A Byzantine Fault Tolerant Storage for Permissioned Blockchain

Xiaodong Qi[1] Zhihao Chen[1] Zhao Zhang[1,2] Cheqing Jin[1,2] Aoying Zhou[1,2]
Haizhen Zhuo[3] Quanqing Xu[3]
[1]School of Data Science and Engineering, East China Normal University
[2]Shanghai Engineering Research Center of Big Data Management, East China Normal University
[3]Ant Group
{xdqi, chenzh}@stu.ecnu.edu.cn, {zhzhang, cqjin, ayzhou}@dase.ecnu.edu.cn
{haizhen.zhz, xuquanqing.xqq}@antgroup.com

ABSTRACT

The full-replication data storage mechanism, as commonly utilized in existing blockchains, suffers from poor scalability, since it requires every node to preserve a complete copy of the whole block data locally to tolerant potential Byzantine failures. In a hostile environment, the malicious node may discard or tamper data deliberately. Thus, existing distributed storage method, which partitions data into multiple parts and distributes them over all nodes, cannot suit for blockchains. This demonstration showcases BFT-Store, a novel distributed storage engine for blockchains to break full-replication by integrating erasure coding with Byzantine Fault Tolerance (BFT) consensus protocol. This demonstration will (i) allow audience members to see how BFT-Store partitions block data over all nodes to reduce the storage occupation of system, and (ii) allow audience members to see how BFT-Store recovers blocks under distributed scenario even with Byzantine failure.

CCS CONCEPTS

• **Computing methodologies → Distributed algorithms**.

KEYWORDS

blockhchain, storage partition, erasure coding, PBFT

ACM Reference Format:
Xiaodong Qi[1] Zhihao Chen[1] Zhao Zhang[1,2] Cheqing Jin[1,2] Aoying Zhou[1,2], Haizhen Zhuo[3] Quanqing Xu[3] . 2021. A Byzantine Fault Tolerant Storage for Permissioned Blockchain. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21), June 20–25, 2021, Virtual Event, China.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3448016.3452744

1 INTRODUCTION

Blockchain is a distributed append-only ledger maintained by all nodes in a hostile environment. In general, each node holds a complete copy of block data, and consensus protocol, such as PoW [8] in permissionless blockchain and BFT (Byzantine Fault Tolerance) [5] in permissioned blockchain, ensures the data consistency among nodes. Thus the overall storage consumption per block is $O(n)$ with n participants. However, with this full-replication manner, each node has to devote huge storage space to preserving blocks, which increases the barrier for participants. For example, the total amount of block data of Bitcoin is over 200 GB right now and the Ethereum network generates about 0.2 GB data every day [2]. This will be worse in a permssioned setting that leverages consensus with higher throughput. To address this issue, some techniques are introduced, such as light-weight client [8, 12] and lightning network [4]. The light-weight client merely saves block headers to verify block bodies received from full nodes. However, this technique just alleviates the overhead of client while the full node still follows the full-replication manner. The basic idea of lightning network is to reduce the amount of block data by gathering multiple micropayments between two accounts and rarely uploading the final balances to blockchain. Although this approach eases the stress of storage for each node, it does not subvert the full-replication mechanism fundamentally.

From the view of traditional distributed systems, a straightforward concern is whether we can partition the data of blockchains over all nodes. This approach divides the original data into a number of partitions and distributes them over all nodes. For high availability, each partition is replicated for several times (e.g., 3 in Hadoop [7]) to tolerate the failure of a single point. However, if all replicas of a block happen to be dispatched to malicious nodes who may keep silent or tamper data deliberately, this block may become lost forever. Alternatively, *erasure coding* (EC for short) provides significantly lower storage overhead than multi-replication at the same fault tolerance level [11]. In a nutshell, EC transforms the original blocks into a set of coded blocks, namely "chunks", such that any subset with sufficient available chunks can reconstruct the original data. Existing systems adopting EC just consider the omission node failures (e.g. fail-stop) where every node will not taint other nodes. In contrast, in a Byzantine environment [6], a node will reconstruct incorrect data via EC decoding based on the invalid chunks sent by malicious nodes.

This demonstration will showcase BFT-Store, a store engine, which partitions and distributes block data among network to reduce the storage consumption of system, for permissioned blockchain employing PBFT consensus protocol. BFT-Store combines the EC technique and PBFT protocol to break up the full-replication manner. Basically, BFT-Store encodes block data into n chunks via EC and distributes them to all nodes securely. Therefore, each node just needs to save a fragment of block data instead of all. Even if Byzantine nodes exist, BFT-Store promises all blocks are recoverable based on the EC decoding and chunk verification. The complete content

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SIGMOD '21, June 20–25, 2021, Virtual Event, China
© 2021 Association for Computing Machinery.
ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3452744

of BFT-Store was introduced in [9]. Conference attendees will be able to observe BFT-Store's automatic encoding and distribution of block data during running of the blockchain. Additionally, they will be able to observe how each node in BFT-Store deals with reading requests for blocks from clients in different cases, including the recovery of blocks when malicious node keeps silent or tampers data.

2 BACKGROUND AND SETTING

In this section, we briefly present the basic background and setting of BFT-Store, and the formal setting is described in [9].

PBFT protocol. BFT-Store employs the PBFT protocol for agreement of blocks and inherits the assumption of PBFT. There are at most f faulty nodes out of n participants such that $n \geq 3f+1$. Each node occupies a private and public key pair and knows all others' public keys in advance. PBFT is a three-phases protocol including *Pre-Prepare*, *Prepare* and *Commit*. In an ordinary case, the consensus will be reached by all honest nodes with two rounds of voting: *Prepare* and *Commit*. In *Pre-Prepare* phase, the leader proposes a block to all nodes. Subsequently, in *Prepare* phase, each node votes for the proposed block by broadcasting signed prepare-vote message, if the block is correct and complete. Upon receiving $n-f$ prepare-votes with valid signatures, each node broadcasts its commit-vote for the block again in *Commit* phase. Finally, a node commits the block once receiving more than $n-f$ commit-votes again.

Adversary model. The malicious node in BFT-Store can misbehavior in arbitrary way, such as keeping silent or tampering data. Thus, a faulty node can fail to respond to data requests from others or send back incorrect data.

Erasure coding. Erasure coding is a common technique to tolerate failures in storage systems, and the Reed-Solomon (RS) code [10] is the most common one. There are two configurable parameters in RS code, K and M. The RS divides the data into K equal-sized fragments and then encodes them to generate M redundant fragments called *parities*. The $K+M$ fragments are called chunks. The encoding of RS guarantees that any K out of $(K+M)$ chunks are sufficient to recover the original data. Emphasise that EC itself cannot check the correctness of chunks and it may recover incorrect data based on invalid chunks. Therefore, every node in blockchain needs to additionally validate the correctness of the received chunks.

Threshold signature. An (n,t)-threshold signature [3] on a message m is a single, constant-sized aggregate signature that passes verification if and only if at least t out of the n participants sign m. The verifier does not need to know the identities of t signers. The t signers can sign the message independently and then a prover can aggregate these t individual signatures into a single one.

3 BFT-STORE FUNCTIONALITY

Figure 1 presents the architecture of BFT-Store, a storage engine for blockchain with Byzantine fault tolerance on a node. Clients submit transactions (TXs) to the blockchain, which are packed in the form of blocks for consensus among nodes. Each block agreed by network are then written into BFT-Store for persistence. Clients read blocks by sending requests to nodes in BFT-Store. In essence, the functionality of BFT-Store can be split into two sub-functions: *block encoding* and *block reading*.

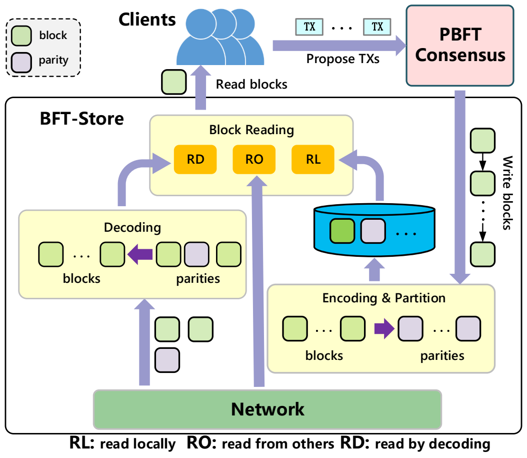
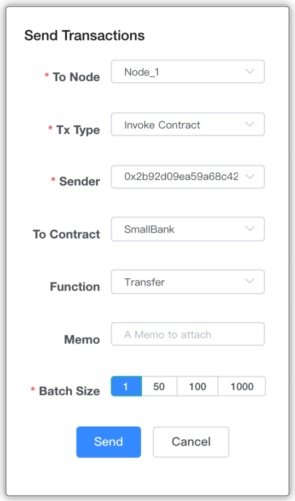


Figure 1: Architecture of BFT-Store on a node

Block encoding. BFT-Store employs a $(n-2f, 2f)$-RS to encode blocks. In particular, every $n-2f$ original blocks, denoted by $\{B_1, \cdots, B_{n-2f}\}$ are encoded to generate $2f$ parities. Then the original $n-2f$ blocks and $2f$ parities make up the n chunks, denoted by $\{C_1, \cdots, C_{n-2f}, \cdots, C_n\}$. We can gain $n-2f$ original blocks by decoding based on any $n-2f$ chunks of $\{C_1, \cdots, C_{n-2f}, \cdots, C_n\}$. These n chunks are distributed over all n nodes, where each node preserves a unique chunk. Thus, every node merely stores a small fragment instead of entire blockchain. In a hostile environment, there is no trusted third-party to encode blocks and distribute chunks. BFT-Store requires each node to encode blocks independently without any network communication. Note that the blocks $B_1 \sim B_{n-2f}$ are produced by consensus of blockchain, thus all honest nodes will obtain the same blocks. Every node maintains a *block buffer* to cache the newly generated blocks in a coding group before encoding. Once collecting every $n-2f$ blocks from consensus, each node encodes these blocks into n chunks, picks one out of them for local preservation, and deletes others for space efficiency.

BFT-Store exploits a deterministic method to ensure different nodes will different chunks [9]. This approach avoids the negotiation among nodes to determine the distribution of chunks. If a chunk C_j that is stored by node P_i, we say that P_i is the *target node* of chunk C_j. There are at least honest $n-2f$ nodes out of any $n-f$ nodes, a quorum of PBFT. Then the $n-2f$ chunks stored these $n-2f$ honest node are always available, which are sufficient to recover all original blocks by decoding.

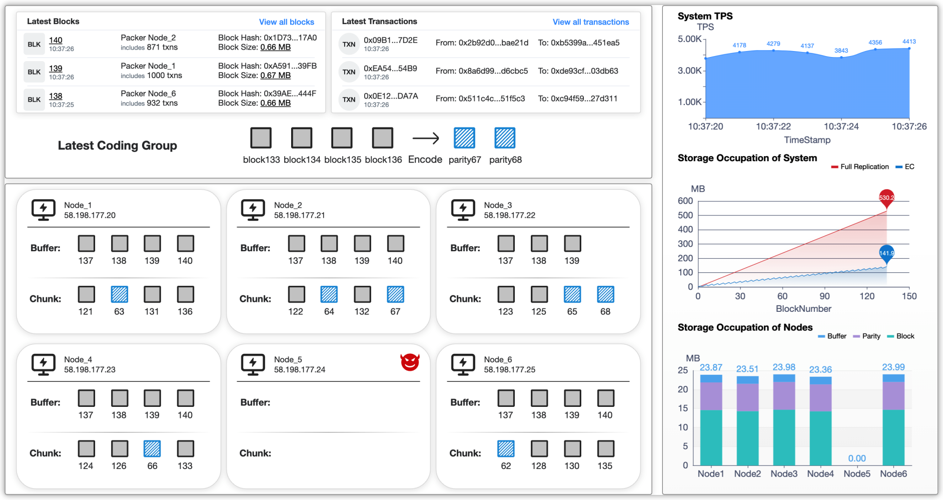
Block reading. To read a block B_i, a client sends its request to one node P, called the *delegated node*. In BFT-Store, node P faces different cases when reads block B_i. (i) If node P is the target node of block B_i, P can read it locally and sends B_i to the client directly, which is depicted as the **RL** manner in Figure 1. If block B_i resides in the block buffer of P, P can also handle the request locally. (ii) If P is not the target node of block B_i, it tries to pull B_i from its target node P'. An honest node P' will transmit B_i back to P as the **RO** manner in Figure 1. (iii) If the malicious target node P' responds P with nothing or incorrect block, P is forced to recover all blocks $B_1 \sim B_{n-2f}$ by RS decoding, as the **RD** manner in Figure 1. To achieve this, P broadcasts the request to all nodes and others send their chunks in the same coding group back. Upon collecting $n-2f$ correct chunks, P recovers $B_1 \sim B_{n-2f}$ via RS decoding and



Send Transactions

* To Node	Node_1
* Tx Type	Invoke Contract
* Sender	0x2b92d09ea59a68c42
To Contract	SmallBank
Function	Transfer
Memo	A Memo to attach
* Batch Size	[✓ 1] 50 100 1000

Figure 2: Transaction invoke.



Latest Blocks — View all blocks

BLK 140 10:37:26	Packer Node_2 includes 871 txns	Block Hash: 0x1D73...17A0 Block Size: 0.66 MB
BLK 139 10:37:26	Packer Node_1 includes 1000 txns	Block Hash: 0xA591...39FB Block Size: 0.67 MB
BLK 138 10:37:25	Packer Node_6 includes 932 txns	Block Hash: 0x39AE...444F Block Size: 0.66 MB

Latest Transactions — View all transactions

TXN 0x09B1...7D2E 10:37:26	From: 0x2b92d0...bae21d	To: 0xb5399a...451ea5
TXN 0xEA54...54B9 10:37:26	From: 0x8a6d99...d6cbc5	To: 0xde93cf...03db63
TXN 0x0E12...DA7A 10:37:26	From: 0x511c4c...51f5c3	To: 0xc94f59...27d311

Figure 3: Block encoding and distribution of BFT-Store over nodes.

picks B_i from them as the reply to client. As an optimization for case **RD**, if a node has already cached the block B_i in its buffer, it sends the block B_i instead of chunk to P directly, which prevents P from decoding.

Since the Byzantine node may tamper chunks, every node has to validate every chunk received from other nodes. Otherwise, the incorrect block may be recovered by RS decoding based on invalid chunks. BFT-Store adopts the threshold signature as the proof of chunks. Specifically, each node broadcasts its signatures for all chunks in a coding group after encoding, and then aggregates $n - 2f$ signatures for each chunk into a single one via $(n, n - 2f)$-ts when collects $n - 2f$ signatures. When requested, each node sends back the threshold signature together with the chunk. Thus, a node can ensure at least $n - 2f$ nodes have ever accepted this chunk by verifying the threshold signature. There is at least one honest node who agrees with the chunk, out of these $n - 2f$ nodes $(n - 2f - f = n - 3f \geq 1)$. Therefore, even a node has no knowledge about a chunk in advance, it can still check the correctness of that chunk.

4 DEMONSTRATION

Our demonstration is conducted on a cluster with 6 nodes, running the PBFT protocol [1] to achieve agreement on blocks. All nodes are equipped with 16 CPU cores with 2.8GHz, 96GB RAM, 512GB SSD disk space, and 1Gbps network bandwidth. Moreover, five nodes are honest and one is malicious, so that every four blocks are encoded into six chunks with two parities in a coding group. The demonstration is split into two scenarios. In the first, audiences can observe the process of encoding and partition for block data as new blocks are generated by consensus continuously. Furthermore, audiences can find the comparison of storage occupation of BFT-Store and full-replication manner. In the second, audiences will be able to query blocks as a client by sending a request to any node, and observe the progress of the query process for three cases.

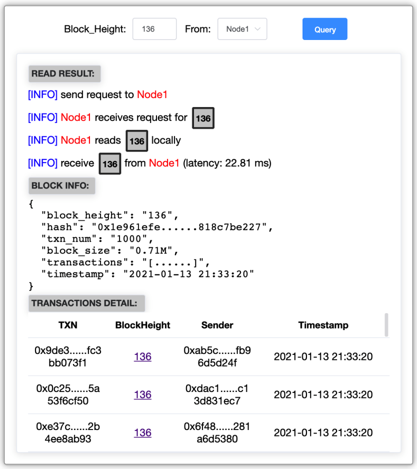
At the beginning of the demonstration, attendees invoke the transactions to start the process. As shown in Figure 2, the front-end will automatically sign these transactions and send them to the specified node for processing. Besides, users can invoke a number of transactions with the same arguments in a batch.

4.1 Scenario 1: Block Encoding & Chunk Distribution

Figure 3 shows the front-end of block encoding, and the back-end will synchronize the statistic data from 6 nodes periodically to refresh the interface of front-end.

Block encoding. The top left panel of Figure 3 depicts the information of the blocks generated by consensus recently, which contains the block height, block/transaction hashes, size, and creation time. When a node commits a new block, its information is updated in this panel. The bottom part of the panel shows the encoding of blocks, which reveals the relationship between blocks and parities in the latest coding group. In this screenshot, BFT-Store encodes the blocks 133 ~ 136 in a coding group with two parities 67 ~ 68. Then these six chunks are distributed over six nodes.

Chunk distribution. At the bottom left panel in Figure 3, there are six boxes, each of which represents a node. These six boxes present the distribution of chunks among six nodes. In each box, every square with black boundary is an original block and every square with blue shadow is a parity. Since every node performs the encoding independently, the node needs to cache the blocks generated recently, which are shown at the top of each box. Due to the delay of the network, different nodes may stay at different block height. At this point in the demonstration, nodes 1,2,4 and 6 are running at block height 140, thus blocks 137 ~ 140 reside in their buffers. Sequentially, these nodes will start a new coding group for blocks 137 ~ 140. Instead, node 3 just commits block 139 at the previous height, thus its buffer contains three blocks 137 ~ 139. The box labelled by a red devil of node 5 is blank, since it is a Byzantine node, which will not supply any information for presentation. After encoding, each node picks a unique chunk for preservation. In the latest coding group, nodes $\{4, 5, 6, 1\}$ store original blocks 133 ~ 136 respectively, while nodes $\{2, 3\}$ save parities 67 and 68. Since the original blocks are meaningful to users, every node storing the original block will gain more workload for reading in each group.



READ RESULT:

```
[INFO] send request to Node1
[INFO] Node1 receives request for 136
[INFO] Node1 reads 136 locally
[INFO] receive 136 from Node1 (latency: 22.81 ms)
```

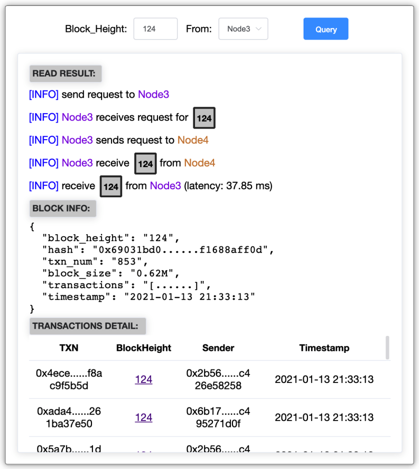
BLOCK INFO:

```
{
  "block_height": "136",
  "hash": "0x1e961efe......818c7be227",
  "txn_num": "1000",
  "block_size": "0.71M",
  "transactions": "[......]",
  "timestamp": "2021-01-13 21:33:20"
}
```

TRANSACTIONS DETAIL:

TXN	BlockHeight	Sender	Timestamp
0x9de3......fc3 bb073f1	136	0xab5c......fb9 6d5d24f	2021-01-13 21:33:20
0x0c25......5a 53f6cf50	136	0xdac1......c1 3d831ec7	2021-01-13 21:33:20
0xe37c......2b 4ee8ab93	136	0x6f48......281 a6d5380	2021-01-13 21:33:20

(a) RL: read locally



READ RESULT:

```
[INFO] send request to Node3
[INFO] Node3 receives request for 124
[INFO] Node3 sends request to Node4
[INFO] Node3 receive 124 from Node4
[INFO] receive 124 from Node3 (latency: 37.85 ms)
```

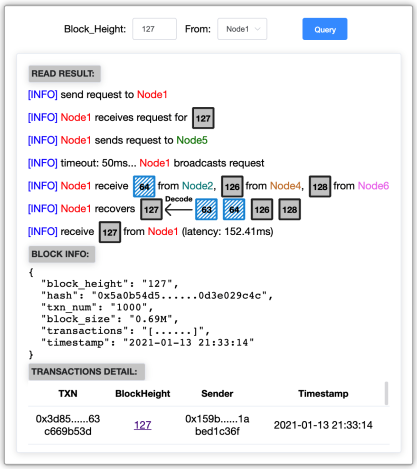
BLOCK INFO:

```
{
  "block_height": "124",
  "hash": "0x69031bd0......f1688aff0d",
  "txn_num": "853",
  "block_size": "0.62M",
  "transactions": "[......]",
  "timestamp": "2021-01-13 21:33:13"
}
```

TRANSACTIONS DETAIL:

TXN	BlockHeight	Sender	Timestamp
0x4ece......f8a c9f5b5d	124	0x2b56......c4 26e58258	2021-01-13 21:33:13
0xada4......26 1ba37e50	124	0x6b17......c4 95271d0f	2021-01-13 21:33:13
0x5a7b......1d	[illegible]	0x2b56......c4	[illegible]

(b) RO: read from others



READ RESULT:

```
[INFO] send request to Node1
[INFO] Node1 receives request for 127
[INFO] Node1 sends request to Node5
[INFO] timeout: 50ms... Node1 broadcasts request
[INFO] Node1 receive 64 from Node2, 126 from Node4, 128 from Node6
[INFO] Node1 recovers 127 ← 63 64 126 128
[INFO] receive 127 from Node1 (latency: 152.41ms)
```

BLOCK INFO:

```
{
  "block_height": "127",
  "hash": "0x5a0b54d5......0d3e029c4c",
  "txn_num": "1000",
  "block_size": "0.69M",
  "transactions": "[......]",
  "timestamp": "2021-01-13 21:33:14"
}
```

TRANSACTIONS DETAIL:

TXN	BlockHeight	Sender	Timestamp
0x3d85......63 c669b53d	127	0x159b......1a bed1c36f	2021-01-13 21:33:14

(c) RD: read by decoding

Figure 4: Block reading for three cases.

To balance the workload, BFT-Store assigns chunks to all nodes randomly. Although node 5 does not store block 134 correctly, BFT-Store is still able to recover it based on any four chunks on other nodes via decoding.

Storage consumption. The attendees can observe the throughput and storage consumption of the system in real time as shown in right panel of Figure 3. The table at the top part of the panel reveals the throughput of blockchain. The encoding of blocks has no significant impact on throughput since every node encodes block locally without any network communication. The table at the middle part of the panel presents the overall storage overhead of BFT-Store (including node 5) over time. As a comparison, the storage occupation of full-replication manner is drawn as well. The table shows the block height on the x-axis and storage occupation (MB) of the system on the y-axis. Since there are 6 nodes in this demonstration, the storage occupation of BFT-Store is reduced by a factor about $n - 2f = 6 - 2 \times 1 = 4$. The bottom side of this panel details the storage occupation of all nodes (excluding node 5), which consists of three parts: *buffer*, *block*, and *parity*. The storage consumption of each node is 1/6 of the entire system, since each node merely preserves a fragment of all data. Besides, the proportion of block data to total storage consumption is close on all nodes, meaning the workload is balanced across nodes.

4.2 Scenario 2: Block Reading

In this scenario, attendees can see the process of block reading for three cases as depicted in Figure 4. In the front-end, the user can type the block height of the target block to be requested and the delegated node. (i) In the **RL** case as shown in Figure 4(a), the delegated node 1 is the target node of block 136, thus it sends block 136 to front-end within a latency about 22.81ms. (ii) In the **RO** case as depicted in Figure 4(b), since the delegated node does not store block 124 locally, node 3 tries to pull it from target node 4. Then node 3 replies block 124 back with a latency 37.85ms. (iii) In the **RD** case, the target node of block 127 (node 5) is malicious as shown in Figure 4(c), thus node 1 cannot receive any data from it and needs to recover block 127 by decoding. To deploy decoding, node 1 broadcasts the request to all nodes. Then node 1 collects 4 chunks (parities 63 and 64, blocks 126 and 128) from nodes 1(itself), 2, 4, and 6 respectively. Node 1 performs the decoding based on the 4 chunks and picks out the block 127. The latency 152.41ms for this case is much greater than the best case, since node 1 needs to wait for a timeout 50ms before broadcasting. The probability that the target node stays malicious is $\frac{f}{n} = 1/6$. In a permissioned blockchain system, the number f of faulty node(s) is small, thus the average latency for reading is acceptable for clients. When n is great (i.e, ≥ 20), BFT-Store can assign a block to multiple (e.g. 3) nodes to reduce the probability of the third case at the cost of a bit storage space. With 3 target nodes for each block, the probability that the third case happens is bounded by $(\frac{f}{n})^{-3} < 3.7\%$. The evaluation of reading performance can be found in [9].

5 CONCLUSION

BFT-Store provides a secure partition method for block data of permissioned blockchain by combining PBFT protocol and erasure coding. Our demonstration allows the audience to explore BFT-Store's behaviors on all nodes during encoding and partition, emphasizing the reduction of storage consumption. Additionally, audience members can directly observe how block is read by node in three different cases and the read latency for various cases.

ACKNOWLEDGEMENT

This work was supported in part by National Science Foundation of China (61972152, U1911203 and U1811264) and ECNU Academic Innovation Promotion Program for Excellent Doctoral Students (YBNLTS2019-021). Cheqing Jin is the corresponding author.

REFERENCES

[1] 2020. PBFT. https://github.com/rleonardco/fabric-0.6.
[2] 2020. Statista. https://www.statista.com/.
[3] Dan Boneh, Craig Gentry, et al. 2003. A survey of two signature aggregation techniques. *RSA cryptobytes* 6, 2 (2003), 1–10.

[4] Conrad Burchert et al. 2018. Scalable funding of bitcoin micropayment channel networks. *Royal Society open science* (2018).
[5] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *OSDI*.
[6] Haibo Chen et al. 2017. Efficient and available in-memory KV-store with hybrid erasure coding and replication. *TOS* 13, 3 (2017), 25.
[7] Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. 2003. The Google file system. In *SOSP*.
[8] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. (2008).
[9] Xiaodong Qi, Zhao Zhang, Cheqing Jin, and Aoying Zhou. 2020. A Reliable Storage Partition for Permissioned Blockchain. *IEEE TKDE* 33, 1 (2020), 14–27.
[10] I. S. Reed and G. Solomon. 1960. Polynomial Codes Over Certain Finite Fields. *Journal of the Society for Industrial Applied Mathematics* (1960).
[11] Hakim Weatherspoon et al. 2002. Erasure coding vs. replication: A quantitative comparison. In *International Workshop on Peer-to-Peer Systems*. Springer.
[12] Gavin Wood et al. 2014. Ethereum: A secure decentralised generalised transaction ledger. *Ethereum project yellow paper* 151, 2014 (2014), 1–32.