PolyShard: Coded Sharding Achieves Linearly Scaling Efficiency and Security Simultaneously

Songze Li†, Mingchao Yu†, Chien-Sheng Yang†, A. Salman Avestimehr‡, Sreeram Kannan* and Pramod Viswanath‡
† University of Southern California  * University of Washington  ‡ University of Illinois at Urbana-Champaign

Abstract—Today’s blockchain designs suffer from a trilemma claiming that no blockchain system can simultaneously achieve decentralization, security, and performance scalability. For current blockchain systems, as more nodes join the network, the efficiency of the system (computation, communication, and storage) stays constant at best. A leading idea for enabling blockchains to scale efficiency is the notion of sharding: different subsets of nodes handle different portions of the blockchain, thereby reducing the load for each individual node. However, existing sharding proposals achieve efficiency scaling by compromising on trust - corrupting the nodes in a given shard will lead to the permanent loss of the corresponding portion of data. In this paper, we settle the trilemma by demonstrating a new protocol for coded storage and computation in blockchains. In particular, we propose PolyShard: “polynomially coded sharding” scheme that achieves information-theoretic upper bounds on the efficiency of the storage, system throughput, as well as on trust, thus enabling a truly scalable system.

I. INTRODUCTION

While Blockchain systems promise a host of new and exciting applications, such as digital cryptocurrency [1], industrial IoT [2], and healthcare management [3], their scalability remains a critical challenge [4]. In fact, a well-known blockchain trilemma has been raised [5] claiming that no decentralized ledger system can simultaneously achieve 1) security (against adversarial attacks), 2) decentralization (of computation and storage resources), and 3) scalability (of throughput with the network size). All existing major blockchains either achieve decentralization at the cost of efficiency, or efficiency at the cost of decentralization and/or security.

The focus of this paper is to formalize and study a version of the blockchain trilemma, in order to understand whether it is fundamental to blockchain systems. Corresponding to the three traits of the trilemma, we study the following performance measures of blockchain systems: security - measured as the number of malicious nodes the system can tolerate, decentralization - measured as the fraction of the block chain (or ledger) that is stored and processed by each node (we denote its inverse by storage efficiency), and scalability - measured as the total throughput of the system that is the number of computations performed in the system (e.g., number of transactions verified) in a unit period of time.

Within this context, let’s examine the current blockchain systems. Bitcoin [1] and Ethereum [6] are designed based on a full replication system, where each network node stores the entire blockchain and replicates all the computations (e.g., transaction verifications), as demonstrated in Figure 1(a). This feature enables high security (of tolerating 49% adversarial nodes), however drastically limits the storage efficiency and throughput of the system: they stay constant regardless of the number of nodes N. For example, Bitcoin restricts its block size to 1 MB, and processing rate to 7 transactions/sec [7]. In practice, the computational burden even increases with N (e.g., mining puzzles get harder as time progresses and more users join), causing the throughput to drop.

To scale out throughput and storage efficiency, the leading solution being discussed in the blockchain literature is via sharding [8]–[10]. The key idea is to partition the blockchain into K independent sub-chains, which are then replicated separately at q = N/K nodes to yield K smaller full-replication systems, a.k.a., shards (Figure 1(b)). This way, both storage efficiency and throughput are improved by a factor of K. However, to scale this improvement with network size, K must increase linearly with N. Consequently, the number of nodes q per shard has to be held constant, which allows an attacker to corrupt as few as q/2 nodes to compromise a shard. This yields a security level of q/2, which approaches zero as N grows. Although various efforts have been made to alleviate this security issue (e.g., by periodically shuffling the nodes [9]), they are susceptible to powerful adversaries (e.g., who can corrupt the nodes after the shuffling), yet none scale security.

In summary, both full replication and sharding based blockchain systems make trade-offs between the scalability of throughput, storage efficiency, and security. However, such a trade-off is far from optimal from information theoretical point of view. Given the N× computation and N× storage resources across all the N network nodes, the following information-theoretic upper bounds hold: security ≤ Θ(N); throughput ≤ Θ(N); storage efficiency ≤ Θ(N). It is intuitive that these bounds can be simultaneously achieved by a centralized system, allowing all the three metrics to scale. However, as pointed out by the trilemma, this has not been achieved by any existing decentralized system. This raises the following fundamental open problem:
Fig. 1: A blockchain system with 30 nodes, each of which is capable of verifying 3 transactions per epoch. (a) Full replication: It can tolerate \( \frac{N}{2} - 1 = 14 \) malicious nodes. (b) Sharding with 3 shards: With 3-sharding, the ledger is partitioned into 3 sub-ledgers, and transactions are limited to between the accounts within the same sub-ledger. It is capable of verifying 9 transactions per epoch and can tolerate \( \frac{3N}{2} - 1 = 4 \) malicious nodes. Sharding improves the storage and throughput efficiency by 3 time, at the cost of compromising security by about 3 times.

Fig. 2: The proposed PolyShard system with 30 nodes. Each node computes a verification function on a coded sub-ledger and a coded block, which are created by a distinct linear combination of the original sub-ledgers, and the proposed blocks respectively. Since encoding does not change the size of the sub-ledger, PolyShard achieves the same storage and throughput efficiency (i.e., 9 transactions per epoch) as the conventional sharding solution. Additionally, PolyShard improves the security guarantee to protect against \( (N - K)/2 = 13 \) malicious nodes, for degree-1 verification functions.

Is there a blockchain design that simultaneously scales storage efficiency, security, and throughput?

We answer this question affirmatively by introducing the concept of *coded sharding*. In particular, we propose PolyShard (polynomially coded sharding), a scheme that simultaneously achieves linear scaling in throughput, storage efficiency, and security (i.e., \( \Theta(N) \)). We show that PolyShard achieves all three information-theoretic upper bounds and enables a truly scalable blockchain system. (see Table I)

| Verification Item                  | Storage Efficiency | Security Efficiency | Throughput Efficiency |
|------------------------------------|--------------------|---------------------|----------------------|
| Full replication                   | \( \Theta(1) \)    | \( \Theta(N) \)     | \( \Theta(1) \)     |
| Sharding                           | \( \Theta(N) \)    | \( \Theta(1) \)     | \( \Theta(N) \)     |
| Information-theoretic limit        | \( \Theta(N) \)    | \( \Theta(N) \)     | \( \Theta(N) \)     |
| PolyShard (this paper)             | \( \Theta(N) \)    | \( \Theta(N) \)     | \( \Theta(N) \)     |

TABLE I: Performance comparison of the proposed PolyShard verification scheme with other benchmarks and the information-theoretic limits.

PolyShard is inspired by recent developments in *coded computing* [11]–[19], in particular Lagrange Coded Computing [19], which provides a transformative framework for injecting computation redundancy in unorthodox coded forms in order to deal with failures and errors in distributed computing. The key idea behind PolyShard is that instead of storing and processing a single uncoded shard as done conventionally, each node stores and computes on a coded shard of the same size that is generated by linearly mixing uncoded shards (Figure 2), using the well-known Lagrange polynomial. This coding provides computation redundancy to simultaneously provide security against erroneous results from malicious nodes, which is enabled by noisy polynomial interpolation techniques (e.g., Reed-Solomon decoding).

While coding is generally applicable in many distributed computing scenarios, the following two salient features make PolyShard particularly suitable for blockchain systems.

- **Oblivious**: The coding strategy applied to generate coded shards is oblivious of the verification function. That means, the same coded data can be simultaneously used for multiple verification items (examples: digital signature verification and balance checking in a payment system);
- **Incremental**: PolyShard allows each node to grow its local coded shard by coding over the newest verified blocks, without needing to access the previous ones. This helps to maintain a constant coding overhead as the chain grows.

**Other related works.** Prominent sharding proposals in the literature are [8]–[10], [20]–[31]. As an example, ELASTICO [8] partitions the incoming transactions into shards, and each shard is verified by a disjoint committee of nodes in parallel. Omnilayer [9] improved upon ELASTICO in multiple avenues, including new methods to assign nodes into shards with a higher security guarantee, an atomic protocol for cross-shard transactions, and further optimization on the communication and storage designs. Finally, there is also a growing interest for leveraging coding to deal with the issues of data availability, computation verification, and storage overhead in decentralized blockchain systems (e.g., [32]–[36]).

II. PROBLEM FORMULATION: BLOCK VERIFICATION

A blockchain system manages a decentralized ledger of its clients’ transactions, over a large number of untrusted network nodes. The clients submit their transactions to the network nodes, who group the transactions into blocks that to be included in the system. The accepted blocks are organized into a chain where each block contains a hash pointer to its predecessor. The chain structure provides high security guarantee since for an adversary to tamper the contents of any block, it has to re-grow the entire chain afterwards, which is extremely expensive in computation power. Here we consider a *sharded* blockchain system whose grand ledger is partitioned into \( K \) independent shards, each of which maintains a disjoint sub-chain that records the transactions between the accounts associated with the same shard. At a high level, at each time epoch, every shard proposes one block of transactions, and verifies it over the current state of its sub-chain. Once the block passes the verification, it will be appended to the corresponding sub-chain. We now define the system formally.
A. Computation model

Each shard $k$ ($k \in [1, K]$) maintains its own sub-chain. We denote the state of the $k$-th sub-chain before epoch $t$ by $Y_k^{t-1} = (Y_k(1), \ldots, Y_k(t-1))$, where $Y_k(t) \in U$ denotes the block accepted to shard $k$ at epoch $t$, and $U$ is a vector space over a field $\mathbb{F}$. When a new block $X_k(t) \in U$ is proposed for shard $k$ in epoch $t$ (using some consensus algorithm like proof-of-work (PoW)), the blockchain system needs to verify its legitimacy (e.g., sender accounts have sufficient fund, no double-spending) over the current state $Y_k^{t-1}$. We abstract out the mechanism which generates the proposals and focus on verifying the proposed blocks.

We denote the verification function by $f^t : U^t \rightarrow \mathbb{V}$, over $X_k(t)$ and the sub-chain $Y_k^{t-1}$, for some vector space $\mathbb{V}$ over $\mathbb{F}$. For instance, for a cryptocurrency blockchain that keeps track of balances, $f^t$ could be a function that calculates the difference between the incoming and outgoing transactions for account $X_k(t)$ in epoch $t$.

**Balance checking**: to check each transaction has more input values than those spent in its outputs; and also check that the transactions in a block contain sufficient funds to pay the transaction fees/mining rewards.

**Signature checking**: to verify that a payment transaction spending some funds in an account is indeed submitted by the owner of that account. This often involves computing cryptographic hashes using the account's public key, and verifying the results with the digital signature.

Having obtained $h_k^t = f^t(X_k(t), Y_k^{t-1})$, shard $k$ computes an indicator variable $e_k^t \triangleq 1(h_k^t \in W)$, where $W \subseteq \mathbb{V}$ denotes the set of function outputs that affirm $X_k(t)$. Finally, the verified block $Y_k(t)$ is computed as $Y_k(t) = e_k^t X_k(t)$, and added to the sub-chain of shard $k$. We note that if the block is invalid, an all-zero block will be added at epoch $t$.

B. Networking model

The above blockchain system is implemented distributedly over $N$ untrusted nodes. We consider a homogeneous and synchronous network, i.e., all nodes have similar process power, and the delay of communication between any pair of nodes is bounded by some known constant. A subset $\mathcal{M} \subset \{1, \ldots, N\}$ of the nodes may be corrupted, and are subject to Byzantine faults, i.e., they may compute and communicate arbitrary erroneous results during block verification. We aim to design secure verification schemes against the following strong adversary model:

- The adversaries can corrupt a fixed fraction of the network node, i.e., the number of malicious nodes grows linearly with $N$.
- If a conventional sharding solution were employed, the adversaries know the allocation of nodes to the shards, and are able to adaptively select the subset $\mathcal{M}$ of nodes to attack.

We note that under this adversary model, the random shard rotation approach [8], [9] is no longer secure since the adversaries can focus their power to attack a single shard after knowing which nodes are assigned into this shard. Next, we present the networking protocol that will be followed by the honest nodes. Note that adversarial nodes are not required to follow this protocol.

**Storage.** At epoch $t$, each node $i$, $i = 1, \ldots, N$, locally stores some data, denoted by $Z_i^{t-1} = (Z_i(1), \ldots, Z_i(t-1))$, where $Z_i(j) \in \mathbb{W}$ for some vector space $\mathbb{W}$ over $\mathbb{F}$. The locally stored data $Z_i^{t-1}$ is generated from all shards of the blockchain using some function $\phi_i^{t-1}$, i.e., $Z_i^{t-1} = \phi_i^{t-1}(Y_1^{t-1}, \ldots, Y_K^{t-1})$.

**Verification.** Given the $K$ proposed blocks $\{X_k(t)\}_{k=1}^K$, one for each shard, the goal of block verification is to compute $\{f^t(X_k(t), Y_k^{t-1})\}_{k=1}^K$ distributedly over untrusted nodes. We implement this verification process in two steps. In the first step, each node $i$ computes an intermediate result $g_i^t$ using some function $\rho_i^t$ on the proposed blocks and its local storage, such that $g_i^t = \rho_i^t(X_i(t), \ldots, X_K(t), Z_i^{t-1})$, and then broadcasts the result $g_i^t$ to all other nodes.

The nodes exploit the received computation results to reduce the final verification results in the second step. Specifically, each node $i$ decodes the verification results for all $K$ shards $\hat{h}_i^1, \ldots, \hat{h}_i^K$, using some function $\psi_i^t$, i.e., $(\hat{h}_i^1, \ldots, \hat{h}_i^K) = \psi_i^t(g_i^1, \ldots, g_N^N)$. Using these decoded results, node $i$ computes the indicator variables $\hat{e}_i^t = 1(\hat{h}_i^t \in W)$, and then the verified blocks $\hat{Y}_i(t) = \hat{e}_i^t X_i(t)$, for all $k = 1, \ldots, K$. Finally, each node $i$ utilizes the verified blocks to update its local storage using some function $\chi_i^t$, i.e., $Z_i^t = \chi_i^t(\hat{Y}_i^1(t), \ldots, \hat{Y}_i^K(t), Z_i^{t-1}) = \phi_i^t(\hat{Y}_1^t, \ldots, \hat{Y}_K^t)$. Here $\hat{Y}_k^t$ is the sequence of blocks in shard $k$ verified at node $i$ up to time $t$. To update the local storage $Z_i^t$, while node $i$ can always apply $\phi_i^t$ on the uncoded shards $\hat{Y}_1^t, \ldots, \hat{Y}_K^t$, it is highly desirable for $\chi_i^t$ to be incremental: we only need to create a coded block from $\hat{Y}_1^t, \ldots, \hat{Y}_K^t$, and append it to $Z_i^{t-1}$. This helps to significantly reduce the computational and storage complexities.

C. Performance metrics

We denote a block verification scheme by $S$, defined as a sequence of collections of the functions, i.e., $S = (\{\phi_i^t, \psi_i^t, \chi_i^t\}_{t=1}^\infty)$. We are interested in the following three performance metrics of $S$.

**Storage efficiency.** Denoted by $\gamma_S$, it is defined as the ratio between the size of the entire block chain and the size of the data stored at each node, i.e.,

$$\gamma_S \triangleq \frac{K \log |U|}{\log |\mathbb{W}|}.$$ (1)

The above definition also applies to a probabilistic formulation where the blockchain elements $Y_k(j)$’s and the storage elements $Z_i(j)$’s are modelled as i.i.d. random variables with uniform distribution in their respective fields, where the storage efficiency is defined using the entropy of the random variables.

**Security.** We say $S$ is $b$-secure if the honest nodes can recover all the correct verification results under the presence of up to $b$ malicious nodes. More precisely, for any subset $\mathcal{M} \subset \{1, \ldots, N\}$ of malicious nodes with $|\mathcal{M}| \leq b$, and each node $i \notin \mathcal{M}$, a $b$-secure scheme will guarantee that $(\hat{h}_1^1, \ldots, \hat{h}_K^1) = (h_1^1, \ldots, h_K^1)$, for all $t = 1, 2, \ldots$. We define
the security of \( S \), denoted by \( \beta_S \), as the maximum \( b \) it could achieve:
\[
\beta_S \triangleq \sup\{b : S \text{ is } b\text{-secure}\}. \tag{2}
\]

Throughput. We measure throughput of the system by taking into account the number of blocks verified per epoch and the associated computational cost. We denote by \( c(f) \) the computational complexity of a function \( f \), which is the number of additions and multiplications performed to evaluate \( f \).

We define the throughput of \( S \), denoted by \( \lambda_S \), as the average number of blocks that are correctly verified per unit discrete round, which includes all the computations performed at all \( N \) nodes to verify the incoming \( K \) blocks. That is,
\[
\lambda_S \triangleq \liminf_{t \rightarrow \infty} \frac{1}{K} \sum_{i=1}^{N} \frac{(c(\mu_i^t) + c(\psi_i^t) + c(\chi_i)) / (Nc(f^t))}{}. \tag{3}
\]

The above three metrics correspond to the three traits in the blockchain trilemma, and all current blockchain systems have to trade off one for another. The goal of this paper is to understand the information-theoretic limits on these metrics, and design verification schemes that can simultaneously achieve the limits, hence settling the trilemma.

III. BASELINE PERFORMANCE

We now present the information-theoretic upper bounds on the three performance metrics for any blockchain, and then study the performance of two state-of-the-art blockchain schemes.

Information-theoretic upper bounds. In terms of security, the maximum number of adversaries any verification scheme can tolerate cannot exceed half of the number of network nodes \( N \). Thus, the security \( \beta \leq N/2 \). In terms of storage, for the verification to be successful, the size of the chain should not exceed the aggregated storage resources of the \( N \) nodes. Otherwise, the chain cannot be fully stored. We thus have \( \gamma \leq N \). Finally, to verify the \( K \) incoming blocks, the verification function \( f^t \) must be executed at least \( K \) times in total. Then, we have \( \lambda \leq \frac{K}{Nc(f^t)} = N \). Thus, the information-theoretic upper bounds of security, storage efficiency, and throughput all scale linearly with the network size \( N \).

Full replication. In terms of storage efficiency, since each node stores all the \( K \) shards of the entire blockchain, full replication scheme yields \( \gamma_{\text{full}} = 1 \). Since every node verifies all the \( K \) blocks, the throughput of the full replication scheme is \( \lambda_{\text{full}} = \frac{K}{Nc(f^t)(Nc(f^t))} = 1 \). Thus the full replication scheme does not scale with the network size, as both the storage and the throughput remain constant as \( N \) increases. The advantage is that the simple majority-rule will allow the correct verification and update of every block as long as there are less than \( N/2 \) malicious nodes. Thus, \( \beta_{\text{full}} = N/2 \).

Uncoded sharding scheme. In conventional sharding, the blockchain consists of \( K \) disjoint sub-chains known as shards. The \( N \) nodes are partitioned into \( K \) groups of equal size \( q = N/K \), and each group of nodes manages a single shard; this is a full replication system with \( K' = 1 \) shard and \( N' = q \) nodes. Since each node stores and verifies a single shard, the storage efficiency and throughput become \( \gamma_{\text{sharding}} = K \) and \( \lambda_{\text{sharding}} = \frac{K}{Nc(f^t)(Nc(f^t))} = K \), respectively. For these two metrics to scale linearly with \( N \), it must be true that \( K = \Theta(N) \). Consequently, the group size \( q \) becomes a constant. Hence, compromising as few as \( q/2 \) nodes will corrupt one shard and the chain. Thus, this scheme only has a constant security of \( \beta_{\text{sharding}} = q/2 = O(1) \). Although system solutions such as shard rotations can help achieve linearly scaling security guarantees, they are only secure when the adversary is non-adaptive (or very slowly adaptive). When the adversary is dynamic, it can corrupt all nodes belonging to a particular shard instantaneously after the shard assignment has been made. Under this model, the security reduces to a constant.

Neither full replication nor the above sharding scheme exempts from the blockchain trilemma, and has to make tradeoff in scaling towards the information-theoretic limits.

IV. PolySHARD FOR BALANCE CHECKING

In this section, we present Polyshard and show how it works on a simple cryptocurrency blockchain system that records balance transfers between accounts. We assume that there are \( M \) accounts (or addresses) associated with each shard, for some constant \( M \) that does not scale with \( t \). For the purpose of balance checking, we can compactly represent the block of transactions submitted to shard \( k \) at epoch \( t \), \( X_k(t) \), as a pair of real vectors \( X_k(t) = (X_k^{\text{send}}(t), X_k^{\text{receive}}(t)) \in \mathbb{R}^M \times \mathbb{R}^M \). Given a transaction in the block that reads “Account \( p \) sends \( s \) units to account \( q \), we will have \( X_k^{\text{send}}(t)[p] = -s \), and \( X_k^{\text{receive}}(t)[q] = s \). Accounts that do no send/receive funds will have their entries in the send/receive vectors set to zeros. To verify \( X_k(t) \), we need to check that all the sender accounts in \( X_k(t) \) have accumulated enough unspent funds from previous transactions. This naturally leads to the following verification function.
\[
f^t(X_k(t), Y_k^{t-1}) = X_k^{\text{send}}(t) + \sum_{i=1}^{t-1} (Y_k^{\text{send}}(i) + Y_k^{\text{receive}}(i)).
\]

This function is linear in its input, and has computational complexity \( c(f^t) = O(t) \). We claim the block \( X_k(t) \) valid (i.e., \( e_k = 1 \)) if no entry in the function’s output vector is negative, and invalid (i.e., \( e_k = 0 \)) otherwise. After computation, we have the verified block \( Y_k(t) = (Y_k^{\text{send}}(t), Y_k^{\text{receive}}(t)) = e_k X_k(t) \). Note that this balance checking can be alternatively implemented by having each shard simply store a dimension-\( M \) vector that records the aggregated balances of all associated accounts, and the storage and the verification complexity will stay constant as time progresses. However, important transaction statistics including how many transactions occur in each block, and the input/output values in each transaction, and how these statistics evolve over time will be lost in this simplified implementation. Moreover, storing a single block in each shard without the protection of a long chain makes the shard more vulnerable to malicious tampering, compromising the security guarantee. Thus, we stick to the chain structure where all past transactions are kept in the ledger.

We consider operating this sharded payment blockchain over a network of \( N \) nodes, with a maximum \( \mu \) fraction of which are corrupted by quickly-adaptive adversaries. For this
system, we propose a coded transaction verification scheme, named PolyShard, which simultaneously scales the storage efficiency, security, and throughput with the network size.

A. Coded sub-chain

In PolyShard, at epoch $t$, each node $i$ stores a coded sub-chain $\tilde{Y}_i(t) = (\tilde{Y}_i(1), \ldots, \tilde{Y}_i(t))$ where each component $\tilde{Y}_i(m)$ is a coded block generated by the verified blocks $Y_i(m), \ldots, Y_k(m)$ from all $K$ shards in epoch $m$. The coding is through evaluation of the well-known Lagrange interpolation polynomial. Specifically, we pick $K$ arbitrarily distinct real numbers $\omega_1, \ldots, \omega_K \in \mathbb{R}$, each corresponding to a shard. Then for each $m = 1, \ldots, t$, we create a Lagrange polynomial in variable $z$ as follows.

$$ u_m(z) = \sum_{k=1}^{K} Y_k(m) \prod_{j \neq k} \frac{z - \omega_j}{\omega_k - \omega_j}. \quad (4) $$

We note that this polynomial is designed such that $u_m(\omega_k) = Y_k(m)$ for all $k = 1, \ldots, K$.

Next, we pick $N$ arbitrarily distinct numbers $\alpha_1, \ldots, \alpha_N \in \mathbb{R}$, one for each node. For each $i = 1, \ldots, N$, we create a coded block $\tilde{Y}_i(m)$ that is stored at node $i$, by evaluating the above $u_m(z)$ at the point $\alpha_i$, i.e.,

$$ \tilde{Y}_i(m) = u_m(\alpha_i) = \sum_{k=1}^{K} Y_k(m) \prod_{j \neq k} \frac{\alpha_i - \omega_j}{\omega_k - \omega_j} = \sum_{k=1}^{K} \ell_{ik} Y_k(m). \quad (5) $$

We note that $\tilde{Y}_i(m)$ is a linear combination of the uncoded blocks $Y_1(m), \ldots, Y_k(m)$, and the coefficients $\ell_{ik} = \prod_{j \neq k} \frac{\alpha_i - \omega_j}{\omega_k - \omega_j}$ do not depend on the time index $m$. Therefore, one can think of each node $i$ as having a fixed vector $\ell_{ik}$ by which it mixes the different shards to create $\tilde{Y}_i(t)$, and stores it locally. The size of each coded sub-chain is $K$ times smaller than the size of the entire blockchain, and the storage efficiency of PolyShard is $\gamma_{\text{PolyShard}} = K$.

B. Coded verification

At epoch $t$, each shard $k$ proposes and broadcasts a new block $X_k(t)$ to be added to the sub-chain after balance checking. PolyShard verifies these blocks in three steps.

Step 1: block encoding. Upon receiving the proposed blocks, each node $i$ computes a coded block $\tilde{X}_i(t)$ as a linear combination using the same set of coefficients in (5). That is, $\tilde{X}_i(t) = (\tilde{X}_i^{\text{send}}(t), \tilde{X}_i^{\text{receive}}(t)) = \sum_{k=1}^{K} \ell_{ik} X_k(t)$. We note that this encoding operation can also be viewed as evaluating the polynomial $v_i(z) = \sum_{k=1}^{K} X_k(t) \prod_{j \neq k} \frac{z - \omega_j}{\omega_k - \omega_j}$ at the point $\alpha_i$. This step incurs $O(NK)$ operations across the network, since each node computes a linear combination of $K$ blocks.

Step 2: coded computation. Each node $i$ applies the verification function $f^i$ directly to the coded block $\tilde{X}_i(t)$, and its locally stored $\tilde{Y}_i^{t-1}$ to compute an intermediate vector

$$ g^i_t = \tilde{X}_i^{\text{send}}(t) + \sum_{m=1}^{t-1} (\tilde{Y}_i^{\text{send}}(m) + \tilde{Y}_i^{\text{receive}}(m)). \quad (6) $$

Each node $i$ carries out $\Theta(t)$ operations to compute $g^i_t$, and broadcasts it to all other nodes.

Step 3: decoding. It is easy to see that $f^i(v_i(z), u_i(z), \ldots, u_{i-1}(z))$ is a univariate polynomial of degree $K - 1$, and $g^i_t$ can be viewed as evaluating this polynomial at $\alpha_i$. Given the evaluations at $N$ distinct points $\alpha_1, \ldots, \alpha_N$, each node can recover the coefficients of $f^i(v_i(z), u_i(z), \ldots, u_{i-1}(z))$ following the process of decoding a Reed-Solomon code with dimension $K$ and length $N$ (see, e.g., [37]). In order for this decoding to be robust to $\mu N$ erroneous results (i.e., achieving security $\beta_{\text{PolyShard}} = \mu N$), we must have $2\mu N \leq N - K$. In other words, a node can successfully decode $f^i(v_i(z), u_i(z), \ldots, u_{i-1}(z))$ only if the number of shards $K$ is upper bounded by $K \leq (1 - 2\mu)N$. Based on this constraint, we set the number of shards of the PolyShard scheme, $K_{\text{PolyShard}} = \lceil (1 - 2\mu)N \rceil$, which scales linearly with network size $N$. The complexity of decoding a length-$N$ Reed-Solomon code at each node is $O(N^2 log^2 N log log N)$, and the total complexity of the decoding step is $O(N^2 log^2 N log log N)$.

Having decoded $f^i(v_i(z), u_i(z), \ldots, u_{i-1}(z))$, each node evaluates it at $\omega_1, \ldots, \omega_K$ to recover $\{f^i(X_k(t), Y_{k-1}^{t-1})\}_{k=1}^{K}$, to obtain the verification results $\{\ell_{ik}X_k(t)\}_{k=1}^{K}$, and the verified blocks $\{Y_k(t) = \ell_{ik}X_k(t)\}_{k=1}^{K}$. Finally, each node $i$ computes $\tilde{Y}_i(t)$ following (5), and appends it to its local coded sub-chain. Updating the sub-chains has the same computational complexity with the block encoding step, which is $O(NK)$.

C. Performance of PolyShard

So far, we have shown that PolyShard achieves a storage efficiency $\gamma_{\text{PolyShard}} = K_{\text{PolyShard}} = \Theta(N)$, and it is also robust against $\mu N = \Theta(N)$ quickly-adaptive adversaries. The total number of operations during the verification and the sub-chain update processes is $O(NK) + N\Theta(t) + O(N^2 log^2 N log log N)$, where the term $O(NK) + O(N^2 log^2 N log log N)$ is the additional coding overhead compared with the uncoded sharding scheme. Since $K_{\text{PolyShard}} \leq N$, the coding overhead reduces to $O(N^2 log^2 N log log N)$. The throughput of PolyShard for balance checking is computed as

$$ \lambda_{\text{PolyShard}} = \lim_{t \to \infty} \inf \frac{K_{\text{PolyShard}}N\Theta(t)}{N\Theta(t) + O(N^2 log^2 N log log N)} \quad (7) $$

$$ = \lim_{t \to \infty} \inf \frac{K_{\text{PolyShard}}N\Theta(t)}{1 + \Theta(N^2 log^2 N log log N)} = \Theta(N). \quad (8) $$

We can see that since the complexities of the encoding and decoding operations of PolyShard do not scale with $t$, the coding overhead becomes irrelevant as the chain grows. The PolyShard scheme simultaneously achieves information-theoretically optimal scaling on security, storage efficiency, and throughput. We note that when the verification function is linear with the block data, codes designed for distributed storage (see, e.g., [38], [39]) can be used to achieve similar scaling as PolyShard. However, PolyShard is designed for a much more general class of verification functions including arbitrary multivariate polynomials, which cannot be handled by state-of-the-art storage codes.
REFERENCES
[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.
[2] A. Bahga and V. K. Madisetti, “Blockchain platform for industrial internet of things,” Journal of Software Engineering and Applications, vol. 9, no. 10, p. 533, 2016.
[3] M. Mettler, “Blockchain technology in healthcare: The revolution starts here,” in IEEE 18th International Conference on e-Health Networking, Applications and Services (Healthcom), pp. 1–3, IEEE, 2016.
[4] K. Croman, C. Decker, I. Eyal, A. E. Gencer, A. Juels, A. Kosba, A. Miller, P. Saxena, E. Shi, E. G. Sirer, et al., “On scaling decentralized blockchains,” in International Conference on Financial Cryptography and Data Security, pp. 106–125, Springer, 2016.
[5] T. Ometorowa, “Solving the blockchain trilemma: Decentralization, security & scalability,” https://www.coinbureau.com/analysis/solving-blockchain-trilemma/, 2018. Accessed: 2018-12-21.
[6] G. Wood, “Ethereum: A secure decentralised generalised transaction ledger,” Ethereum project yellow paper, vol. 151, pp. 1–32, 2014.
[7] A. Gervais, G. O. Karame, K. Wüst, V. Glykantzis, H. Ritzdorf, and S. Capkun, “On the security and performance of proof of work blockchains,” in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 3–16, ACM, 2016.
[8] 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, pp. 17–30, ACM, 2016.
[9] E. Kokoris-Kogias, P. Vovanovic, L. Gasser, N. Gayilly, and B. Ford, “Omniledger: A secure, scale-out, decentralized ledger,” IACR Cryptology ePrint Archive, vol. 2017, p. 406, 2017.
[10] A. E. Gencer, R. van Renesse, and E. G. Sirer, “Short paper: Service-oriented sharding for blockchains,” in International Conference on Financial Cryptography and Data Security, pp. 393–401, Springer, 2017.
[11] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, “Coded MapReduce,” 53rd Allerton Conference, Sept. 2015.
[12] S. Li, M. A. Maddah-Ali, Q. Yu, and A. S. Avestimehr, “A fundamental tradeoff between computation and communication in distributed computing,” IEEE Transactions on Information Theory, vol. 64, Jan. 2018.
[13] K. Lee, M. Lam, R. Pedarsani, D. Papailiopoulos, and K. Ramchandran, “Speeding up distributed machine learning using codes,” IEEE Transactions on Information Theory, vol. 64, no. 3, pp. 1514–1529, 2018.
[14] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, “A unified coding framework for distributed computing with straggling servers,” IEEE Workshop on Network Coding and Applications, Sept. 2016.
[15] S. Dutta, V. Cadambe, and P. Grover, “Short-dot: Computing large linear transforms distributedly using coded short dot products,” in NIPS, pp. 2100–2108, 2016.
[16] Q. Yu, M. A. Maddah-Ali, and A. S. Avestimehr, “Polynomial codes: an optimal design for high-dimensional coded matrix multiplication,” in NIPS, pp. 4406–4416, 2017.
[17] C. Karakus, Y. Sun, S. Digvaji, and W. Yin, “Straggler mitigation in distributed optimization through data encoding,” in NIPS, pp. 5440–5448, 2017.
[18] R. Tandon, Q. Lei, A. G. Dimakis, and N. Karampatziakis, “Gradient coding: Avoiding stragglers in distributed learning,” in Proceedings of the 34th International Conference on Machine Learning, pp. 3368–3376, Aug. 2017.
[19] Q. Yu, S. Li, N. Raviv, S. M. M. Kalan, M. Soltankhotabi, and A. S. Avestimehr, “Lagrange coded computing: Optimal design for resiliency, security, and privacy,” in NIPS Systems for ML Workshop, 2018.
[20] Y. Gao and H. Nobuhara, “A proof of stake sharding protocol for scalable blockchains,” Proceedings of the Asia-Pacific Advanced Network, vol. 44, pp. 13–16, 2017.
[21] M. Zamani, M. Movahedi, and M. Raykova, “Rapidchain: A fast blockchain protocol via full sharding,” Cryptology ePrint Archive, 2018. https://eprint.iacr.org/2018/460.pdf.
[22] S. Bano, M. Al-Bassam, and G. Danezis, “The road to scalable blockchain designs,” USENIX: login: magazine, 2017.
[23] Z. Ren and Z. Erkin, “A scale-out blockchain for value transfer with spontaneous sharding,” e-print arXiv:1801.02531, 2018.
[24] H. Yoo, J. Yim, and S. Kim, “The blockchain for domain based static shading,” in 2018 17th IEEE International Conference On Trust, Security And Privacy In Computing And Communications/12th IEEE International Conference On Big Data Science and Engineering (TrustCom/BigDataSE), pp. 1689–1692, IEEE, 2018.