PartitionChain: A Scalable and Reliable Data Storage Strategy for Permissioned Blockchain

Zhengyi Du, Xiongtao Pang, and Haifeng Qian

Abstract—Blockchain, a specific distributed database which maintains a list of data records against tampering and corruption, has aroused wide interests and become a hot topic in the real world. Nevertheless, the increasingly heavy storage consumption brought by the full-replication data storage mechanism, becomes a bottleneck to the system scalability. To address this problem, a reliable storage scheme named BFT-Store (Qi *et al.* 2020), integrating erasure coding with Byzantine Fault Tolerance (BFT), was proposed recently. While, three critical problems are still left open: (i) The complex re-initialization process of the blockchain when the number of nodes varies; (ii) The high computational overload of downloading data; (iii) The massive communication on the network. This paper proposes a better trade-off for blockchain storage scheme termed PartitionChain which addresses the above three problems, maintaining the merits of BFT-Store. First, our scheme allows the original nodes to merely update a single aggregate signature (e.g., 320 bits) when the number of nodes varies. Using aggregate signatures as the proof of the encoded data not only saves the storage costs but also gets rid of the trusted third party. Second, the computational complexity of retrieving data by decoding, compared to BFT-Store, is greatly reduced by about 2^{18} times on each node. Third, the amount of transmitted data for recovering each block is reduced from $O(n)$ (assuming n is the number of nodes) to $O(1)$, by partitioning each block into smaller pieces and applying Reed-Solomon coding to each block. Furthermore, this paper also introduces a reputation ranking system where the malicious behaviors of the nodes can be detected and marked, enabling PartitionChain to check the credits of each node termly and expel the nodes with misbehavior to the specific extent. Comparing with BFT-Store, our scheme allows blockchain system to suit dynamic network with higher efficiency and scalability.

Index Terms—Blockchain storage, reed-solomon coding, Byzantine fault tolerance, system scalability, dynamic network

1 Introduction

Blockchain, first proposed in 2008 as a solution to the double-spending problem existing in purely peer-to-peer version of electronic cash system in [2], [3], has aroused interests in various fields for recent decades. Generally, blockchain is well known as a special shared-database which integrates cryptology, Peer-to-peer network and consensus mechanism, in order to maintain the integrity of data and prevent from being deceived by frauds [4], [5]. The blockchain technology has been applied in a wide range of popular scenarios recently, containing the Internet of Things [6], [7] and electronic invoice [8], and accordingly the amount of stored data on the chain increases constantly. Based on the latest statistics shown in Fig. 1, the size of Bitcoin blockchain system, for instance, has exceeded 340 gigabytes in June 2021, and still has a rapid growth. In order to ensure data consistency in blockchain, most nodes in the system are required to store all the historical data in the existing blockchain systems. According to this full-replication storage strategy, the storage overhead per block is $O(n)$, where n is the number of nodes in the system. Namely, the storage overhead of each block increases linearly with respect to the number of nodes in the system, which leads to a huge storage cost and may even break through the storage capacity of a single node when the system is extended and more nodes join it. Apparently, the high-speed increase of storage consumption for maintaining the complete ledger [9], is becoming a barrier to the system scalability.

1.1 Related Work

When the concept of blockchain was first formed in [2], we suppose that the blocks are generated every 10 minutes. Apparently, the storage consumption of the data on the chain will increase continuously, which impedes the system scalablity. Thus, some solutions are offered along with the proposal of blockchain. One primary method is to reclaim disk space on the nodes, in which the previous transactions will be discarded [2]. To achieve this without breaking the hash of blocks, Satoshi *et al*. [10] suggest to hash the transactions in a Merkle-Tree [11], [12], which allows the blocks to merely include the root hash of the tree in the block header. Through this method, the size of each block header is roughly decreased to 200 bytes; meanwhile, the nodes can still verify the payments through the root hash. To further reduce the storage consumption, a new kind of nodes termed light-nodes is introduced in [13], [14]. In contrast to the traditional full-nodes who preserving the complete blocks, the light-nodes merely store the block headers.

Addition to the above, other technologies and mechanisms are also introduced recently, to reduce the storage

• *The authors are with Software Engineering Institute, East China Normal University, Shanghai 200050, China. E-mail: {51205902043, 51205902129}@stu.ecnu.edu.cn, hfqian@cs.ecnu.edu.cn.*

Manuscript received 21 Sept. 2021; revised 6 Dec. 2021; accepted 12 Dec. 2021. Date of publication 20 Dec. 2021; date of current version 7 Mar. 2023.
This work was supported in part by NSFC-ISF Joint Scientific Research Program under Grant 61961146004 and in part by the Innovation Program of Shanghai Municipal Education Commission under Grant 2021-01-07-00-08-E00101.
(Corresponding author: Haifeng Qian.)
Recommended for acceptance by Y. Zhang.
Digital Object Identifier no. 10.1109/TKDE.2021.3136556

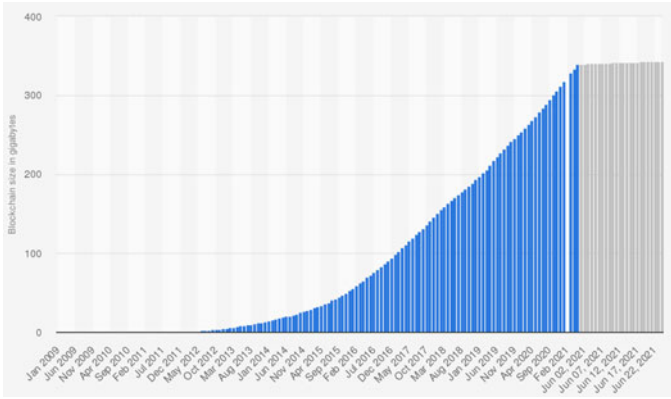

Fig. 1. Size of bitcoin blockchain system from January 2009 to June 2021.[1]

bloating problem. Dimakis *et al*. [15] suggest to apply erasure codes and network coding to distributed storage [16], [17]; thus, the intermediate nodes merely store the previously-received input data, and generate the output data by computing certain functions of the input. Moreover, utilizing micropayment channels [18], [19] seems to be another feasible solution. A mass of micropayments can be offloaded from the chain to a trusted custodian, and just the latest results after these micropayments are stored on the chain. In addition, the recent works [20], [21] introduce lightning network, which transfers the micro-payment transactions from online to offline, so that only the final results of the transactions are preserved on the chain. However, the aforementioned schemes do not fundamentally overturn the full-replication storage strategy. In order to avoid the adverse effects of the full-replication strategy as well as to guarantee the availability of all the blocks, a reliable storage scheme named BFT-Store for permissioned blockchain is proposed recently in [1].

BFT-Store combines RS (Reed-Solomon) coding and PBFT (Practical Byzantine Fault Tolerance) Protocol, so as to reduce the storage consumption per block from $O(n)$ to $O(1)$ in regard to the number of nodes n, by enabling each node in the system to store just a part coded blocks (dubbed as chunks) rather than the whole chain. BFT-Store adopts $(n-2f,n)$-RS schema which means to encode every $n-2f$ original blocks into n chunks by adding $2f$ additional redundant blocks (dubbed as parities), and all the original blocks can be recovered through random $n-2f$ chunks. After encoding, these n chunks are distributed to n corresponding nodes through some specific algorithms. Since there are at most f Byzantine nodes and f crashed honest nodes in a system with $n \geqslant 3f+1$ nodes based on the precondition of PBFT [22], each node is guaranteed to receive at least correct $n-2f$ chunks while decoding, ensuring data availability.

BFT-Store also introduces an online re-encoding process. Assume that there are n nodes in the system, and distinctly the blocks are encoded by $(n-2f,n)$-RS, once a new node joins the system, the number of node increases to $N=n+1$ and the threshold of malicious nodes becomes $F=\lfloor\frac{N-1}{3}\rfloor$ accordingly. Therefore, the schema of RS should be adjusted to $(N-2F,N)$-RS, leading to an online re-encoding process. The whole system broadcasts and downloads chunks to recover them into original blocks, and re-encodes them through the updated schema afterwards. The re-encoding process for removal of nodes is similar.

Undoubtedly, BFT-Store [1] provides an innovative storage partition, subverting the full-replication strategy for permissioned blockchain; however, there existing three critical problems in this scheme:

- *Complex re-initialization of the blockchain.* Each time the number of nodes n changes, all the nodes have to participate in a blockchain update process. To re-initialize the system, each node devotes to broadcasting and downloading all the chunks for recovery and sequentially re-encoding the recovered blocks through the updated RS schema.
- *High computational complexity of coding.* Before replying the data to a client, a node needs to decode the requested chunk first in BFT-Store, and thus the computational complexity of decoding becomes a predominant factor influencing the response time and system throughput. In BFT-Store, the computational complexity of decoding and encoding per block are $O(T^2 \cdot n^3)$ and $O(T^2 \cdot n^2)$ respectively, where T is the assumed size of a block. Obviously, frequent decoding and encoding badly increase the computational overheads and impair the performance of the nodes in BFT-Store.
- *Massive communication on the network.* BFT-Store encodes every $n-2f$ blocks into n chunks through RS code. Therefore, each time a block needs to be recovered, at least $n-2f$ blocks will be transmitted through the network. Namely, the amount of communication over the whole network is $O(n)$, which will impair the performance of the network due to the specifics of P2P networks [23] when increasing nodes join the system.

1.2 Our Contribution

In addition to the aforementioned existing implementation problems, BFT-Store also leaves two points open:

- *Reliance on the trusted third-party.* BFT-Store employs TS (Threshold Signature) to verify the correctness of each chunk [1]. However, most existing TS schemes depend on the attendance of a TTP (Trusted Third-Party) or a center node termed dealer [24], [25], which is contradictory to decentralization in blockchain. Relying on a TTP will probably incur unexpected data damages and unavoidable extra costs [17], [26]. Additionally, it is troublesome to appoint a third-party who is completely trusted by all the participants in practical application scenarios.
- *Lack of measures against malicious behaviors.* In BFT-Store, the nodes merely check the correctness of received chunks, ignoring the forged messages and the dishonest nodes. That is, the nodes can exhibit a series of dishonest behaviors without any probable costs and punishment. Apparently, BFT-Store lacks

1 https://www.statista.com

TABLE 1
Comparison Between BFT-Store and PartitionChain

	BFT-Store	PartitionChain
Storage Consumption	$O(1)$	$O(1)$
Operations in System Re-initialization on Original Nodes	$O(n)$	$O(1)$
Transmitted Data for decoding	$O(n)$	$O(1)$
Complexity of Decoding	$O(T^2 \cdot n^3)$	$O(Tc \cdot n^2)$
Complexity of Encoding	$O(T^2 \cdot n^2)$	$O(Tc \cdot n)$
Measures against Dishonest Nodes	None	✓
Security Strategy	TS (with TTP)	CLAS (without TTP)

an effective measure to detect the malicious behaviors and remove dishonest nodes.

To address the above, we propose a better storage partition scheme named PartitionChain, which optimizes the performance of the system and re-initialization process, meanwhile, maintaining other advantages of BFT-Store. To avoid the reliance on a TTP, we adopt an aggregate signature scheme instead of Threshold Signature to guarantee the correctness of the encoded data. Moreover, we design an audit mechanism for detecting dishonest behaviors and punishing malicious nodes. The major advantages and functionalities of PartitionChain are listed as follows:

1) PartitonChain maintains the advantage of BFT-Store that it reduces the storage overhead of each block to $O(1)$. Moreover, when the number of nodes changes, it relieves the system overload by optimizing the application of RS coding. For example, when a fresh node joins the system, PartitionChain enables the original nodes just to update their aggregate signatures respectively, instead that all the nodes update both their chunks and signatures. Thus, the computation consumption of system re-initialization can be greatly reduced.
2) In order to reduce the computational complexity, PartitionChain partitions each block into smaller pieces on account of the size of a single block before encoding. Through this method, the computational complexity of decoding per block is declined from $O(T^2 \cdot n^3)$ to $O(Tc \cdot n^2)$, where c is the fixed size of each piece of an original block. Similarly, the computational complexity of encoding per block is reduced from $O(T^2 \cdot n^2)$ to $O(Tc \cdot n)$.
3) In PartitionChain, we apply RS encoding polynomial to each block rather than every $n - 2f$ blocks to reduce the amount of communicated data. Thus, when a block needs to be recovered, each node merely broadcasts the encoded pieces of the original block. The size of each piece is limited to $c = 1024$ bits in this paper. Therefore, PartitionChain decreases the amount of communicated data of the whole system from $O(n \cdot T)$ in [1] to $O(T)$, assuming the size of each original block is T.
4) On the contrary to adopting TS in BFT-Store, PartitionChain adopts CLAS (Certificateless Aggregate Signature) [27] to verify the correctness of stored information. After each participant gets his partial private key from a KGC (Key Generation Center), the whole system can completely eliminate the dependence on any third-party. In addition, even the KGC does not know the secret keys of the participants, and thus the adversary has no chance to steal the keys by attacking the KGC.
5) PartitionChain introduces an audit mechanism that detects the malicious behaviors and expels dishonest nodes. PartitionChain checks the credits of the nodes termly, and the nodes whose credits are below a preset lower bound will be removed from the system.

The comparison of the performance and functionalities between PartitionChain and BFT-Store are summarized in Table 1.

1.3 Organization

The rest of this paper is organized as follows:

Section 2 describes the assumptions of the model, the tools utilized in PartitionChain and the goals of this paper. Section 3 proposes the realized functionalities and algorithms of PartitionChain. Section 4 analyzes the performance of PartitionChain and Section 5 presents the corresponding experimental evaluations. Section 6 concludes this paper.

2 Preliminaries

2.1 Assumptions

PartitionChain is an optimized combination of PBFT and RS coding, applied in permissioned blockchains where a group of identifiable participants who have a common goal but do not fully trust each other [28], [29]. We set two assumptions for PartitionChain as follows on account of the characteristics of permissioned blockchains and the preconditions of PBFT:

1) The existing malicious nodes in the system may randomly exhibit a series of bad behaviors (e.g., keep silence, send conflicting messages to different nodes in the system), in order to prevent the system from reaching consensus. Additionally, the honest nodes in the system may keep silence due to malfunction for any reasons. In order to tolerate these two kinds of failures, we assume that in a system containing n nodes, there are at most f malicious nodes may behave dishonestly and f honest nodes may fail to work for some reason at the same time, where $f = \lfloor \frac{n-1}{3} \rfloor$.
2) PBFT provides both safety and liveness of a system [22]. It can provide safety in an asynchronous system, while it must rely on synchronization to provide liveness. Therefore, we build a partial synchronous model where a sent message may be delayed for some reason

but is bound to be received by its destination within a certain delay. In addition, all the channels in the model are assured to be secure. This partial synchronous model is feasible to be applied in real systems, as long as the network faults can be eventually repaired [22].

2.2 Tools

Reed-Solomon Coding. RS coding [30], [31], a widely-applied EC (erasure coding), assumes that F_q is a finite field, and $\alpha_0, \ldots, \alpha_{n-1}$ are n distinct elements (dubbed as evaluation points) from F_q. For an original message $m = (m_0, m_1, \ldots, m_{k-1})$, (k, n)-RS generates a polynomial evaluated by the n distinct evaluation points respectively, where $k \leqslant n \leqslant q$. A participant can recover the original message so long as it receives random k of the n values. The corresponding RS polynomial $f_m(X)$ of degree $k-1$ is defined as

$$f_m(X) = \sum_{i=0}^{k-1} m_i X^i.$$

Accordingly, the original message fragments are mapped to n values through RS coding as follows:

$$RS(m) = (f_m(\alpha_0), f_m(\alpha_1), \ldots, f_m(\alpha_{n-1})).$$

Each participant N_i in the system computes and then preserves its own value $f_m(\alpha_i)$ evaluated by the assigned evaluation point α_i. When receiving a request for the original message from a client, the participant just needs to receive k different values: $f_m(\alpha_1), \ldots, f_m(\alpha_k)$ from other participants for decoding. Then the participant substitutes the k received values for $f_m(X)$, and the k corresponding evaluation points: $\alpha_0, \ldots, \alpha_{k-1}$ for X into the RS polynomial and obtains a k-element linear equation set as follow:

$$\begin{cases} m_0 + m_1\alpha_1 + m_2\alpha_1^2 + \cdots + m_{k-1}\alpha_1^{k-1} = f_m(\alpha_1) \\ \qquad\qquad \vdots \\ m_0 + m_1\alpha_k + m_2\alpha_k^2 + \cdots + m_{k-1}\alpha_k^{k-1} = f_m(\alpha_k) \end{cases}.$$

The participant will solve this equation set and apparently the solutions $(m_0, \ldots, m_{k-1})$ are the original message fragments. For a k-element linear equation set, the time complexity to solve it by Gaussian elimination method [32] is $O(l^2 \cdot k^3)$, where l is the assumed size of each element in the equation set. Through RS coding, each participant merely needs to preserve its own value instead of the complete message while guaranteeing the same reliability as full-replication strategy.

Practical Byzantine Fault Tolerance Protocol. PBFT is widely-adopted in permissioned blockchain system as a consensus protocol and has been proved to perform well in [33]. Based on the precondition of PBFT [22], it can guarantee the consistency of information, provided there are at most $f = \lfloor \frac{n-1}{3} \rfloor$ malicious nodes which can cause Byzantine failures in the system.

In PBFT, a newly-generated block is committed in a view which contains a primary node (selected in Round-Robin order [34] or other methods) and several backup nodes. After a client sends transactions which are packaged as a new block, the consensus on this block will be reached through three phases called Pre-prepare, Prepare and Commit respectively [1], [35]. In Pre-prepare phase, the primary node proposes the new block and broadcasts a signed pre-prepare message to all the backups after receiving a request from a client. Once a backup node accepts the pre-prepare message, it enters Prepare phase and then broadcasts a signed prepare message to all the nodes in the view. If a node receives $2f$ different prepare messages which matches its accepted pre-prepare message, it becomes prepared and broadcasts a signed commit message to others. Again, if a node receives $2f+1$ different commit message matching its accepted pre-prepare message and prepare message, it commits the newly-proposed block and replies to the client. The block will be considered verified and can be appended to the blockchain so long as the client receives $f+1$ same replies from different nodes in the view.

Certificateless Aggregate Signature. CLAS scheme [27] is proposed on the basis of Schnorr Signature [36]. On the contrary to Identity-based public key cryp-tography (ID-PKC) [37] and Certificateless public key cryptography (CL-PKC) [38], CLAS enables participants to generate their complete private key and public key by themselves. PartitionChain adopts CLAS as the verification of correctness of the stored evaluations on the nodes for the following reasons. First, instead of relying on the TTP to update the secret keys of the participants and system parameters for signing each time the number of participants n varies in TS [1], [39], CLAS merely requires KGC to provide the partial private key to each participant. Once the public key and complete private key of a participant are generated, they maintain invariable whether n changes or not, achieving totally decentralization [27]. Second, CLAS reduces the total signature length (e.g., the size of a CLAS is always under 320 bits) as well as the computational cost for signature verification [40].

In general, for n certain users $u_1, u_2, \ldots, u_n$ whose identities are $ID_1, ID_2, \ldots, ID_n$ with n corresponding public keys $pk_1, pk_2, \ldots, pk_n$ and some state information δ, CLAS scheme can compress the n signature $\sigma_1, \sigma_2, \ldots, \sigma_n$ on the given messages $m_1, m_2, \ldots, m_n$ into a single aggregate signature σ [27]. CLAS also provides an algorithm for verification. The verification algorithm takes the user identities, the related public keys, the original messages and the same state information δ of the sign process as inputs, and then it will compute and output whether the aggregate signature σ is valid.

2.3 Goals

PartitionChain inherits the fundamental advantage of BFT-Store that reduces the storage consumption per block to $O(1)$. Besides, this paper also aims to solve the open problems left by BFT-Store and add new functionalities detailedly illustrated in Chapter 1. According to these requirements, PartitionChain is designed on the basis of the following aims:

1) *Availability.* So long as there are at most $f = \lfloor \frac{n-1}{3} \rfloor$ malicious nodes in the system, PartitionChain can guarantee that any original block can be recovered by all the non-faulty nodes (i.e., honest nodes which work properly without any malfunctions) through decoding, so that a client can request for data successfully at any time.

TABLE 2
Description of the Notations Mentioned in This Paper

Notation	Description
$B(h)$	original block with hash h
$B_i(h)$	ith primary-piece of $B(h)$
$b_{i,j}(h)$	jth second-level-piece of $B_i(h)$
$f_{h,i}(X)$	RS-polynomial of $B_i(h)$
$f_{h,i}(s)$	evaluation of $B_i(h)$ stored by node s
$F_h(s)$	evaluation set stored by node s for $B(h)$
$\sigma_{h,i,s}$	CLAS stored by node s on $f_{h,i}(s)$
$\sigma_{h,s}$	CLAS stored by node s on $F_h(s)$

2) *Scalability*. The complexity of storage consumption per block of PartitionChain will maintain within $O(1)$ while the number of nodes n increases. In addition, we attempt to reduce the computational costs of system re-initialization when the number of nodes varies.
3) *Efficiency*. Due to the extra calculation and communication in data recovery compared to the traditional full-replication strategy, the response time of clients' request to read a block will certainly increase [1]. This paper aims to lower the computational complexity of decoding and encoding and minimize the loss of read performance.
4) *Validity*. Byzantine nodes exist in permissioned blockchain, randomly behaving dishonestly to impede the system consensus. On the premise of achieving availability and scalability, the validity of the system should be ensured and the number of existing malicious nodes should be reduced to a certain extent.

2.4 Notations

To make it clear to illustrate, we first present the principal notations mentioned in the next sections and the corresponding descriptions in Table 2.

3 Design

3.1 Block Partition

In BFT-Store, at least $n-2f$ complete blocks are transmitted over the network when a node needs to recover an original block, which brings about massive communication bits over the network. In order to relieve the communication overheads, we attempt to partition each original block before encoding in PartitionChain. The number of partitioned pieces is determined by the block size together with the number of nodes in the system. Each node preserves encoded pieces of every original block rather than the entire chunk, hence the realization of lower communication and storage consumption.

The computational complexity of RS encoding and decoding are respectively $O(c^2 \cdot n^2)$ and $O(c^2 \cdot n^3)$, where c is the assumed size of the pieces of an original block. Intuitively, the size of pieces is a dominant factor influencing the overall throughput of the system. Thus, we limit the size of each piece to $c = 1024$ bits, to avoid the heavy computational overloads on the nodes and improve system performance. In order to guarantee the availability of the blocks, meanwhile, we intend to apply $(n-2f, n)$-RS to each original block based on the fault tolerance of PBFT [1]. In existing permissioned blockchain systems, the number of nodes n is generally under 100, and the size of a single block is about 1 MB; predictably, partitioning once is insufficient to bound the size of each piece to 1 Kb. Therefore, we partition each block twice before encoding in PartitionChain. To make it clear, we term the pieces generated in the first partitioning as primary-pieces and the pieces generated in the second partitioning as second-level-pieces.

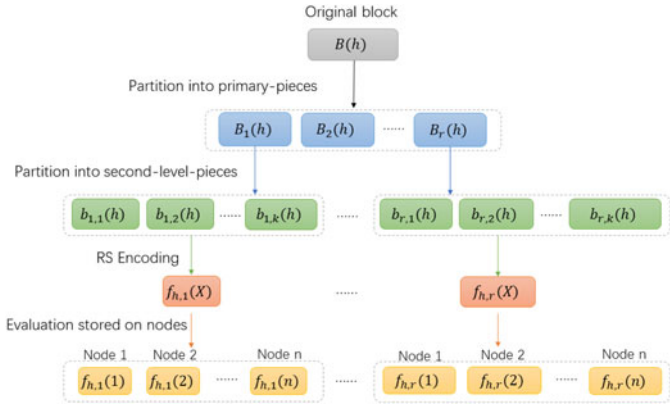

Fig. 2. The procedure of block partition and RS encoding.

For an original block $B(h)$ of size T, where h is the unique hash value of the block, we initially determine the number of primary-pieces. According to PBFT, the number of second-level-pieces k partitioned from each primary-piece should equal to the lower bound (i.e., $n-2f$) of the honest nodes in the system. Assuming that we will partition $B(h)$ into r primary-pieces, r can be deduced from the following equation:

$$\frac{T}{r(n-2f)} = c.$$

Here, we take $T = 1$ MB, $f = 33$, $n = 3f + 1 = 100$ as an example, so we can determine the value of r is

$$r = \frac{T}{c(n-2f)} \approx 2^8.$$

Then, as shown in Fig. 2, each block is partitioned by the following two steps in PartitionChain:

1) Partition an original block $B(h)$ into r primary-pieces

$$B(h) = \{B_1(h), B_2(h), \ldots, B_r(h)\}.$$

In our example, r approximately equals to 2^8 and thus the size of each primary-pieces is 2^{15} bits. To further reduce the size of the primary-pieces, we need to partition each of them again into smaller second-level-pieces.
2) Partition each primary-piece $B_i(h)$ where $1 \leqslant i \leqslant r$ into $k = n - 2f$ smaller second-level-pieces

$$B_i(h) = \{b_{i,1}(h), b_{i,2}(h), \ldots, b_{i,k}(h)\},$$

which are represented as the green ones in Fig. 2. According to our assumption, r is approximately equal to 2^5; thus, the size of each second-level-piece is reduced to 2^{10} bits.

3.2 RS Encoding

After partitioning, we select a large prime p which is large enough, (e.g., p is greater than 2^{1024} in our example), and then convert every binary-stored second-level-piece $b_{i,j}(h)$ ($1 \leqslant i \leqslant r$ and $1 \leqslant j \leqslant k$) to a p-base number.

In $GF(p)$ field, each partitioned primary-piece

$$B_i(h) = \{b_{i,1}(h), b_{i,2}(h), \ldots, b_{i,k}(h)\},$$

is mapped by a $(n, n-2f)$-RS (in our example it turns to be $(100, 34)$-RS) to a degree $n-2f-1$ polynomial $f_{h,i}(X)$ which is defined as

$$f_{h,i}(X) = \sum_{j=1}^{k} b_{i,j}(h) \cdot X^{j-1},$$

where $X \in \{\alpha_0, \ldots, \alpha_{n-1}\}$.

As Fig. 2 presents, the k second-level-pieces of a primary-piece are mapped to n specific evaluations, which are determined by substituting the corresponding evaluation points and calculating the above polynomial, and then distributed to the n nodes in the system. In order to determine the distinct evaluation point X for each node, we sort all the nodes in the system according to their public keys and take the position s of a node in the sort as its corresponding evaluation point. That is, after encoding, the node N_s whose index is s, preserves a specific evaluation $f_{h,i}(s)$ for the primary-piece $B_i(h)$. It can be seen from Fig. 2 that, for a block $B(h)$ which is divided into r second-level-pieces

$$B_1(h), B_2(h), \ldots, B_r(h),$$

each node s is required to calculate r generated polynomials and preserve the corresponding evaluation set

$$F_h(s) = \{f_{h,1}(s), f_{h,2}(s), \ldots, f_{h,r}(s)\}.$$

Expanding to the whole chain with t blocks

$$B(h_1), B(h_2), \ldots, B(h_t),$$

for instance, and thus $t \cdot r$ degree-$n-2f-1$ polynomials are generated. For a node N_s in the system, it maintains its own t evaluation sets

$$F_{h_1}(s), \ldots, F_{h_t}(s),$$

which can be also presented as $t \cdot r$ evaluations

$$f_{h_1,1}(s), \ldots, f_{h_1,r}(s), \ldots, f_{h_t,1}(s), \ldots, f_{h_t,r}(s).$$

3.3 Certificateless Aggregate Signature

In consideration of the existence of Byzantine nodes [41], [42] which may arbitrarily behave maliciously to violate the data integrity in the system, we adopt CLAS [27] to ensure the complete and correctness of the stored evaluations on the nodes.

After a new block is generated and partitioned, each node is required to calculate and provisionally preserve all the evaluations of each encoded primary-pieces for the next verification. The verification and signing process is embedded in the Commit phase of PBFT. In this phase, each node N_s first signs its all computed evaluations for the block $B(h)$: $f_{h,i}(s)$ (where $1 \leqslant i \leqslant r$ and $1 \leqslant s \leqslant n$), and then broadcasts its signatures and index s embedded in the commit messages.

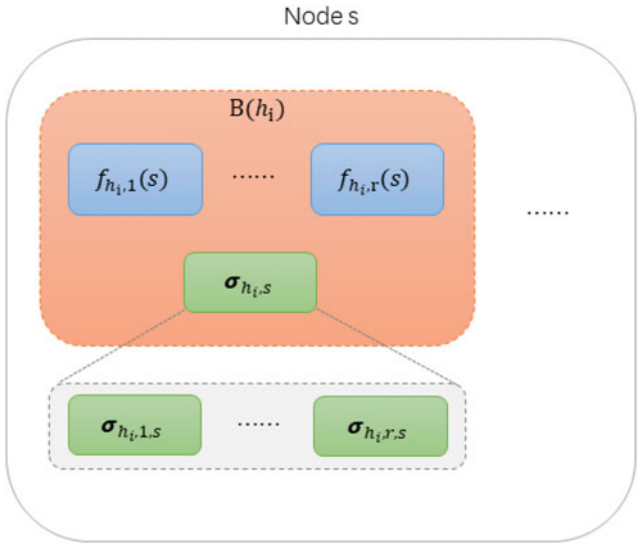

Fig. 3. The evaluations and the corresponding aggregate signatures for each block a node N_s needs to store.

When node N_s receives the commit message from another node N_q, it checks the validity of the evaluation $f_{h,i}(s)$ computed by N_q through verifying the related signature. If the signature is valid, the node N_s will extract the signature for the next aggregation. Until the node N_s receives sufficient commit messages from different nodes and extracts the $f+1$ valid signatures on the evaluation $f_{h,i}(s)$, it aggregates them into a single signature $\sigma_{h,i,s}$ with a constant length (the length of CLAS is always under 320 bits). This aggregate signature is regarded as the proof of the correctness of the evaluation. After that, each node preserves its own evaluations together with the related aggregate signatures.

Since CLAS can be performed incrementally [39], the signatures

$$\sigma_{h,1,s}, \sigma_{h,2,s}, \ldots, \sigma_{h,r,s},$$

on each encoded primary-piece of $B(h)$ stored on node N_s can be further aggregated into a single signature $\sigma_{h,s}$, which is regarded as the proof of the evaluation set $F_h(s)$ for block $B(h)$ stored on node N_s. Through this method, the nodes can verify the received evaluations by just verifying a single aggregate signature instead of a set of signatures. After all the nodes have aggregated the signatures for the evaluations which they need to preserve, they can remove the calculated evaluations for other nodes. The evaluations and relative aggregate signatures that a node N_s needs to store for a series of blocks $\{B(h_1), B(h_2), \ldots\}$ are shown in Fig. 3.

Algorithm 1 shows the details of partitioning and encoding process for an original block $B(h)$, which is implemented on node N_s in PartitionChain.

First, the original block $B(h)$ is partitioned into r primary-pieces $\mathcal{B} = \{B_1(h), \ldots, B_r(h)\}$ through function $Partition()$ (Line 1). Then, for each primary-piece $B_i(h)$, where i is in the range from 1 to r, we partition it into $k = n - 2f$ second-level-pieces $\mathcal{B}_i = \{b_{i,1}(h), \ldots, b_{i,k}(h)\}$ based on the assumption of

PBFT (Line 2-3). After that, each set of second-level-pieces partitioned from one primary-piece is encoded into n evaluations $\mathcal{F}_i = \{f_{h,i}(1), \ldots, f_{h,i}(n)\}$ by function $RSEncode()$ (Line 4-5). Sequentially, node N_s signs every computed evaluation (Line 6).

Algorithm 1. Partition and Encoding in PartitionChain

Input: node N_s, original block $B(h)$
1: $\mathcal{B} = \{B_1(h), \ldots, B_r(h)\} \leftarrow$ Partition($B(h)$, r)
2: **for** i from 1 to r **do**
3: $\mathcal{B}_i = \{b_{i,1}(h), \ldots, b_{i,k}(h)\} \leftarrow$ Partition($B_i(h)$, k)
4: **for** j from 1 to n **do**
5: $\mathcal{F}_i = \mathcal{F}_i \cup \{f_{h,i}(j) \leftarrow$RSEncode($\mathcal{B}_i$, n, $n-2f$, j)$\}$
6: $H_i \leftarrow H_i \cup$ Sign($f_{h,i}(j)$, s)
7: **end for**
8: $\mathcal{F} \leftarrow \mathcal{F} \cup \{f_{h,i}(s)\}$
9: $\mathcal{H} \leftarrow \mathcal{H} \cup \mathcal{H}_i$
10: **end for**
11: broadcast $\mathcal{H}$ along with commit message
12: **if** receive $f+1$ commit messages **then**
13: $H_1, \ldots, H_{f+1} \leftarrow f+1$ signature sets
14: **for** i from 1 to r **do**
15: $signs \leftarrow \{H_1[f_{h,i}(s)], \ldots, H_{f+1}[f_{h,i}(s)]\}$
16: $\sigma_{h,i,s} \leftarrow$ Aggregate($signs$)
17: **end for**
18: $\sigma_{h,s} \leftarrow$ Aggregate($\sigma_{h,1,s}, \ldots, \sigma_{h,r,s}$)
19: store $\sigma_{h,s}$ together with $\mathcal{F}$
20: **end if**

After encoding and signing, node N_s packages the evaluations which it should preserve into $\mathcal{F}$ and the signatures into $\mathcal{H}$ respectively, and then it broadcasts the signature set $\mathcal{H}$ together with the commit message during Commit phase in PBFT (Line 8-11). If node N_s receives $f+1$ commit messages from different nodes, it aggregates the signatures on each evaluations which it needs to preserve (Line 12-18). In detail, it first extracts the $f+1$ signature sets from the commit messages (Line 13). For each evaluation $f_{h,i}(s)$, node N_s packages the $f+1$ signatures into $signs$ and then aggregates them into a single signature $\sigma_{h,i,s}$ with constant length via $Aggregate()$ (Line 14-17).

Gaining the r aggregate signatures: $\sigma_{h,1,s}, \ldots, \sigma_{h,r,s}$, node N_s aggregates them again into a signature $\sigma_{h,s}$, served as the proof of the evaluation set $\mathcal{F}$ (Line 18). Finally, node N_s preserves its own evaluation set $\mathcal{F}$ along with the corresponding aggregate signature $\sigma_{h,s}$ (Line 19).

3.4 Data Recovery

Through RS encoding, each node preserves its own evaluations rather than the whole historical data. According to the assumption of our model, there are at most $f = \lfloor \frac{n-1}{3} \rfloor$ malicious nodes in the system, and thus we can guarantee that at least $n-f$ evaluations for each primary-piece $B_i(h)$ are stored on the honest nodes. Even in the worst case that there are f honest nodes which are crashed for any reasons and do not reply to the request, it can be assured that at least $n-2f$ valid evaluations for each primary-piece $B_i(h)$ can be received for data recovery. Consequently, data availability of the system can be achieved through $(n-2f, n)$-RS schema.

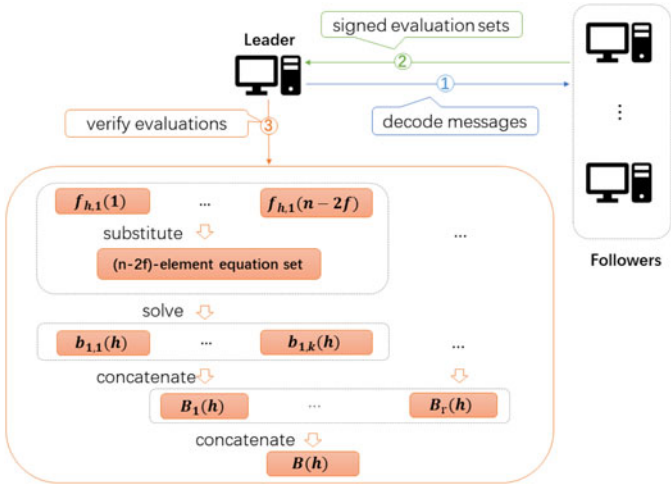

Fig. 4. Details of the data recovery process.

PartitionChain will nominate a node as the leader to be responsible for decoding while the remaining nodes are termed followers. In order to ensure that the elected leader is most likely to be honest, PartitionChain selects it from the nodes whose credit is higher than the lower bound by Robin-Round.

When the leader receives a request from a client to read a certain block $B(h)$, it will launch the data recovery process for the target block $B(h)$. The details of the process for block $B(h)$ is presented in Fig. 4, which contains following three steps:

1): The leader first broadcasts a decode message $< DECODE, h, n, n-2f >$ (where h is the unique hash value of the target block $B(h)$, and n and $n-2f$ indicate the related RS-scheme) to the followers in this process. Since there exist at most f malicious nodes and f faulty nodes according to the assumption, it can be guaranteed that the leader can receive at least $n-2f$ correct evaluation sets which are sufficient for decoding.

2): After receiving the decode message, each follower N_s replies its evaluation set $F_h(s)$, the corresponding aggregate signature $\sigma_{h,s}$ and its index s to the leader. For each primary-piece $B_i(h)$, the leader needs to receive at least $nndash;2f$ RS evaluations $f_{h,i}(k)$ $(1 \leqslant k \leqslant n)$ out of the n evaluations and corresponding node index k from different non-faulty nodes. The correctness of received evaluations can be verified through the corresponding aggregate signature by the leader. Therefore, if the leader detects that there exists fake evaluations, it just needs to rebroadcast the decoding message and repeat this step until it receives $nndash;2f$ valid evaluation sets and indexes.

3): The leader substitutes the $k = n-2f$ valid evaluations and related indexes into the RS polynomial respectively, so as to solve this k-element equation set, which is equal to calculate a k-order matrix. Intuitively, the solutions of this equation set are the original second-level-pieces: $b_{i,1}(h), \ldots, b_{i,k}(h)$ for $B_i(h)$. Sequentially, the primary-piece $B_i(h)$ can be recovered by concatenating all the recovered second-

level-pieces $b_{i,j}(h)$ according to the index j of each second-level-piece. As mentioned in previous sections, each primary-piece $B_i(h)$ corresponds to such a k-element equation set; thus, to recover all the primary-pieces, we need to solve altogether r such k-equation sets. After all r primary-pieces $B_i(h)$ are recovered in the same way illustrated above, the leader concatenates these primary-pieces by their indexes i to recover the original block $B(h)$.

Through the steps described above, the original block can be recovered and sent by the leader to the client.

Algorithm 2 detailedly illustrates how the leader recovers the data when it receives a request for a target block $B(h)$ from a client. When the leader L receives a request for a target block $B(h)$, it first broadcasts a decode message $< DECODE, h, n, n-2f >$ to all the followers (Line 2). The follower N_s replies its preserved evaluation set $\mathcal{F}_h(s)$, the corresponding aggregate signature $\sigma_{h,s}$ and its own index s back to the leader (Line 13-17).

Algorithm 2. Data Recovery of a Requested Block

1: (Processing on leader L)
2: broadcast decode message $< DECODE, h, n, n-2f >$ to the followers
3: **if** receive $n-2f$ evaluation messages **then**
4: $\mathcal{F}_h(1), \ldots, \mathcal{F}_h(n-2f) \leftarrow n-2f$ evaluation sets
5: **for** i from 1 to r **do**
6: $f_i \leftarrow \{\mathcal{F}_h(1)[f_{h,i}(1)], \ldots, \mathcal{F}_h(n-2f)[f_{h,i}(n-2f)]\}$
7: $\mathcal{B}_i \leftarrow \text{RSDecode}(f_i, n, n-2f)$
8: $\mathcal{B} \leftarrow \mathcal{B} \cup \mathcal{B}_i$
9: **end for**
10: $\mathcal{B} \leftarrow \text{Concatenate}(\mathcal{B})$
11: return $\mathcal{B}$
12: **end if**
13: (Processing on followers N_s)
14: **if** receive the decode message **then**
15: $msg \leftarrow < \mathcal{F}_h(s), \sigma_{h,s}, h, s >_{\sigma_s}$
16: SendMessage(L, msg)
17: **end if**

After the leader receives $n-2f$ messages including the evaluation sets and CLAS from different followers, it extracts the evaluations for each primary-piece and decode them into the corresponding second-level-pieces by function $RSDecode()$ sequentially (Line 4-10). This process is performed r times for r primary-pieces. After that, the leader concatenates the recovered second-level-pieces by their unique indexes through function $Concatenate()$ (Line 10). Finally, it replies the target block $B(h)$ to the client (Line 11).

3.5 System Re-Initialization Process

In the permissioned blockchain, the number of nodes n may change if a new node is accepted to join the system or an existing node decides to leave. The variations of nodes can be divided into two main categories which are outlined in Fig. 5.

When a new node is accepted to join the system, the updated number of nodes $N = n+1$ is greater than the previous number n, so the updated $(n-2f, n+1)$-RS schema can still guarantee the data availability. If an existing node intends to quit the system, there are two different strategies

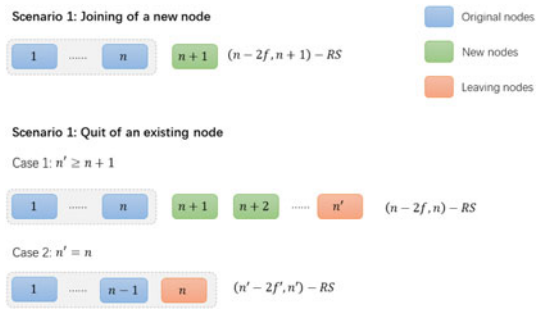

Fig. 5. Different scenarios in the variation of the number of nodes.

provided by PartitionChain. In the first scenario, after a node leaves, the current number of nodes n' is still larger than n, and thus $(n-2f, n)$-RS schema still works. Therefore, in this situation, it is unnecessary to update the RS scheme. In the second scenario, the current number of node n' is smaller than n which means $(n-2f, n)$-RS can no longer ensure that the leader can receive enough correct evaluations. Consequently, PartitionChain will launch a system re-initialization with the updated $(n'-2f', n')$-RS where $f' = \lfloor \frac{n'-1}{3} \rfloor$.

In this paper, we assume that the $(n, n-2f)$-RS schema is still resultful for the system when a node leaves because PartitionChain has a dishonest behavior audit mechanism which can maintains the number of malicious nodes at a low level in practical; therefore, we principally focus on the first two scenarios.

Joining of New Nodes. When a new node intends to join the system, the number of nodes changes from n to $N = n+1$ and accordingly the bound of malicious nodes changes to $F = \lfloor \frac{N-1}{3} \rfloor$. In order to relieve the calculation stress of re-encoding process, we adopt $(n-2f, n+1) - RS$ schema which can still guarantee the data availability for $n+1-2f < n-2f$, indicating that the leader is capable of receiving $n-2f$ evaluations in decoding.

After the new node is approved to join the system, the re-initialization of the whole chain is launched. Here, we present the details of the re-initialization of an original block $B(h)$ in Fig. 6 and Algorithm 3 (Note that the details of re-encoding and decoding are not explained in Algorithm 3 since they have been illustrated in Algorithms 1 and 2):

1): The new node N_{n+1} broadcasts a decode message $< DECODE, h, n, n-2f >$ for block $B(h)$ to all the original nodes (Line 2).
2): Receiving the decode message, each original node N_s replies its preserved evaluation set $\mathcal{F}_h(s)$ of the block $B(h)$, the aggregate signature $\sigma_{h,s}$, the hash h and its index s to the new node N_{n+1} (Line 14-17). For recovering each second-level-piece, the new node needs to receive at least $n-2f$ correct evaluation sets from different nodes (Line 4). It checks the validity of the received evaluations by verifying the corresponding signature.
3): After decoding, the new node N_{n+1} calculates and signs the evaluations $\mathcal{F}$ on all the other nodes (Line 5-6). Then it broadcasts a re-initialize message $< RE\_INITIALIZE, \mathcal{B}, \mathcal{H}_{n+1}, h, n+1, n-2f >$, where $\mathcal{B}$ is the recovered second-level-pieces set, $\mathcal{H}_{n+1}$ is the signature set on $\mathcal{F}$, and $n+1$ and $n-2f$ indicate

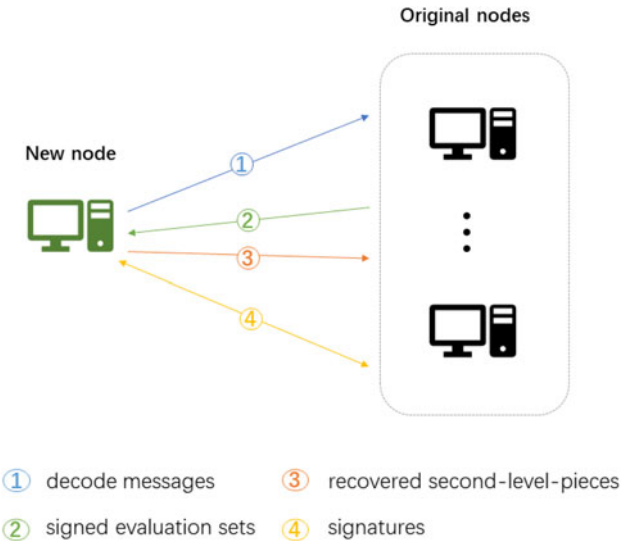

Fig. 6. The main steps and communicated data in the system re-initialization for joining of a new node.

the modified $(n-2f, n+1)$-RS scheme for re-initialization process to all the original nodes (Line 7).

4): Each original node N_s calculates and signs the evaluations of the new node N_{n+1} after receiving the re-initialize message (Line 19-20). Afterwards, the node N_s sends its signatures on the evaluation set $F_h(n+1)$ to the new node N_{n+1} (Line 21). The new node N_{n+1} aggregates the signatures after it has received $F+1$ signature sets from different nodes (Line 9-11). Meanwhile, if the node N_s receives the signature set from node N_{n+1}, it extracts the signatures of node N_{n+1} on its evaluations and then compresses it along with its previous aggregate signature into a single signature (Line 23-26).

If the verification is achieved, the new node N_{n+1} preserves the aggregate signature $\sigma_{h,n+1}$ together with its own evaluation set $F_h(n+1)$ and discards the computed evaluations of other nodes. Note that the re-initialization process can be launched in parallel if more than one new nodes join the system at the same time. The first new node who launch the re-initialization is responsible for the recovery and broadcast of the blocks, while the latter nodes merely calculate the evaluations of all the other nodes and sign the evaluations for verification.

Quit of Original Nodes. Since the model of this paper is based on the assumption of PBFT illustrated above, we just consider the case where the number of nodes $nrsquo$; still satisfies the assumption $n^{'} \geqslant 3f^{'}+1$ after an existing node quits.

In this case, the quit of the node has no influence on the schema of $(n-2f, n)$-RS and the remaining nodes. Since the $(n-2f, n)$-RS schema still works, the leaving node just needs to delete its preserved data and quits the system, and no additional behaviors on other nodes are required. The data availability can still be guaranteed.

3.6 Malicious Behavior Audit

Each node in PartitionChain has its own credit based on the malicious behaviors it have done, such as sending fake messages or keeping silence. To record the credits of the nodes in the system, each node maintains a dynamic array. Since the nodes can verify the correctness of the received evaluations from other nodes through CLAS, the misbehaviors can be detected. A node checks the correctness of the received evaluations, and if it detects the evaluation from another node N_s is forged, it broadcasts a message $< DETECTED, f_{h,j}(s), s>$, where $f_{h,j}(s)$ is the forged evaluation from node N_s and s is the index of the malicious node. When a node receives more than $f+1$ messages that tell N_s has behaved dishonestly, it reduces the credit of node N_S accordingly in the array.

Algorithm 3. Re-Initialization for Joining of a New Node

1: (Processing on new node N_{n+1})
2: broadcast decode message $< DECODE, h, n, n-2f >$ to the original nodes
3: **if** receive $n-2f$ evaluation sets **then**
4: $\mathcal{B} \leftarrow$ the set of recovered second-level-pieces of $B(h)$
5: $\mathcal{F} \leftarrow$ the evaluations of all the nodes
6: $\mathcal{H}_{n+1} \leftarrow$ the signatures on $\mathcal{F}$
7: broadcast re-initialize message $< RE\_INITIALIZE, \mathcal{B}, \mathcal{H}_{n+1}, h, n+1, n-2f >$
8: **end if**
9: **if** receive $F+1$ signature sets on $F_h(n+1)$ **then**
10: $signs \leftarrow$ the signatures on $\mathcal{F}$ from $F+1$ nodes
11: $\sigma_{h,n+1} \leftarrow$ Aggregate($signs$)
12: **end if**
13: (Processing on original node N_s)
14: **if** receive the decode message **then**
15: $msg \leftarrow < \mathcal{F}_h(s), \sigma_{h,s}, h, s >_{\sigma_s}$
16: SendMessage(N_{n+1}, msg)
17: **end if**
18: **if** receive the re-initialize message **then**
19: $F_h(n+1) \leftarrow$ the evaluation set of N_{n+1}
20: $\mathcal{H}_{n+1,s} \leftarrow$ the signatures on $F_h(n+1)$
21: SendMessage($N_{n+1}, \mathcal{H}_{n+1,s}$)
22: **end if**
23: **if** receive $\mathcal{H}_{n+1}$ from node N_{n+1} **then**
24: $signs \leftarrow$ signatures on evaluation set of N_s by N_{n+1}
25: $\sigma_{h,s} \leftarrow$ Aggregate($signs, \sigma_{h,s}$)
26: **end if**

When initializing, PartitionChain sets a lower bound of credits on account of the related system status. PartitionChain checks the credit of each node termly and expels the nodes with the credits below the bound. In order to relieve the overload of the system, the removal of dishonest nodes will only occur when the current number of nodes n' is greater than the n used in $(n-2f, n)$-RS schema so that the unessential updating operations can be avoided. Moreover, PartitionChain classifies the nodes into different security levels on the grounds of their credits, enabling clients to select more credible nodes to handle their requests.

Algorithm 4 presents the detailed process of periodic inspection of malicious nodes in the system. PartitionChain nominates a leader L to be responsible for this process. First, the leader L broadcasts an inspect message $< INSPECT, t >$ where t is a specific time point when the inspection process initiates (Line 1-2). When a node N_s receives the inspect message, it finds out the indexes of the nodes whose credits are lower than the lower bound and signs the found indexes (Line 10-15). Then node N_s

aggregates the signatures into a single signature for the purpose of relieving communication pressure and sends it together with the found indexes and the specific time point t to the leader (Line 16-18). After receiving the messages, the leader L extracts all the index lists from the different nodes (Line 3-4). Similar to the evaluations, the correctness of the indexes can also be verified through the corresponding aggregate signatures. If a node whose index is existing in more than $f+1$ index lists which means the consensus that this node is malicious is achieved, the leader will expel it from the system by function $Quit()$ (Line 5-7).

Algorithm 4. Inspection of Malicious Nodes

1: (Processing on leader L)
2: broadcast inspect message $< INSPECT, t >$
3: **if** receive n messages **then**
4: $index(1), \dots index(n) \leftarrow n$ index lists from different nodes
5: $malicious = m_1, m_2 \dots \leftarrow$ indexes of nodes whose indexes exist in more than $f+1$ lists
6: **for** m_i in $malicious$ **do**
7: Quit(N_{m_i})
8: **end for**
9: **end if**
10: (Processing on node N_s)
11: **if** receive inspect message **then**
12: $index = m_1, m_2, \dots \leftarrow$ indexes of nodes whose scores are lower than the lower bound
13: **for** m_i in $index$ **do**
14: $signs \leftarrow signs \cup$ Sign(m_i)
15: **end for**
16: $as \leftarrow$ Aggregate($signs$)
17: $msg \leftarrow < index, as, t >$
18: SendMessage(L, msg)
19: **end if**

4 Performance Analysis

4.1 Storage Consumption

In this part, we will prove that PartitionChain reduces the storage consumption per block to a constant complexity $O(1)$ with respect to the number of nodes n. As illustrated in Section 3, an original block $B(h)$ is first divided into r primary-pieces and then each primary-piece is divide again into $k = n - 2f$ second-level-pieces. Assume that the size of each block is T, and then the size of each primary-piece and second-level-piece are respectively equal to $\frac{T}{r}$ and $\frac{T}{k \cdot r}$. After encoding, every k second-level-pieces are mapped to n evaluations through the corresponding RS polynomial. Therefore, the storage consumption per block is

$$r \cdot n \cdot \frac{T}{k \cdot r} \approx \frac{3f \cdot T}{f} = 3T,$$

on the basis of the precondition of PBFT that $n = 3f + 1$. In comparison of the storage consumption $O(n)$ in full-replication strategy, PartitionChain greatly economizes the storage overheads while maintaining the same reliability.

4.2 Computational Efficiency

Now we will prove that the computational complexity of encoding and decoding are greatly reduced to $O(Tc \cdot n)$ and $O(Tc \cdot n^2)$ in regard to the number of nodes n respectively. Further, the computation consumption in coding is reduced by about 2^{18} times in PartitionChain compared with BFT-Store.

In $(n, n-2f)$-RS schema, the computational complexity in encoding and decoding can be expressed as

$$ERS(n) = O(n(n-2f)), \tag{1}$$

and

$$DRS(n) = O((n-2f)^3). \tag{2}$$

First, we compare the computational complexity in decoding between PartitionChain and BFT-Store. Assuming that the size of a block is T, the computational complexity of decoding a block in BFT-Store can be expressed as

$$O(T^2 \cdot DRS(n)) = O(T^2 \cdot n^3). \tag{3}$$

Note that we simplify $(n-2f)$ to n in the deductions based on the precondition of PBFT [22] that $f = \lfloor \frac{n-1}{3} \rfloor$.

In PartitionChain, we limit the size of each second-level-piece to $c = 2^{10}$ bits, and thus the number of primary-pieces is $r = \frac{T}{c(n-2f)}$, deduced detailedly in Section 3. To recover a single block, the leader needs to decode r times for the r primary-pieces in total; therefore, the computational complexity of decoding a block in PartitionChain is

$$O(r \cdot c^2 \cdot DRS(n)) = O\left(\frac{T}{cn} \cdot c^2 \cdot n^3\right) = O(Tc \cdot n^2). \tag{4}$$

Further, we can learn that the computational complexity of decoding in PartitionChain is decreased approximately by 2^{18} times than that in BFT-Store from Equation (5)

$$\frac{T^2 \cdot n^3}{Tc \cdot n^2} = \frac{Tn}{c} \approx 2^{18}, \tag{5}$$

where the size of an original block T is 2^{10} and n is under 100 in our model.

Similarly, the computational complexity of encoding a block is

$$O(T^2 \cdot ERS(n)) = O(T^2 \cdot n^2), \tag{6}$$

in BFT-Store, while the computational complexity of encoding in PartitionChain is

$$O(r \cdot (c^2 \cdot ERS(n)) = O\left(\frac{T}{cn} \cdot c^2 \cdot n^2\right) = O(Tc \cdot n), \tag{7}$$

which is accordingly decreased by about 2^{18} times compared to BFT-Store. Undoubtedly, PartitionChain reduces the response time and improves the overall throughput of the system significantly.

4.3 Improvements of Re-Initialization Process

In BFT-Store, each time the number of nodes n in the system varies, all the nodes are forced to participate in the re-initialization, decoding all the chunks and then re-encoding the recovered blocks with the updated RS schema. Namely, the re-initialization of BFT-Store requires $O(n)$ operations on each node.

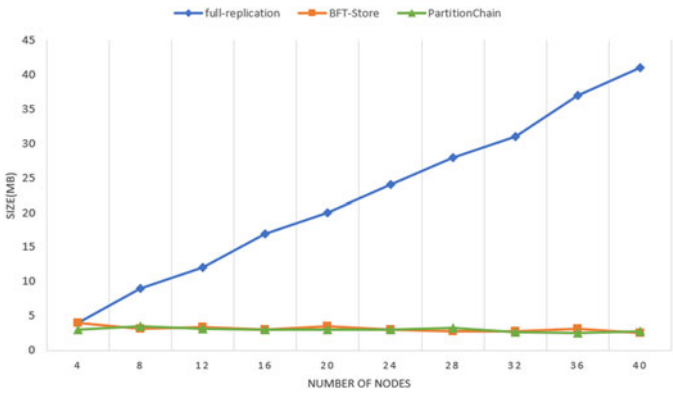

Fig. 7. Storage overhead per block against the number of nodes.

On the contrary, in PartitionChain, when a new node joins the system, the original nodes merely need to calculate the evaluation set of the new node for verification, and update their aggregate signature sets. That is, PartitionChain only requires $O(1)$ operations on the original node to accomplish the re-initialization. When a node quits the system, since $(n-2f,n)$-RS schema still works, no additional operation is required, and thus the computational overload on each node for quit of nodes is also reduced to $O(1)$.

During the re-encoding process, the leader is requested to deliver all the recovered second-level-pieces to the followers, which makes the system upload bandwidth a bottleneck. Thus, we employ Gossip Protocol [43] in PartitionChain for communication to relieve the communication pressure as well as to improve the network fault-tolerance. In Gossip network, each node can choose to relay the received messages to others randomly so that all the nodes can receive the broadcast message within time complexity $O(\log n)$.

4.4 Read Performance

Since the original blocks are partitioned and encoded into relative evaluations, each time a client requests for a block, the system has to launch the data recovery process. Consequently, the response time of a client requests to read a block will accordingly increase compared to the traditional full-replication strategy. In order to minimize the read performance loss, PartitionChain adopts caching technology for the several most frequently-requested blocks on a node. A Least Recently Used (LRU) cache [44] is proved to have good performance and relatively high hit-rate for Hot Data [45]. In PartitionChain, each node maintains a cache for several blocks most frequently-requested during the recent period. After receiving the recovered original block for the client, the node stores it in its cache. Thus, when a client request for these blocks again, the node can directly replies the target blocks to the client without data recovery since they have been locally-stored on the node.

5 Experimental Evaluation

In order to compare the performance between full-replication strategy, BFT-Store and PartitionChain more visibly, we follow the basic preconditions and setups of the experiments in [1], [35]. According to the principal goals of this paper, our experimental evaluations focus on the following three main aspects of the performance: storage consumption, computational efficiency and system re-initialization. Note that we select the case that $r=3$ the faulty factor of the system and the number of replicas of each block is equal to 1 as the BFT-Store sample in our experiment illustrated in [1].

TABLE 3
Comparison of Throughput of Encoding and Decoding Between PartitionChain and BFT-Store Against the Number of Nodes

Throughput(MB/sec) / Number of nodes	Encoding		Decoding	
	PartitionChain	BFT-Store	PartitionChain	BFT-Store
4	6.97×10^{12}	2.46×10^{7}	5.63×10^{12}	2.15×10^{7}
8	2.56×10^{12}	9.65×10^{6}	2.37×10^{12}	9.02×10^{6}
16	1.10×10^{12}	4.23×10^{6}	1.05×10^{12}	3.97×10^{6}
24	9.83×10^{11}	3.68×10^{6}	9.75×10^{11}	3.47×10^{6}
32	9.23×10^{11}	3.26×10^{6}	9.05×10^{11}	3.13×10^{6}
40	8.71×10^{11}	3.12×10^{6}	8.47×10^{11}	3.01×10^{6}

Storage Consumption. To evaluate the storage consumption of PartitionChain, we also employ full-replication strategy and BFT-Store for comparation. In this case, we set the size of each block fixed in 1MB and there is merely one block in the blockchain, while the number of nodes ranges from 4 to 40. For each node, we assume that each node has 200 MB storage space, meaning that each node can store 200 blocks at most.

We can learn from Fig. 7 that in full-replication strategy, the storage overhead per block is proportional to the number of nodes while the storage consumption in both BFT-Store and PartitionChain approximately remains steady. Specially, the storage consumption of BFT-Store and PartitionChain maintain roughly the same. The storage overhead per block when the number of nodes increases to 40 in PartitionChain is only one eighth of that in full-replication strategy. Therefore, it can be proved that PartitionChain successfully improves the system scalability in practical application.

Computational Efficiency. To evaluate the computational efficiency of PartitionChain, we observe the throughput of encoding and decoding. Note that coding is unnecessary in full-replication strategy, so we neglect it in this section. Now, we compare the throughput of encoding and decoding against the number of nodes between PartitionChain and BFT-Store.

As shown in Table 3, while the throughput of both PartitionChain and BFT-Store decreases with the number of nodes n ranging from 4 to 40, the throughput of PartitionChain remains approximately 2^{18} times of that in BFT-Store. The overall throughput of decoding and encoding in PartitionChain is about 10^{12} MB/sec; thus it can been seen that the computational efficiency is significantly improved in PartitionChain.

System Re-Initialization Process. In this case, we compare the average re-initialization time per node in BFT-Store and PartitionChain for joining of a new node as well as quit of an existing node($n' > n$). Note that full-replication strategy is neglected in this part because when the number of nodes varies, no additional operation for re-encoding is needed in this case.

It can be seen from Fig. 8 that the re-initialization time per node of BFT-Store increases as the number of nodes is growing, while the re-encoding time per node in PartitionChain for both cases hold steady against the number of nodes. The time consumption of re-initialization in PartitionChain for joining of a new node and quit of an original node are significantly reduced to 50 ms and 10 ms respectively, less than a third of that in BFT-Store.

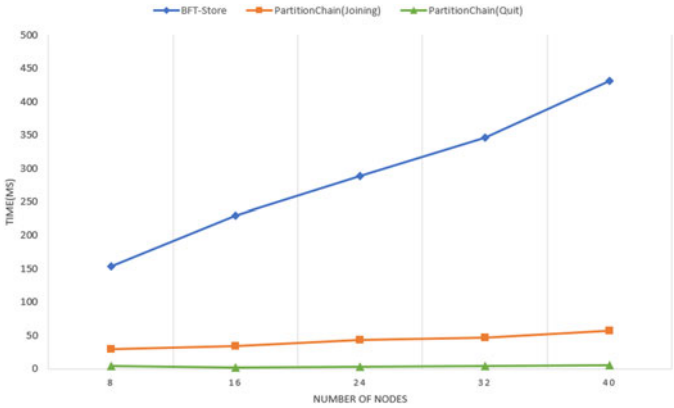

Fig. 8. Average re-initialization time per node when the number of nodes changes.

6 Conclusion

In order to relieve the heavy storage overloads of nodes in blockchain systems caused by full-replication strategies, we propose an optimal storage partition strategy termed PartitionChain. The overall contributions of this paper include: (i) reduce the storage consumption per block from $O(n)$ to $O(1)$, on the premise of maintaining data availability; (ii) relieve the computational overload of the re-initialization when the number of nodes varies, so that the original nodes are merely required to update signatures; (iii) reduce the computational complexity and communication overload in decoding and encoding; (iv) adopt CLAS instead of TS as the proof of data, achieving total decentralization without rely on any TTP; (v) introduce an audit mechanism for dishonest behaviors of nodes in system to purify the network. Furthermore, PartitionChain enables blockchain systems to suit dynamic network with higher efficiency and easier scalability.

References

[1] X. Qi, Z. Zhang, C. Jin, and A. Zhou, "BFT-store: Storage partition for permissioned blockchain via erasure coding," in *Proc. IEEE 36th Int. Conf. Data Eng.*, 2020, pp. 1926–1929.
[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf
[3] G. O. Karame, E. Androulaki, and S. Capkun, "Double-spending fast payments in Bitcoin," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2012, pp. 906–917.
[4] M. Nofer, P. Gomber, O. Hinz, and D. Schiereck, "Blockchain," *Bus. Inf. Syst. Eng.*, vol. 59, no. 3, pp. 183–187, 2017.
[5] S. S. Gupta, "Blockchain," *IBM*, 2017. [Online]. Available: http://www. IBM. COM
[6] N. Kshetri, "Can blockchain strengthen the Internet of Things?," *IT Professional*, vol. 19, no. 4, pp. 68–72, 2017.
[7] T. M. Fern ández-Caramés and P. Fraga-Lamas, "A review on the use of blockchain for the Internet of Things," *IEEE Access*, vol. 6, pp. 32979–33001, 2018.
[8] X. Liu, "Research and application of electronic invoice based on blockchain," in *Proc. MATEC Web Conf.*, 2018, Art. no. 04012.
[9] A. Deshpande, K. Stewart, L. Lepetit, and S. Gunashekar, "Distributed ledger technologies/blockchain: Challenges, opportunities and the prospects for standards," *Overview Rep. Brit. Standards Inst.*, vol. 40, p. 40, 2017.
[10] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *White Paper*, vol. 3, no. 37, 2014.
[11] H. Massias, X. S. Avila, and J.-J. Quisquater, "Design of a secure timestamping service with minimal trust requirement," in *Proc. 20th Symp. Inf. Theory Benelux*, 1999.
[12] R. C. Merkle, "Protocols for public key cryptosystems," in *Proc. IEEE Symp. Secur. Privacy*, 1980, pp. 122–122.
[13] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum Project Yellow Paper*, vol. 151, no. 2014, pp. 1–32, 2014.
[14] S. M. Pudukotai Dinakarrao, H. Sayadi, H. M. Makrani, C. Nowzari, S. Rafatirad, and H. Homayoun, "Lightweight node-level malware detection and network-level malware confinement in IoT networks," in *Proc. Des., Automat. Test Eur. Conf. Exhib.*, 2019, pp. 776–781.
[15] A. G. Dimakis, P. B. Godfrey, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," in *Proc. 26th IEEE Int. Conf. Comput. Commun.*, 2007, pp. 2000–2008.
[16] A. G. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A survey on network codes for distributed storage," *Proc. IEEE*, vol. 99, no. 3, pp. 476–489, Mar. 2011.
[17] M. Dai, S. Zhang, H. Wang, and S. Jin, "A low storage room requirement framework for distributed ledger in blockchain," *IEEE Access*, vol. 6, pp. 22970–22975, 2018.
[18] C. Decker and R. Wattenhofer, "A fast and scalable payment network with Bitcoin duplex micropayment channels," in *Proc. Symp. Self-Stabilizing Syst.*, 2015, pp. 3–18.
[19] C. Burchert, C. Decker, and R. Wattenhofer, "Scalable funding of bitcoin micropayment channel networks," *Roy. Soc. Open Sci.*, vol. 5, no. 8, 2018, Art. no. 180089.
[20] J. Tremback and Z. Hess, "Universal payment channels," 2015.
[21] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.
[22] M. Castro, B. Liskov *et al.*, "Practical byzantine fault tolerance," in *Proc. 3rd Symp. Oper. Syst. Des. Implementation*, 1999, pp. 173–186.
[23] R. Schollmeier, "A definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications," in *Proc. 1st Int. Conf. Peer-to-Peer Comput.*, 2001, pp. 101–102.
[24] R. Steinfeld, J. Pieprzyk, and H. Wang, "Lattice-based threshold changeability for standard shamir secret-sharing schemes," *IEEE Trans. Inf. Theory*, vol. 53, no. 7, pp. 2542–2559, Jul. 2007.
[25] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979.
[26] K. Wang, J. Mi, C. Xu, Q. Zhu, L. Shu, and D.-J. Deng, "Real-time load reduction in multimedia big data for mobile internet," *ACM Trans. Multimedia Comput. Commun. Appl*, vol. 12, no. 5, pp. 1–20, 2016.
[27] L. Zhang and F. Zhang, "A new certificateless aggregate signature scheme," *Comput. Commun.*, vol. 32, no. 6, pp. 1079–1085, 2009.
[28] E. Androulaki *et al.*, "Hyperledger fabric: A distributed operating system for permissioned blockchains," in *Proc. 13th EuroSys Conf.*, 2018, pp. 1–15.
[29] C. Cachin *et al.*, "Architecture of the hyperledger blockchain fabric," in *Proc. Workshop Distrib. Cryptocurrencies Consensus Ledgers*, 2016.
[30] S. B. Wicker and V. K. Bhargava, *Reed-Solomon Codes and their Applications*. Hobokken, NJ, USA: Wiley, 1999.
[31] V. Guruswami, A. Rudra, and M. Sudan, "Essential coding theory," 2012. [Online]. Available: http://www. cse. buffalo. edu/atri/courses/coding-theory/book
[32] T. Sasaki and H. Murao, "Efficient gaussian elimination method for symbolic determinants and linear systems," *ACM Trans. Math. Softw.*, vol. 8, no. 3, pp. 277–289, 1982.
[33] H. Sukhwani, J. M. Martínez, X. Chang, K. S. Trivedi, and A. Rindos, "Performance modeling of PBFT consensus process for permissioned blockchain network (hyperledger fabric)," in *Proc. IEEE 36th Symp. Reliable Distrib. Syst.*, 2017, pp. 253–255.
[34] R. V. Rasmussen and M. A. Trick, "Round robin scheduling–A survey," *Eur. J. Oper. Res.*, vol. 188, no. 3, pp. 617–636, 2008.
[35] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," Ph.D. Dissertation, University of Guelph, Guelph, ON, Canada, 2016.
[36] C.-P. Schnorr, "Efficient signature generation by smart cards," *J. Cryptology*, vol. 4, no. 3, pp. 161–174, 1991.
[37] A. Shamir, "Identity-based cryptosystems and signature schemes," in *Proc. Workshop Theory Appl. Cryptographic Techn.*, 1984, pp. 47–53.
[38] S. S. Al-Riyami and K. G. Paterson, "Certificateless public key cryptography," in *Proc. Int. Conf. Theory Appl. Cryptology Inf. Secur.*, 2003, pp. 452–473.
[39] D. Boneh *et al.*, "A survey of two signature aggregation techniques," *RSA Cryptobytes*, Citeseer, vol. 6, no. 2, pp. 1–10, 2003.
[40] V. Shoup, "Practical threshold signatures," in *Proc. Int. Conf. Theory Appl. Cryptographic Techn.*, 2000, pp. 207–220.
[41] D. Malkhi and M. Reiter, "Byzantine quorum systems," *Distrib. Comput.*, vol. 11, no. 4, pp. 203–213, 1998.

[42] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," in *Concurrency: The Works of Leslie Lamport*, San Rafael, CA, USA: Morgan & Claypool Publishers, 2019, pp. 203–226.
[43] J. Leitao, J. Pereira, and L. Rodrigues, "Hyparview: A membership protocol for reliable gossip-based broadcast," in *Proc. 37th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw*, 2007, pp. 419–429.
[44] H. Che, Y. Tung, and Z. Wang, "Hierarchical web caching systems: Modeling, design and experimental results," *IEEE J. Sel. Areas Commun*, vol. 20, no. 7, pp. 1305–1314, Sep. 2002.
[45] C. Fricker, P. Robert, and J. Roberts, "A versatile and accurate approximation for LRU cache performance," in *Proc. 24th Int. Teletraffic Congr.*, 2012, pp. 1–8.

Zhengyi Du is currently working toward the master's degree with Software Engineering Institute, East China Normal University, Shanghai, China. Her research interests include blockchain applications and information security, especially the data storage strategies and consensus protocols of the blockchain.

Xiongtao Pang is currently working toward the master's degree with Software Engineering Institute, East China Normal University, Shanghai, China. His research interests include blockchain applications and information security, especially the data storage strategies and consensus protocols of the blockchain.

Haifeng Qian received the bachelor's and master's degrees in algebraic geometry from the Department of Mathematics, East China Normal University, in 2000 and 2003, respectively, and the PhD degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University, in 2006. He is currently a professor with Software Engineering Institute, East China Normal University, Shanghai, China. His research interests include network security, cryptography, and algebraic geometry.