BSS: An Efficient Block Storage System for Block Archive Service in Blockchains

Woochang Jeong
Computer Science and Engineering
POSTECH
Pohang, South Korea
wcjeong@postech.ac.kr

Chanik Park
Computer Science and Engineering
POSTECH
Pohang, South Korea
cipark@postech.ac.kr

***Abstract* — While blockchain enables decentralized services with zero trust, every node needs to store continuously generating block information in their local storage. Because blocks are generating continuously in blockchain, there exists a significant storage space burden on each node. A typical technique is to introduce a special node customized for block archiving. In case of Ethereum, a special node called block archive node to store all historical blocks. With the help of block archive node, each full node is enough to maintain only latest 128 blocks. In this paper, we propose BSS, an efficient block storage system for block archive service in blockchains, which can support any blockchain including Ethereum. BSS is a distributed system consisting of some number of block archive nodes. The storage overhead of BSS is O(1). The main idea is to encode each block into several small chunks and enforce each archive node to store one chunk, not an entire block. BSS also supports Byzantine tolerant chunk storage protocol for possible Byzantine attacks by block archive nodes. It is shown from the experiments on AWS, BSS enables both substantial block availability and notable performance even in Byzantine attacks.**

Keywords—Blockchain, Byzantine Fault Tolerance, storage

I. INTRODUCTION

Blockchain is a decentralized and distributed ledger technology that records transactions in a secure and immutable manner. Each new transaction is added to a block, which is then linked to the previous block, creating a chain of blocks. Storage overhead in blockchain refers to the additional data and resources required to maintain and store the blockchain's data beyond the actual transaction information. While blockchain enables decentralized services with zero trust, every node needs to store continously generating block information. Thus, given a nodes in a blockchain, the storage overhead is O(n). (n: the number of blockchain nodes). Storage overhead refers to the size of block data accumulated in blockchain nodes, and on a general blockchain, storage overhead increases linearly as the number of nodes increases.

For an example, Assuming that the size of each generated block is 1MiB, the number of tx included in one block is 1000, and transaction data occurs annually (24 hours / day * 365 days / year). Assume that Hyperledger Fabric is used to process transaction data for VISA, which processes large amounts of payment/payment transaction data. In the case of VISA, 140,839M txs were processed annually in 2020 [1], and when converted into TPS units, approximately 4500 TPS comes out. Accordingly, the blockchain node storage size per 4500 TPS is calculated at about 135.338 TiB per year, and it can be confirmed that the blockchain node storage size increases by about 676 TiB over 5 years. It can be confirmed that this is a large number when considering the storage capacity for actual data.

Efforts have been undertaken to mitigate the storage overhead associated with blockchains. Selective blockchain transaction pruning [2] and implementations such as Nano [3] and introduce pruning mechanisms, permitting nodes to discard outdated transaction data that is no longer necessary for validation purposes. This approach aids in diminishing storage prerequisites while upholding network security and integrity. Sharding, as demonstrated by methods like Elastico [4], Omniledger [5], and RapidChain [6], involves segmenting the blockchain into smaller sections or shards, each managed by distinct nodes. This division aids in distributing the storage load across nodes, subsequently enhancing scalability. Slimchain [7] and distributed off-chain storage of medical data [8] adopt off-chain or layer-2 solutions, to relocate specific transactions or data away from the primary blockchain, thereby alleviating storage and processing pressures on the core chain. The well-known Ethereum blockchain [10] employs a specialized node known as the block archive node, responsible for storing all historical blocks. Consequently, each full node is solely required to maintain the most recent 128 blocks.

The recent study in [9] proposed a general and robust block storage system based on a large-scale storage subsystem such as cloud storage. However, the service architecture of [9] has two main issues to be addressed. First, its block availability is limited due to its dependency on a large-scale storage subsystem such as cloud storage Second, its writing performance is constrained due to the application of the BFT consensus mechanism. Contemporary blockchains like Aptos [11] and Sui [12] exhibit remarkable performance, surpassing 100K TPS (transactions per second). So, it is imperative to enhance the write performance of a block archive service architecture.

In this paper, we propose BSS, an efficient block storage system for block archive service in Blockchains. BSS is characterized with high performance and low overhead of block storage. BSS consists of certain number of block archive nodes. The main idea is to encode each block into several small chunks and

enforce each archive node to store one chunk, not an entire block. BSS also supports Byzantine tolerant chunk storage protocol for possible Byzantine attacks by block archive nodes.

We make the following contributions:

- We propose BSS that guarantees block availability through S-node set. BSS does not depend on any particular storage service.
- To our understanding, we introduce the capability to audit block availability using the S-node set for the first time.
- By applying block encoding and byzantine reliable broadcast to the storage network, we support notable performance even in byzantine attacks.
- BSS supports Byzantine tolerant chunk storage protocol for possible Byzantine attacks by block archive nodes to achieve O(1) block storage overhead.

We make the assumption that malicious S-nodes are susceptible to Byzantine node failures. However, we also stipulate that these malicious S-nodes are unable to compromise cryptographic techniques. Within our framework, the communication model among S-nodes is based on the partially synchronous network.

II. BSS: An Efficient Block Storage System for Block Archive Service in Blockchains

BSS refers to a storage service that proceeds with consensus and storage of block data independently of the blockchain network. BSS takes a distinct approach by exclusively utilizing a storage network, devoid of any reliance on external cloud storage.

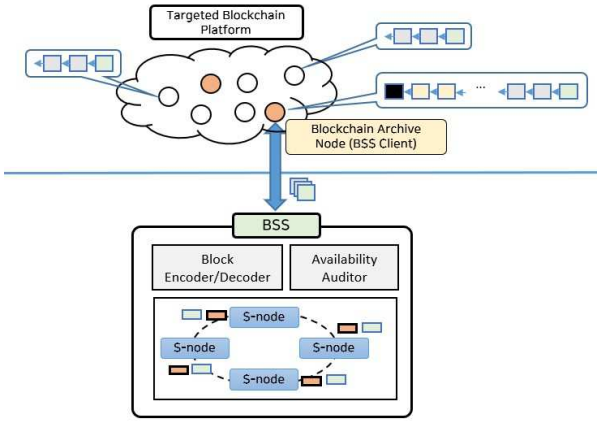

Fig. 1. The overall architecture of BSS. BSS undertakes dual roles. The first involves conducting audits to ensure block availability, achieved through regular auditing and reorganization processes. The second facet pertains to its capacity for byzantine-tolerant writing, enabling block writing while upholding block availability within the storage network, even in the presence of a byzantine S-node. This assurance of block availability is established by means of chunk exchanges facilitated by byzantine reliable broadcasting among S-nodes.

A. System Architecture

Within the framework of BSS, a fundamental element comprises a storage network comprising S-nodes. Through the conveyance of blocks across this storage network and the preservation of block metadata, it becomes achievable to establish the accessibility of these blocks. When employing BSS, this capability might materialize as a blockchain archive node tasked with upholding the entirety of block data, or alternatively, as a blockchain full node. The BSS client interacts with the BSS system through the utilization of three core APIs.

The first API facilitates the propagation of blocks. A block is dispatched to the storage network via the corresponding API. Upon delivery, the block undergoes erasure coding through a block encoder positioned within the storage network. Following this encoding, the block is disseminated throughout the storage network. Notably, the applied encoding follows the scheme (N, N-2f), enabling the reconstruction of a chunk even if solely N-2f fragments are available among the divided parts. This resilience stems from the necessity for N-f votes during the subsequent consensus phase to verify the availability of a specific block. Consequently, even if a maximum of f byzantine nodes exhibit malicious behavior during the recovery process, the block can be restored without complications. Subsequently, S-nodes on the storage network validate the assurance of block availability and store pertinent block metadata as corroborative evidence within individual local DAG structures.

The second aspect involves chunk sampling. The BSS client triggers the audit BA API on the storage network to assess the accessibility of the block it has conveyed. By referring to the block metadata established on the basis of the received block, the storage network verifies the availability of the requested block.

Lastly, there is the concept of block re-encoding. Instances might arise where BSS clients lose blocks or surpass the legally stipulated retention period. To address block recovery under such circumstances, the BSS client employs the block re-encoding API to interact with the storage network. Subsequently, the storage network supplies the BSS client with a block chunk corresponding to the designated block height. This provision is made possible by referencing the block metadata stored by each S-node (Fig. 2).

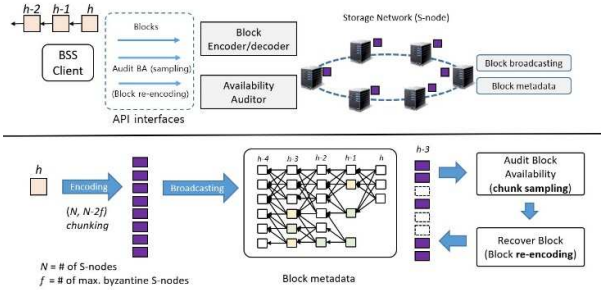

Fig. 2. An illustration of how BSS operates for a given block with height *h*

B. Chunk Sampling

Chunk sampling entails the process of segmenting blocks into chunks equivalent to the count of S-nodes within the storage network, subsequently dispersing and archiving them across the storage network. This technique facilitates the distribution and storage of chunks associated with blocks on the storage network, thereby enabling the audit of block availability.

The process commences when the BSS client initiates an audit inquiry regarding block availability within the storage network. Via the relevant request, the BSS client transmits the block associated with a particular height to the storage network. Subsequently, the conveyed block undergoes encoding through the block encoder module positioned within the storage network. During this encoding phase, the block is partitioned into N chunks using a (N, N-2f) chunking methodology. To validate the coherence between chunks and their respective blocks, the block encoder module constructs a merkle tree utilizing chunks as terminal nodes.

The generated block chunk, accompanied by pertinent chunk-related metadata, is dispatched to the S-nodes. This information encompasses the chunk's associated S-node ID, block height, merkle root, merkle path for the chunk, and the sender's signature. To guarantee the comprehensive propagation of each block chunk to its designated S-node, a dependable background process of chunk reliable broadcasting is executed.

Based on the broadcast block chunk, each S-node issues a block proposal based on the chunk in charge of each S-node to confirm that the block has been sufficiently propagated on the storage network. In the case of a block proposal, it includes the block hash in addition to the existing chunk information, and the certificates for the block proposal of the previous height.

When each S-node receives a block proposal from a specific S-node, it confirms that it has a chunk for the block and casts a vote for the proposal. In the case of Vote, the vote includes the chunk for the block, S-node id, block height, merkle path for the chunk you have, merkle root, block hash, and signature.

From the standpoint of each S-node, if n-f or more block votes for a specific block proposal are gathered, a block certificate is generated, and the DAG node composed of the block proposal and block certificate is stored in its local storage. Each S-node has block metadata as evidence proving that the block has been sufficiently distributed to the storage network. When metadata is available from more than n-f distinct nodes for a specific-height block, it ensures the distribution of over n-f chunks for that particular block throughout the storage network. Consequently, if n-f or a higher number of metadata instances are present for the block at the corresponding height, the commitment of the block pertaining to that specific height is confirmed.

In response to the request sent by the BSS client for block availability audit the other day, the S-node transmits metadata of the corresponding height to the BSS client. After the block metadata is finally created at the S-node level, the block data is removed because the block information is no longer needed.

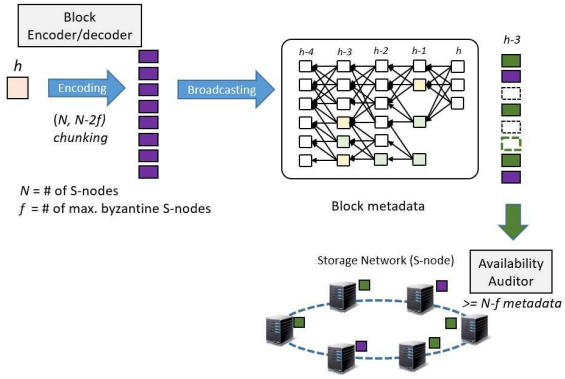

Fig. 3. Chunk Sampling. Dashed line-box denotes for missing block. The availability auditor verifies whether there exist over n-f metadata entries for the block at that specific height.

C. Block Re-encoding

Block re-encoding refers to the method of encoding N-divided block chunks back into the initial complete block structure. This technique involves soliciting block retrieval from Chain for blocks absent in the possession of the BSS client, followed by the re-encoding of the primary block using the block chunks distributed throughout the storage network.

Block Re-encoding serves as a method applicable when an ample quantity of block chunks (exceeding n-f) has been distributed across the storage network via chunk sampling. In scenarios where the distribution of more than n-f chunks has not yet taken place within the storage network, a response indicating that chunk sampling remains underway is conveyed back to the BSS client. In alignment with the chunk sampling procedure mentioned earlier, given the capacity of sincere S-nodes to assure the propagation of each chunk through reliable broadcasting across the storage network, the assurance of block re-encoding is ultimately established.

Assume that each S-node guarantees the availability of a block requested for recovery through block metadata. Then, each S-node can re-generate a block for that height based on n-f or more metadata for that block. The S-node completes the operation of block re-encoding by sending the re-generated block back to the BSS client.

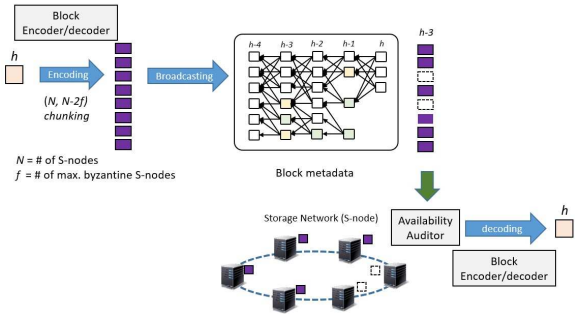

Fig. 4. Block Re-encoding. Block encoder/decoder recovers the missing blocks from the BSS client.

III. SECURITY ANALYSIS

For BSS, the following set of five security properties are upheld to fulfill the prerequisites set by established storage systems.

Theorem 1. (Validity) *If more than n-f S-nodes agreed on a specific block, the specific S-node submitted a block proposal corresponding to the block.*

Proof. As described above, the signature of sender is included in the block proposal. The meaning that more than n-f S-nodes agree on a specific block means that in the case of the block, all honest S-nodes verify the signature in the block proposal and vote for the block proposal.

Theorem 2. (Agreement) *If one honest S-node finally commits a block of a certain height, and another honest S-node also commits a block of that height, the block is the same.*

Proof. Each S-node votes after verifying that it contains a valid merkle root. Consider dividing the case into two commit conditions for the block. (1) If you have at least one certificate in the current round, certificates from different S-nodes in the same round all point to the same block. If it points to another block, each S-node does not vote. (2) Let r' be the round in which a certificate for at least one block proposal is obtained in the future round, when a certificate for at least one block proposal is obtained. n-f certificates of r'-1 belonging to r' certificate are fixed.

That is, the n-f certificates of r'-1 are the certificates of the previous round agreed upon by n-f or more S-nodes. In the same way, the certificates of r'-2, r'-3..., r mean that the block of each round has been agreed upon by all honest S-nodes. Eventually, the same block is committed for all previous rounds.

Theorem 3. (Termination) *Always each block has the same consensus outcome.*

Proof. We assume partially synchronous network. That is, messages exchanged between S-nodes and S-nodes arrive sometime after the global stabilization time (GST). Chunks for blocks must reach each S-node after GST, and all n-f honest S-nodes generate block proposals based on the received chunks. Since block proposals generated from each S-node reach all honest S-nodes, votes for the corresponding block proposal are guaranteed to be received from all honest S-nodes. Since the result of consensus on a block is determined by whether or not a certificate for the block is received, the result of consensus on each block must be derived at some point.

Theorem 4. (Availability) *If the BSS client calls the block re-encoding API, the S-node can finally reconstruct the block B_h.*

Proof. Committing the block B_h means that the certificate for the block proposals B_{hi}' corresponding to the corresponding block exists. That is, it means that more than n-f chunks for block B_h are distributed to the storage network. Even if up to f chunks of a malicious S-node are lost after voting, since the EC(n, n-2f) encoding scheme is applied, even if only n-f-f = n-2f chunks are collected, block B_h can be reconstructed.

Theorem 5. (Integrity) *All honest S-node S_i returns the same block B_h for block re-encoding request invoked by BSS client.*

Proof. In a situation where the height is fixed, all blocks of a specific height that are committed are the same. That is, with respect to the committed block Bh, block B_h can be reconstructed through chunks of S-nodes corresponding to votes included in block proposals B_{hi}'.

IV. EVALUATION

A. Experimental Setup

We performed a performance assessment using the AWS cloud [13] to address three primary research questions. (1) Does it deliver elevated throughput and minimal latency in typical scenarios devoid of Byzantine S-nodes? (2) Does it exhibit reduced storage overhead in contrast to scenarios where chunk sampling is omitted? (3) Does BSS maintain satisfactory throughput and latency levels even within a common setting involving Byzantine S-nodes?

For the AWS cloud instance type, the AWS cloud instance type for the BSS client and all S-nodes is fixed as m5d.xlarge (# of vCPUs: 4, RAM 16GiB, SSD: 1 x 150GB NVMe, Network Bandwidth: 10Gbps) proceeded. Deployment regions were divided into four regions: Seoul (ap-northeast-2), N.Virginia (us-east-1), N.California (us-west-1), and Sydney (ap-northeast-2). There is one BSS client, and the number of S-nodes has changed from 4 to 16. For the experimental workload, the tx size was 2,822 bytes, which was set according to the average hyperledger fabric average tx size, and the batch size was measured as 100.

We undertook a comprehensive comparative examination of three distinct systems. The primary reference system is the sui blockchain [12]. The second system encompasses BSS without the

incorporation of chunk sampling. In scenarios where chunk sampling is absent, entire blocks are exchanged directly between S-nodes. Contrasting the sui blockchain, this particular system deploys a linearized 2-phase consensus process to create a DAG node for a block suggested by a leader node. Subsequent to the commitment of the corresponding DAG node, a conclusive sequence is established for this DAG node concerning the corresponding DAG node linked to the block proposed by the previously endorsed leader node. Our ultimate emphasis in this performance evaluation centers around the BSS system integrated with chunk sampling.

B. Experiment

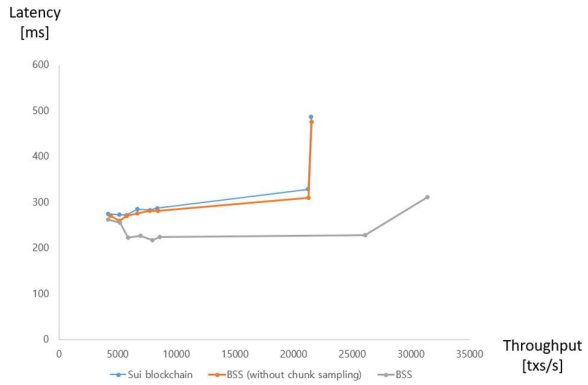

Fig. 5. BSS throughput vs. latency (S-nodes and BSS client are in Seoul. The number of S-nodes is 16.)

BSS supports the highest transaction volume. When chunk sampling is not implemented, it becomes evident that the saturation point aligns with the original sui blockchain's configuration. In a local area network (LAN) setup with minimal communication overhead, the system's performance likely showcases similarities in both throughput and latency.

Upon applying chunk sampling, a reduction in overall latency can be verified. With a decrease in the data size transferred from the client to the S-node, there appears to be a corresponding decrease in the overall latency.

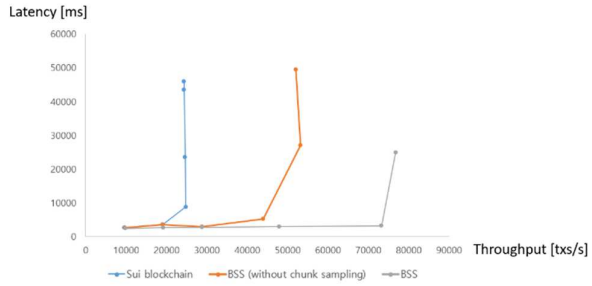

Fig. 6. BSS throughput vs. latency (BSS client is in Seoul. S-nodes are distributed equally through Seoul, N.Virginia, N.California, and Sydney.

Even within a wide area network (WAN) setting, it's evident that BSS supports the highest transaction volume. At a low sending rate, the impact on latency remains relatively minor. However, as the sending rate escalates, the influence of transmission time on data size becomes more pronounced, leading to observable differences in latency.

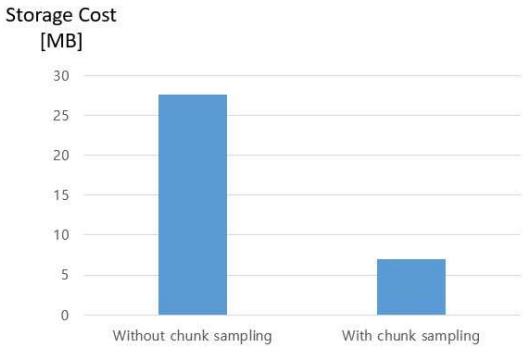

Fig. 7. Storage cost of BSS compared to BSS without chunk sampling

We conduct an experiment on block storage cost under LAN environment, where S-nodes and BSS client are in Seoul, the number of auditors is 4, and the send rate of transactions is 10,000 txs/s. Without chunk sampling, all block data are stored equally in every S-node and it leads to storage overhead in each S-node. However, when we apply chunk sampling to lessen the storage overhead of each S-node, block storage cost reduces by a quarter as each S-node gets a responsibility to each chunk of the block.

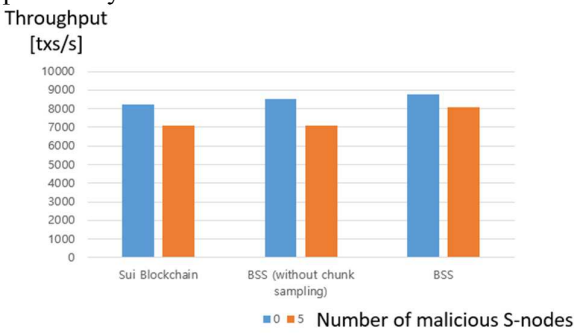

Fig. 8. BSS Throughput vs. number of malicious S-nodes

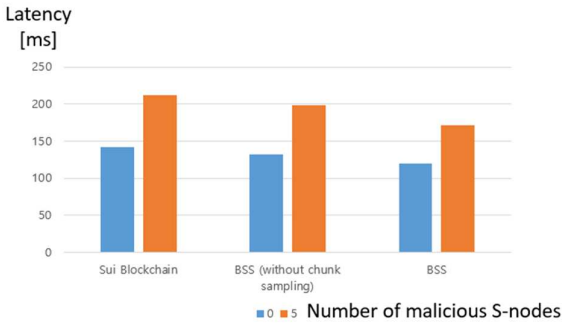

Fig. 9. BSS Latency vs. number of malicious S-nodes (BSS client and S-nodes are in Seoul. Transaction send rate is 10,000 txs/s. The number of S-nodes is 16.)

In order to assess resilience against Byzantine S-nodes, we vary the number of malicious S-nodes from 0 to 5. When both input and output remain unprocessed, the corresponding S-node is identified as displaying malicious behavior. With regards to BSS, introducing malicious S-nodes resulted in a decrease in throughput of approximately 6.8% when compared to the scenario absent of malicious nodes. Furthermore, for BSS, the presence of malicious S-nodes led to an increase in latency by approximately 52 ms compared to the scenario where no malicious S-nodes were involved. This latency increase can be attributed to the deliberate non-responsiveness of the malicious S-nodes to input and output. Remarkably, BSS continues to exhibit commendable performance even when subjected to Byzantine attacks.

V. CONCLUSION

We present BSS, an effective block archive system designed to accommodate any blockchain, including Ethereum. Upon distributing one S-node in each of the four regions, it was verified that BSS exhibited a latency of approximately 3 seconds and a throughput of roughly 73,000 txs/s. Furthermore, with all 16 S-nodes positioned in Seoul, even in the presence of five malicious S-nodes, a throughput of roughly 8,100 tx/s and a latency of around 0.2 seconds were attained when subjected to a send rate of 10,000 txs/s. Experimental results conducted on AWS demonstrate that BSS not only achieves substantial block availability but also maintains commendable performance levels, even when subjected to Byzantine attacks.

At present, our system design is predicated on the even distribution of S-nodes and BSS clients throughout the entire region. Consequently, we did not account for scenarios where nodes in specific regions are affected by geographical faults, and mechanisms for block recovery irrespective of the value of f were not included in our considerations. Developing workload distribution and recovery mechanisms that account for the geographical faults of both S-nodes and BSS clients remains a potential avenue for future exploration.

ACKNOWLEDGMENT

This work was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (2021-0-00136, Development of Big Blockchain Data Highly scalable Distributed Storage Technology for Increased Application.

REFERENCES

[1] Sousa, João & Bessani, Alysson & Vukolic, Marko. (2018). A Byzantine Fault-Tolerant Ordering Service for the Hyperledger Fabric Blockchain Platform. 51-58. 10.1109/DSN.2018.00018.

[2] E. Palm, O. Schelén and U. Bodin, "Selective Blockchain Transaction Pruning and State Derivability," 2018 Crypto Valley Conference on Blockchain Technology (CVCBT), Zug, Switzerland, 2018, pp. 31-40, doi: 10.1109/CVCBT.2018.00009.

[3] LeMahieu, Colin. "Nano: A feeless distributed cryptocurrency network." Nano [Online resource]. URL: https://nano. org/en/whitepaper (date of access: 24.03. 2018) 16 (2018): 17.

[4] Luu, Loi, et al. "A secure sharding protocol for open blockchains." Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. 2016..

[5] Kokoris-Kogias, Eleftherios, et al. "Omniledger: A secure, scale-out, decentralized ledger via sharding." 2018 IEEE symposium on security and privacy (SP). IEEE, 2018.

[6] Zamani, Mahdi, Mahnush Movahedi, and Mariana Raykova. "Rapidchain: Scaling blockchain via full sharding." Proceedings of the 2018 ACM SIGSAC conference on computer and communications security. 2018.

[7] Xu, Cheng, et al. "SlimChain: Scaling blockchain transactions through off-chain storage and parallel processing." Proceedings of the VLDB Endowment 14.11 (2021): 2314-2326.

[8] Kumar, Randhir, Ningrinla Marchang, and Rakesh Tripathi. "Distributed off-chain storage of patient diagnostic reports in healthcare system using IPFS and blockchain." 2020 International conference on communication systems & networks (COMSNETS). IEEE, 2020.

[9] Jeong, Woochang, and Chanik Park. "A General and Robust Blockchain Storage System based on External Storage Service." 2022 13th International Conference on Information and Communication Technology Convergence (ICTC). IEEE, 2022.

[10] Wood, Gavin. "Ethereum: A secure decentralised generalised transaction ledger." Ethereum project yellow paper 151.2014 (2014): 1-32.

[11] Brandon Williams et al. Aptos. https://github.com/aptos-labs/aptos-core, 2022.

[12] Spiegelman, Alexander, et al. "Bullshark: Dag bft protocols made practical." Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. 2022.

[13] AWS Cloud, aws.amazon.com