not forward, verify, and package cross-shard messages. Similar to [59], we set the timeout in our dual-mode appending as 2.5 sec (nearly 20 times the one-way network latency in our WAN environment).

Baselines. We implement a DAG-based blockchain prototype based on BullShark as a non-sharding baseline (*BullShark* for short). In terms of sharding comparison, we implement two non-adaptive sharding baselines on top of *BullShark*: *Monoxide-BS* [28] and *BrokerChain-BS* [13]. Monoxide is a classic sharding scheme that employs the relay mechanism for cross-shard transaction processing. BrokerChain is a state-of-the-art sharding scheme that utilizes the broker-based approach to process cross-shard transactions. For a fair comparison, *Monoxide-BS* and *BrokerChain-BS* execute transactions serially and adopt the single leader-based cross-shard verification [34], as well as the monolithic MPT for state storage. We select the top 40 (the default value in BrokerChain) most frequently accessed accounts in our dataset as broker accounts and distribute them on all shards. Besides, we assume that they behave honestly in coordinating cross-shard transactions.

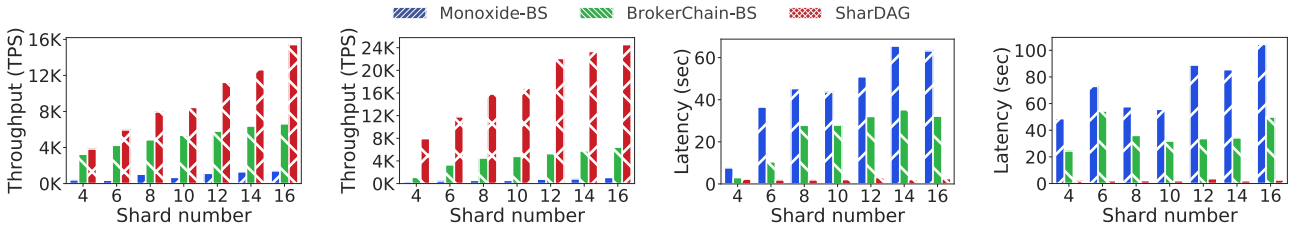
Workload. We still use the same Ethereum workload as used in Section III. Since the datasets from the two time periods exhibit similar characteristics, we use the data from one period for evaluation, specifically block height 12M to 12.2M. By default, we use the hash-based strategy [28] to assign (primary) accounts to shards. Once all the nodes start, the benchmark client continuously fetches transactions from the dataset in chronological order at a fixed rate and distributes them to each node in each shard. Since each node can generate blocks in sharding DAG-based blockchains, we set the total transaction input rate to $K \times$ number of nodes, where K is the input rate per node. Similar to [7], we set a maximum block size of 500 KB and a transaction size of 512 B.

B. Performance Comparison with Sharding DAG

We compare the performance of different cross-shard transaction processing mechanisms of SharDAG, *Monoxide-BS*, and *BrokerChain-BS* under varying shard numbers and input rates.

Throughput. Fig. 11(a) and Fig. 11(b) depict the end-to-end throughput. *Monoxide-BS* achieves the lowest throughput, with less than 1,500 TPS. This is because almost all transactions are cross-shard and need to be relayed. The transaction committed rate is very low with constantly injected transactions. *BrokerChain-BS*'s throughput is better than *Monoxide-BS*'s and rises with the shard number because the broker mechanism reduces the number of cross-shard transactions. However, the throughput grows only slightly as K rises to 200 txs/s since more cross-shard transactions need to be processed, requiring more communication and message verification among brokers and shards. In contrast, SharDAG's throughput increases with the shard number and input rate. In particular, when deployed with 16 shards, SharDAG achieves a throughput of $\sim$ 24K TPS, which is 3.8$\times$ higher than that of *BrokerChain-BS*. This is achieved through adaptive cross-shard avatar account caching, which enables cross-shard transactions to be processed and committed within a single shard, thus reducing cross-shard interactions and improving throughput.

Latency. Fig. 11(c) and Fig. 11(d) show the end-to-end latency. The latency in *Monoxide-BS* is higher than 40 seconds with more than 6 shards and increases with K. This is due to a large number of new transactions consuming the throughput and causing longer processing latency for relay transactions. *BrokerChain-BS*'s latency is more than 20 seconds because cross-shard transactions still need to be executed in two shards with the coordination of brokers. In contrast, SharDAG achieves the lowest confirmation latency, less than 5 seconds, resulting in an 18$\times$ improvement compared to *BrokerChain-BS* under 16 shards and K = 200 txs/s.



(a) End-to-end throughput (K = 100 txs/s) (b) End-to-end throughput (K = 200 txs/s) (c) End-to-end latency (K = 100 txs/s) (d) End-to-end latency (K = 200 txs/s)

Fig. 11. Comparison of overall performance under varying shard number and input rate

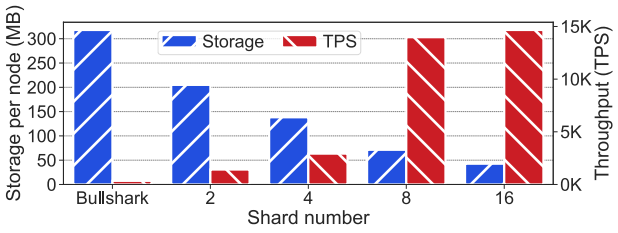
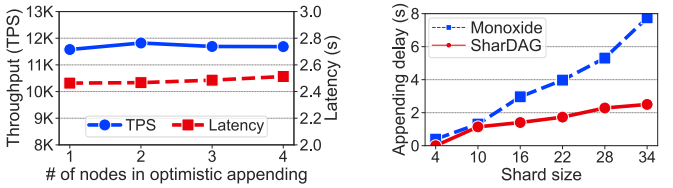


Fig. 12. Comparison of storage overhead and end-to-end throughput



(a) Performance of SharDAG under varying numbers of nodes in optimistic appending (8 shards, each with 10 nodes) (b) Cross-shard message appending delay under varying shard size

Fig. 13. Performance of Byzantine resilient cross-shard verification

Fig. 14 shows the system performance under 16 shards and a decreasing ratio of asset transfer transactions. The designs of SharDAG and *BrokerChain-HS* are applicable to asset transfer transactions. When the ratio of asset transfer transactions decreases, both of their performance suffer. However, SharDAG still outperforms the baselines. Specifically, SharDAG achieves a throughput 1.7× higher than *BrokerChain-HS* and 3.6× higher than *Monoxide-HS*, respectively, when the ratio of asset transfer transactions is 50%.

C. Performance Comparison with Non-sharding DAG

To evaluate the storage and performance scalability of SharDAG over the non-sharding baseline *BullShark*, we measure the storage cost per node and throughput in *BullShark* and SharDAG with varying shard numbers. Since all nodes can generate blocks, we fix the total number of nodes to 64 to avoid its interference on performance comparisons. We then inject 2M transactions at an input rate of K=2,000 txs/s.

Storage overhead. We first measure the storage overhead, as shown in Fig. 12. In *BullShark*, each node needs to store the whole DAG ledger, leading to high storage overhead. In contrast, in SharDAG, each node only needs to store the local DAG ledger of its own shard. As the shard number increases, the storage overhead per node is reduced. In particular, SharDAG achieves 7.5× lower storage overhead than *BullShark* when deploying 16 shards, which illustrates the high storage scalability of SharDAG. In addition, we measure the storage cost of avatar account caching. Under 16 shards, the average and maximum storage costs are approximately 195 KB and 2626 KB, respectively. Given that the primary account's storage cost is at a scale of GB [60], the extra storage cost introduced by avatar account caching is acceptable for nodes.

Throughput. Fig. 12 also shows that the TPS of SharDAG grows linearly with the shard number and is significantly higher than *BullShark*. For instance, SharDAG achieves nearly 40× achievement, with 14,666 TPS under 16 shards compared to *BullShark*'s 343 TPS. This is because although each node can generate blocks in *BullShark*, consensus involves all nodes, and each node needs to process all transactions, resulting in low end-to-end throughput. In contrast, nodes in SharDAG only need to participate in intra-shard consensus and process local transactions, thereby reducing the computation and communication overhead and improving throughput.

D. Performance of Byzantine Resilient Cross-shard Verification

We first explore the performance of SharDAG under varying numbers of nodes in optimistic appending. Fig. 13(a) shows that as the number of nodes in optimistic appending increases, the performance in terms of both throughput and transaction latency deteriorates, and the best performance is achieved when the number of nodes in optimistic appending is 2, consistent with the theoretical derivation in Section VI-B.

We measure the appending delay of the dual-mode cross-shard message appending of SharDAG and the traditional single leader-based approach used by *Monoxide-BS* under 2 shards and varying shard sizes. We do not compare with *BrokerChain-BS* because the broker mechanism cannot guarantee appending liveness when the broker is malicious. Fig. 13(b) shows that the dual-mode method consistently outperforms the single leader-based approach. As shard size increases, the latter's appending delay increases significantly while the former's rises slowly. The reasons are as follows. The single leader-based approach selects only one node at a time to pack cross-shard messages. When nodes fail consecutively, the messages experience long appending delays. On the contrary, SharDAG's dual-mode method leverages the concurrent block generation nature of DAG consensus and selects two nodes to

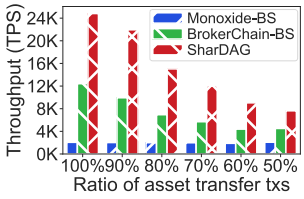
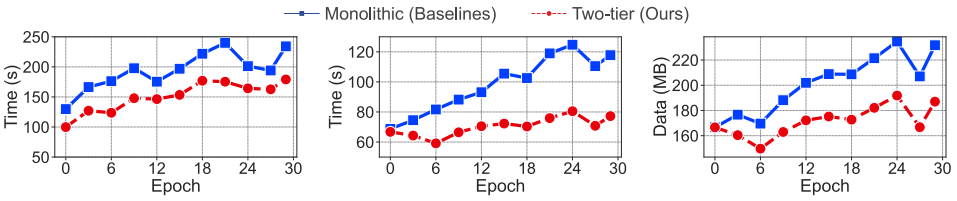


Fig. 14. Performance under varying ratios of asset transfer transactions



(a) Latency for execution (b) Latency for migration (c) Data transmission

Fig. 15. Performance comparison of different state storage in (a) the consensus phase and (b-c) the reconfiguration phase

pack in the optimistic mode, improving success probability. In the worst case, the message can be appended in pessimistic mode, experiencing only one failure.

E. Performance of Two-tier State Storage

We attest to the efficiency of SharDAG's *Two-tier* state storage through simulations. Since state reconfiguration protocol is not the focus of this paper, we simulate the key operations on the underlying state storage in the state reconfiguration phase. Our evaluation involves 8 shards and 30 epochs, each containing 1M transactions, and is implemented on a computing cluster node with Intel Xeon E5-2678 CPU and 270 GB memory. The 0th epoch starts with empty state storage, and at the end of each epoch, the accounts are reassigned via R-METIS [48]. Fig. 15(a) shows the time consumed to execute all transactions in the sampled epoch. *Two-tier* presents up to 26.9% improvement over *Monolithic*. This can be attributed to SharDAG organizing active accounts, which are likely to be accessed again soon, into a lightweight in-memory *Active-Trie*.

Fig. 15(b) shows that *Two-tier*'s average account migration latency is lower than *Monolithic* in all sampled epochs. This is because *Active-Trie* accepts most account migration operations. According to Fig. 15(c), *Two-tier* migrates fewer state data across shards than *Monolithic*. This is because in *Two-tier*, the proofs of migrated active accounts are obtained from the lightweight *Active-Trie* rather than a vast MPT. Moreover, *Two-tier* exhibits increasingly significant superiority as the epoch advances, reducing state reconfiguration delay by up to 35.9% and data transmission by up to 24.5%. This is because even though the total number of accounts increases, the number of active accounts in an epoch is limited and much smaller than the total number of accounts. Hence, our *Two-tier* design performs better when the system runs for a long time.

While *Two-tier* outperforms *Monolithic* by exploiting the active account characteristics, it still migrates over 100 MB of state data across shards. The main reason is that the migrated data contains a lot of Merkle proofs. Possible enhancements include merging duplicate nodes in Merkle proofs or replacing *Active-Trie* with other authenticated data structures that offer more concise state proofs and efficient verification.

VIII. Discussion

Permissioned and permissionless setting. SharDAG can be applied in both permissioned and permissionless settings. To resist Sybil attacks in a permissionless network, SharDAG can use PoW-based identity establishment, which is widely adopted in permissionless sharding blockchains [27], [29].

Application scenario and workload. Although SharDAG design is well-suited for current realistic asset-transfer workloads characterized by the dominant silent assets, we aim to explore its adaptability to more application scenarios. If the ratio of silent assets changes, we can dynamically adjust the cross-shard account aggregation interval accordingly for optimal performance. We leave this as our future work.

Compatibility with other sharding schemes. Our cross-shard avatar account caching and aggregation design is built on the one-to-one account assignment problem. It is compatible with many one-to-one account assignment strategies, e.g., hash-based [28] and graph-based [38]. Additionally, our two-tier state storage is suitable for state reconfiguration schemes characterized by *active accounts* [38], [48], which is more efficient in reducing unnecessary migrations than other schemes.

IX. Conclusion

We present SharDAG, a novel adaptive sharding mechanism that provides storage scalability for DAG-based blockchains. SharDAG supports efficient transaction processing by harnessing characteristics (*silent assets*) discovered in real-world blockchain workloads. SharDAG adaptively caches avatar accounts for receivers when processing cross-shard transactions, thus reducing the cross-shard frequency with minimized extra storage overhead. Besides, SharDAG proposes a Byzantine resilient cross-shard verification mechanism to support secure avatar account aggregation, thus ensuring state consistency across shards. Further, SharDAG devises a two-tier state storage model to support efficient state reconfiguration by leveraging the *active accounts* feature. Evaluation results demonstrate the efficiency and scalability of SharDAG. Our released code is available at https://github.com/CGCL-codes/SharDAG.

X. Acknowledgements

This work was supported by National Key Research and Development Program of China under Grant No. 2021YFB2700700, National Natural Science Foundation of China (Grant No. 62072197), Key Research and Development Program of Hubei Province No. 2021BEA164. The research was supported in part by a RGC RIF grant under the contract R6021-20, RGC CRF grants under the contracts C7004-22G and C1029-22G, and RGC GRF grants under the contracts 16209120, 16200221 and 16207922. Jiang Xiao is the corresponding author of this work.

REFERENCES

[1] "The tangle," 2018, https://www.iota.org/.

[2] "Byteball: A decentralized system for storage and transfer of value," 2016, https://byteball.org/Byteball.pdf.

[3] "The swirlds hashgraph consensus algorithm: Fair, fast, byzantine fault tolerance," 2016, https://www.swirlds.com/downloads/SWIRLDS-TR-2016-01.pdf.

[4] C. Li, P. Li, D. Zhou, Z. Yang, M. Wu, G. Yang, W. Xu, F. Long, and A. C.-C. Yao, "A decentralized blockchain with high throughput and fast confirmation," in *Proceedings of the USENIX Annual Technical Conference (ATC'20)*, 2020, pp. 515–528.

[5] H. Yu, I. Nikolić, R. Hou, and P. Saxena, "Ohie: Blockchain scaling made simple," in *Proceedings of the IEEE Symposium on Security and Privacy (SP'20)*, 2020, pp. 90–105.

[6] J. Xu, Y. Cheng, C. Wang, and X. Jia, "Occam: A secure and adaptive scaling scheme for permissionless blockchain," in *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS'21)*, 2021, pp. 618–628.

[7] A. Spiegelman, N. Giridharan, A. Sonnino, and L. Kokoris-Kogias, "Bullshark: Dag bft protocols made practical," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'22)*, 2022, pp. 2705–2718.

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

[9] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[10] W. Yang, X. Dai, J. Xiao, and H. Jin, "Ldv: A lightweight dag-based blockchain for vehicular social networks," *IEEE Transactions on Vehicular Technology*, vol. 69, no. 6, pp. 5749–5759, 2020.

[11] J. Ni, J. Xiao, S. Zhang, B. Li, B. Li, and H. Jin, "Fluid: Towards efficient continuous transaction processing in dag-based blockchains," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 12, pp. 12 679–12 692, 2023.

[12] Q. Wang, J. Yu, S. Chen, and Y. Xiang, "Sok: Dag-based blockchain systems," *ACM Computing Surveys*, vol. 55, no. 12, pp. 1–38, 2023.

[13] 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 *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'22)*, 2022, pp. 1968–1977.

[14] C. Curino, E. P. C. Jones, Y. Zhang, and S. R. Madden, "Schism: a workload-driven approach to database replication and partitioning," *Proceedings of the VLDB Endowment*, vol. 3, no. 1, pp. 48–57, 2010.

[15] R. Taft, E. Mansour, M. Serafini, J. Duggan, A. J. Elmore, A. Aboulnaga, A. Pavlo, and M. Stonebraker, "E-store: Fine-grained elastic partitioning for distributed transaction processing systems," *Proceedings of the VLDB Endowment*, vol. 8, no. 3, pp. 245–256, 2014.

[16] G. Prasaad, A. Cheung, and D. Suciu, "Handling highly contended oltp workloads using fast dynamic partitioning," in *Proceedings of the International Conference on Management of Data (SIGMOD'20)*, 2020, pp. 527–542.

[17] M. Abebe, H. Lazu, and K. Daudjee, "Proteus: Autonomous adaptive storage for mixed workloads," in *Proceedings of the International Conference on Management of Data (SIGMOD'22)*, 2022, pp. 700–714.

[18] J. Hellings and M. Sadoghi, "Byshard: Sharding in a byzantine environment," in *Proceedings of the International Conference on Very Large Data Bases (VLDB'21)*, 2021, pp. 2230–2243.

[19] L. Luu, V. Narayanan, C. Zheng, K. Baweja, S. Gilbert, and P. Saxena, "A secure sharding protocol for open blockchains," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'16)*, 2016, pp. 17–30.

[20] 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.

[21] P. Zheng, Q. Xu, Z. Zheng, Z. Zhou, Y. Yan, and H. Zhang, "Meepo: Sharded consortium blockchain," in *Proceedings of the IEEE International Conference on Data Engineering (ICDE'21)*, 2021, pp. 1847–1852.

[22] M. J. Amiri, D. Agrawal, and A. El Abbadi, "Sharper: Sharding permissioned blockchains over network clusters," in *Proceedings of the International Conference on Management of Data (SIGMOD'21)*, 2021, pp. 76–88.

[23] Z. Hong, S. Guo, and P. Li, "Scaling blockchain via layered sharding," *IEEE Journal on Selected Areas in Communications*, vol. 40, no. 12, pp. 3575–3588, 2022.

[24] J. Hellings, D. P. Hughes, J. Primero, and M. Sadoghi, "Cerberus: Minimalistic multi-shard byzantine-resilient transaction processing," *Journal of Systems Research (JSys'23)*, vol. 3, no. 1, 2023.

[25] Z. Hong, S. Guo, E. Zhou, J. Zhang, W. Chen, J. Liang, J. Zhang, and A. Zomaya, "Prophet: Conflict-free sharding blockchain via byzantine-tolerant deterministic ordering," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'23)*, 2023, pp. 1–10.

[26] Z. Hong, S. Guo, E. Zhou, W. Chen, H. Huang, and A. Zomaya, "Gridb: Scaling blockchain database via sharding and off-chain cross-shard mechanism," in *Proceedings of the International Conference on Very Large Data Bases (VLDB'23)*, 2023, pp. 1685–1698.

[27] M. Zamani, M. Movahedi, and M. Raykova, "Rapidchain: Scaling blockchain via full sharding," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'18)*, 2018, pp. 931–948.

[28] J. Wang and H. Wang, "Monoxide: Scale out blockchains with asynchronous consensus zones," in *Proceedings of the USENIX Symposium on Networked Systems Design and Implementation (NSDI'19)*, 2019, pp. 95–112.

[29] E. Kokoris-Kogias, P. Jovanovic, L. Gasser, N. Gailly, E. Syta, and B. Ford, "Omniledger: A secure, scale-out, decentralized ledger via sharding," in *Proceedings of the IEEE Symposium on Security and Privacy (SP'18)*, 2018, pp. 583–598.

[30] 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 International Conference on Management of Data (SIGMOD'19)*, 2019, pp. 123–140.

[31] M. Herlihy, B. Liskov, and L. Shrira, "Cross-chain deals and adversarial commerce," in *Proceedings of the International Conference on Very Large Data Bases (VLDB'19)*, 2019, pp. 100–113.

[32] V. Zakhary, D. Agrawal, and A. E. Abbadi, "Atomic commitment across blockchains," in *Proceedings of the International Conference on Very Large Data Bases (VLDB'19)*, 2019, pp. 1319–1331.

[33] Z. Hong, S. Guo, P. Li, and W. Chen, "Pyramid: A layered sharding blockchain system," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'21)*, 2021, pp. 1–10.

[34] Y. Liu, X. Xing, H. Cheng, D. Li, Z. Guan, J. Liu, and Q. Wu, "A flexible sharding blockchain protocol based on cross-shard byzantine fault tolerance," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 2276–2291, 2023.

[35] M. Li, Y. Lin, J. Zhang, and W. Wang, "Cochain: High concurrency blockchain sharding via consensus on consensus," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'23)*, 2023, pp. 1–10.

[36] S. Gupta, S. Rahnama, J. Hellings, and M. Sadoghi, "Resilientdb: Global scale resilient blockchain fabric," in *Proceedings of the International Conference on Very Large Data Bases (VLDB'20)*, 2020, pp. 868–883.

[37] M. Zhang, J. Li, Z. Chen, H. Chen, and X. Deng, "Cycledger: A scalable and secure parallel protocol for distributed ledger via sharding," in *Proceedings of the IEEE International Parallel and Distributed Processing Symposium (IPDPS'20)*, 2020, pp. 358–367.

[38] Y. Zhang, S. Pan, and J. Yu, "Txallo: Dynamic transaction allocation in sharded blockchain systems," in *Proceedings of the IEEE International Conference on Data Engineering (ICDE'23)*, 2023, pp. 721–733.

[39] M. Li, W. Wang, and J. Zhang, "Lb-chain: Load-balanced and low-latency blockchain sharding via account migration," *IEEE Transactions on Parallel and Distributed Systems*, vol. 34, no. 10, pp. 2797–2810, 2023.

[40] X. Qi, "S-store: A scalable data store towards permissioned blockchain sharding," in *Proceedings of the IEEE Conference on Computer Communications (INFOCOM'22)*, 2022, pp. 1978–1987.

[41] C. Li, H. Huang, Y. Zhao, X. Peng, R. Yang, Z. Zheng, and S. Guo, "Achieving scalability and load balance across blockchain shards for state sharding," in *Proceedings of the International Symposium on Reliable Distributed Systems (SRDS'22)*, 2022, pp. 284–294.

[42] V. Hou Su, S. Sen Gupta, and A. Khan, "Automating etl and mining of ethereum blockchain network," in *Proceedings of the ACM International Conference on Web Search and Data Mining (WSDM'22)*, 2022, pp. 1581–1584.

[43] https://etherscan.io/dashboards/contract-statistics.

[44] V. Bagaria, S. Kannan, D. Tse, G. Fanti, and P. Viswanath, "Prism: Deconstructing the blockchain to approach physical limits," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'19)*, 2019, pp. 585–602.
[45] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and tusk: a dag-based mempool and efficient bft consensus," in *Proceedings of the European Conference on Computer Systems (EuroSys'22)*, 2022, pp. 34–50.
[46] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman, "All you need is dag," in *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC'21)*, 2021, pp. 165–175.
[47] J. Xiao, S. Zhang, Z. Zhang, B. Li, X. Dai, and H. Jin, "Nezha: Exploiting concurrency for transaction processing in dag-based blockchains," in *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS'22)*, 2022, pp. 269–279.
[48] E. Fynn and F. Pedone, "Challenges and pitfalls of partitioning blockchains," in *Proceedings of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks Workshops (DSN-W'18)*, 2018, pp. 128–133.
[49] C. Chen, Q. Ma, X. Chen, and J. Huang, "User distributions in shard-based blockchain network: Queueing modeling, game analysis, and protocol design," in *Proceedings of the International Symposium on Theory, Algorithmic Foundations, and Protocol Design for Mobile Networks and Mobile Computing (MobiHoc'21)*, 2021, pp. 221–230.
[50] M. Król, O. Ascigil, S. Rene, A. Sonnino, M. Al-Bassam, and E. Rivière, "Shard scheduler: object placement and migration in sharded account-based blockchains," in *Proceedings of the ACM Conference on Advances in Financial Technologies (AFT'21)*, 2021, pp. 43–56.
[51] Y. Tao, B. Li, and B. Li, "On sharding across heterogeneous blockchains," in *Proceedings of the IEEE International Conference on Data Engineering (ICDE'23)*, 2023, pp. 477–489.
[52] S. Rahnama, S. Gupta, R. Sogani, D. Krishnan, and M. Sadoghi, "Ringbft: Resilient consensus over sharded ring topology," in *Proceedings of the International Conference on Extending Database Technology (EDBT'22)*, 2022, pp. 2:298–2:311.
[53] M. J. Amiri, Z. Lai, L. Patel, B. T. Loo, E. Lo, and W. Zhou, "Saguaro: An edge computing-enabled hierarchical permissioned blockchain," in *Proceedings of the IEEE International Conference on Data Engineering (ICDE'23)*, 2023, pp. 259–272.
[54] Y. Tao, B. Li, J. Jiang, H. C. Ng, C. Wang, and B. Li, "On sharding open blockchains with smart contracts," in *Proceedings of the IEEE International Conference on Data Engineering (ICDE'20)*, 2020, pp. 1357–1368.
[55] M. Li, Y. Lin, J. Zhang, and W. Wang, "Jenga: Orchestrating smart contracts in sharding-based blockchain for efficient processing," in *Proceedings of the IEEE International Conference on Distributed Computing Systems (ICDCS'22)*, 2022, pp. 133–143.
[56] E. Fynn, A. Bessani, and F. Pedone, "Smart contracts on the move," in *Proceedings of the Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'20)*, 2020, pp. 233–244.
[57] H. Huang, Y. Zhao, and Z. Zheng, "tmpt: Reconfiguration across blockchain shards via trimmed merkle patricia trie," in *Proceedings of the IEEE/ACM International Symposium on Quality of Service (IWQoS'23)*, 2023, pp. 1–10.
[58] https://ethereum.org/en/developers/docs/gas/.
[59] Y. Lu, Z. Lu, and Q. Tang, "Bolt-dumbo transformer: Asynchronous consensus as fast as the pipelined bft," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'22)*, 2022, pp. 2159–2173.
[60] J.-Y. Kim, J. Lee, Y. Koo, S. Park, and S.-M. Moon, "Ethanos: efficient bootstrapping for full nodes on account-based blockchain," in *Proceedings of the European Conference on Computer Systems (EuroSys'21)*, 2021, pp. 99–113.