IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 29 March 2024, accepted 2 May 2024, date of publication 8 May 2024, date of current version 16 May 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3398787

RESEARCH ARTICLE

Pipelining and Overlapping: Techniques to Improve Both Throughput and Latency in BFT Consensus Blockchain

HANEUL OH AND CHANIK PARK
Department of Computer Science and Engineering, Pohang University of Science and Technology, Pohang 37673, South Korea

Corresponding author: Chanik Park (cipark@postech.ac.kr)

This research was funded by the Institute of Information Communications Technology Planning and Evaluation (IITP), through grants provided by the Ministry of Science and ICT (MSIT) of South Korea. The specific grants supported were for the projects titled "Development of Big Blockchain Data Highly Scalable Distributed Storage Technology for Increased Applications in Various Industries" and "Core Technologies for Hybrid P2P Network-Based Blockchain Services," under Grant Numbers 2021-0-00136 and 2021-0-00484, respectively.

ABSTRACT For decentralized characteristics of blockchain platforms, Byzantine Fault Tolerance (BFT) consensus protocols have been widely used to reach an agreement among nodes in the presence of malicious node attacks. However, due to their communication complexity, BFT consensus protocols are considered one major cause of the limited scalability of a blockchain platform. In this study, we propose two performance improvement techniques — Pipelining and Overlapping — to simplify the BFT consensus protocol without sacrificing decentralization. These techniques are designed to concurrently optimize throughput and minimize latency, irrespective of an increase or broad dispersion in consensus node deployment. We demonstrate the efficacy of these techniques through their application to AuditChain, a private blockchain leveraging a PBFT-like consensus mechanism with linear communication complexity. In a WAN environment utilizing AWS EC2 with 4 consensus nodes deployed in one region (ap-northeast-2), the system achieves about 1997 tx/s with a latency of 0.1 seconds. By further enhancing the distribution characteristics of our experimental environment, even with 32 consensus nodes distributed across two regions (ap-northeast-2, us-east-1), the system still achieves about 1995 tx/s with a latency of 0.3 seconds. These findings suggest that the proposed techniques are capable of significantly improving the performance metrics of BFT-based blockchain systems.

INDEX TERMS Blockchain, Byzantine fault tolerance, consensus protocol, performance.

I. INTRODUCTION

Blockchain [1] is a technology that uses a distributed database and is managed through a peer-to-peer (P2P) network. Unlike traditional databases that are stored in a central server, all nodes in a blockchain network maintain a copy of the ledger. Nodes in the network validate transactions through a consensus protocol, which groups transactions into blocks and builds hash chains. This process creates an ordered ledger that ensures consistency. Blockchain technology was first introduced with the creation of Bitcoin [2] and is widely used for its decentralization and security features.

The Byzantine Fault Tolerance (BFT) protocol is a fundamental concept in distributed system theory and is used in blockchain systems to ensure security. There are several BFT consensus protocols [4], [5], [6], [7], [8] that have been specifically designed for blockchain systems to handle random node failures and reach consensus among honest nodes even if Byzantine nodes exist in the system. The BFT protocol guarantees safety and liveness in the system, allowing for up to f ($< n/3$) faults when n nodes are participating in the system [9], [10], [11]. To ensure two essential properties, all n nodes must participate in the consensus process in the BFT consensus protocol, achieved through multiple voting rounds.

However, the BFT consensus protocol has performance scalability limitations [12]. These scalability limitations

The associate editor coordinating the review of this manuscript and approving it for publication was Dominik Strzalka.

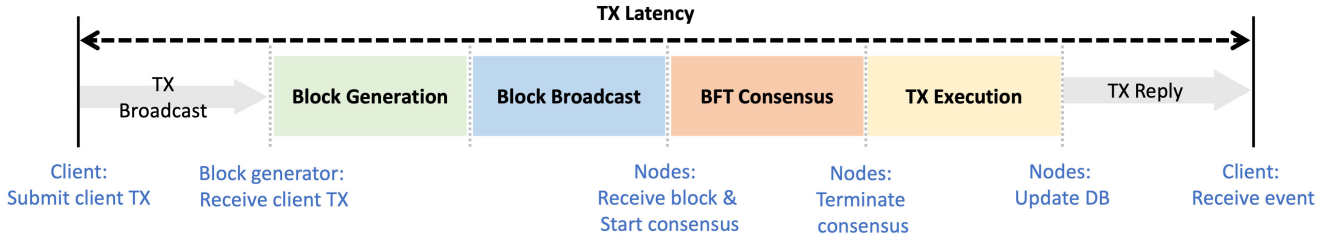

FIGURE 1. A diagram of processing a client transaction: each step affects the performance of a BFT consensus blockchain platform.

TABLE 1. Companions with other consensus.

	BFT steps	Pipelining			Overlapping
		Consensus instance pipelining	BFT steps pipelining	Max. pipelining depth	
PBFT [3]	2	x, v	x, v	no pipelining	x, v
Hotstuff [4]	3	x	o	4	x, v
Tendermint [7]	3	x	x, v	no pipelining	x, v
Narwhal/Tusk [8]	2	x, v	x, v	no pipelining	x, v
AuditChain [21]	2	x, v	x, v	no pipelining	x, v
AuditChain-PO	2	o	o	$\frac{bandwidth}{blocksize}$	o

(x: not adopt, o: adopt, v: can adopt)

result from the network and processing-level bottlenecks caused by the large number of messages that need to be sent, received, and processed to propagate blocks and reach consensus. For example, the typical BFT consensus algorithm, PBFT [3] uses a communication pattern with $O(n^2)$ message complexity. Most blockchain systems based on the BFT consensus protocol use a PBFT-like consensus protocol. The performance of a blockchain platform depends on the performance of the underlying BFT consensus protocol, with throughput and latency as key indicators. Throughput is the number of client transactions processed per second, while latency is the time between sending a client transaction and receiving a response. Processing a client transaction involves several steps (Figure 1), and the sequential process can lead to low throughput and high latency in a highly decentralized network with more nodes causing message and traffic increase and longer latencies between nodes that delay block receiving and quorum collection. There have been attempts to improve throughput [4], [5], [6], [7], [8], but the trade-off between performance and decentralization still exists.

We propose two performance improvement techniques - Pipelining and Overlapping - for the BFT consensus protocol in blockchain systems. These techniques aim to improve the performance of the BFT consensus protocol by increasing throughput and reducing latency while maintaining a high level of decentralization. The techniques are demonstrated on AuditChain [21], a blockchain platform that uses a simple BFT consensus algorithm and is deployed in the AWS cloud environment. We evaluate the effectiveness of these techniques through experiments on the AWS cloud, showing that the proposed techniques effectively improve both throughput and latency while maintaining a high degree of decentralization.

We made the following contributions to this paper:

1) Introduction of two performance improvement techniques — Pipelining and Overlapping — for the BFT consensus protocol which enhance throughput and reduce latency in a highly decentralized network.
2) Implementation of Pipelining and Overlapping in AuditChain, a blockchain system using a simple BFT consensus protocol, resulting in a prototype named AuditChain-PO.
3) Analysis and evaluation of the Pipelining and Overlapping techniques on the AWS cloud environment, using decentralization factors such as the number of nodes and geographical deployment of nodes. For an environment comprising 4 consensus nodes situated within a singular region, the AuditChain-PO prototype demonstrated a throughput of approximately 1997 transactions per second (tx/s) alongside a latency of 0.1 seconds. Remarkably, this performance level was nearly maintained in a more distributed setup involving 32 consensus nodes across two regions, with the system achieving roughly 1995 tx/s and a latency of 0.3 seconds. Our experiment results indicate the effectiveness of the two techniques in improving both throughput and latency.

II. RELATED WORKS

Background on BFT Consensus Protocols: Byzantine Fault Tolerant (BFT) consensus protocols are foundational to the

security and integrity of distributed systems, allowing them to withstand malicious or faulty nodes. Notwithstanding their robustness, protocols like PBFT [3] exhibit inherent challenges related to scalability, throughput, and latency, primarily due to their $O(n^2)$ message complexity.

Advances in Optimizing BFT Protocols: The landscape of BFT consensus protocols is marked by continuous innovation aimed at addressing the intrinsic challenges of scalability, throughput, and latency. Key developments in this area include:

- Reduction of Message Complexity: The SBFT protocol [5] employs a collector mechanism to reduce message complexity from $O(n^2)$ to $O(n)$, enhancing scalability by streamlining communication within the system. However, this centralization through collectors or leader nodes can lead to increased latency.
- Leader-Based Approaches: Protocols such as Hot-Stuff [4] and its derivatives streamline the consensus [19], [20] process by employing a leader node that collects votes, forms a Quorum Certification (QC), and distributes it among other nodes. This model optimizes message dissemination and consensus achievement. But it introduces an additional latency due to its two-step (round trip) communication process.
- Committee-Based Strategies: In the context of consensus mechanisms, Algorand's committee selection and Ouroboros's epochs and committee structures both serve to streamline message complexity by designating a subset of nodes for block proposal and validation [6], [14]. These methodologies, although optimizing efficiency, carry implications for the decentralization characteristic of blockchain networks.
- Gossip Protocols for Efficient Dissemination: Tendermint [7] capitalizes on gossip protocols to ensure reliable and widespread message dissemination across the network. However, this system necessitates predefined intervals to account for the longest expected time for peer-to-peer message propagation.
- Erasure Coding for Bandwidth Optimization: HoneyBadgerBFT [15], Poster [16], and DispersedLedger [17] leverage erasure coding to reduce message sizes and improve consensus efficiency, with the added benefit of enhanced fault tolerance. While this method streamlines the distribution of block information, it concurrently imposes computational burdens due to the complex encoding and decoding required.
- Decoupling Consensus and Propagation: The Narwhal/Tusk protocol [8] introduces an separation of block consensus from broadcasting, enabling it to potentially surpass 100,000 transactions per second (TPS). However, this separation can result in heightened latency within the network.
- Parallel Processing and Pipelining: The pipelining feature introduced by HotStuff allows for the overlap of different consensus phases, facilitating parallel processing of consensus steps and improving overall efficiency. Similarly, the Komorebi protocol [18] explores the synergy between sharding and BFT consensus, enabling parallel transaction processing and consensus across different network partitions, though managing cross-shard communication remains a challenge.

These advancements represent significant strides in the quest to optimize BFT protocols, each contributing unique solutions to the complex puzzle of enhancing distributed systems' performance while maintaining or balancing their foundational security and decentralization principles.

Integration of Blockchain and Database Technologies: The advent of blockchain-inspired databases like LedgerDB [22], BigChainDB [23], and Postchain [24] combines blockchain's auditability with traditional database efficiency, enhancing data storage and query execution while retaining the audit and verification strengths of blockchain. However, these hybrid systems might compromise on blockchain's hallmark features such as absolute decentralization and the high-level security provided by consensus mechanisms inherent in pure blockchain systems. This trade-off highlights the challenges in balancing efficiency with the uncompromised security and trustless nature characteristic of blockchain technology.

Within this evolving landscape, our work introduces Pipelining and Overlapping techniques, specifically designed to enhance the throughput and latency challenges endemic to BFT consensus protocols. By implementing these techniques in AuditChain [21], called AuditChain-PO, which utilizes a simplified PBFT consensus mechanism, we demonstrate substantial improvements in throughput and latency. AuditChain-PO consistently achieves a throughput of 1995 tx/s and a latency of 0.3 seconds, even in settings characterized by elevated decentralization aspects.

This paper's methodology enhances BFT protocol optimization within the constraints of predefined consensus architecture parameters. Our analysis, summarized in Table 1, examines existing research. Notably, while Hotstuff features pipelining in its BFT consensus steps, it is limited to a depth of four. In contrast, our proposed pipelining method expands on this, allowing for multiple consensus processes to be pipelined concurrently. Our techniques, pipelining and overlapping can be applied to various PBFT protocol derivatives, such as Tendermint, HotStuff, and inclusive of AuditChain.

III. PERFORMANCE IMPROVEMENT TECHNIQUES

To improve the performance of the BFT consensus protocol, we propose two techniques:

1) **Pipelining technique focusing on throughput:** By pipelining the consensus process on blocks, nodes allow to start the consensus process on blocks of the following height before the consensus on blocks of the previous height is complete.
2) **Overlapping technique focusing on latency:** By overlapping the block propagation and consensus process, nodes allow to first agree on a small digest of a block (the block's hash), and then each node receives

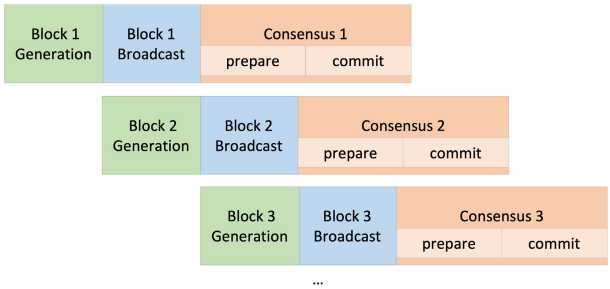

FIGURE 2. Pipelining to increase transaction throughput.

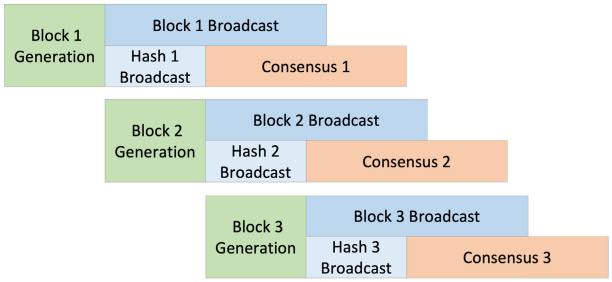

FIGURE 3. Overlapping to reduce transaction latency.

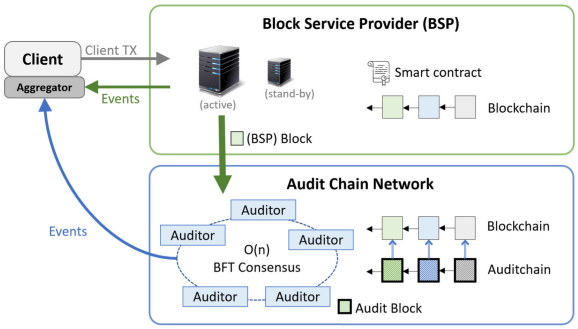

FIGURE 4. AuditChain architecture consists of three components: a client, a BSP, and auditors.

the block in the agreed order and executes a transaction to update its local state.

These techniques aim to improve both the throughput and latency of the BFT consensus protocol in a decentralized environment.

A. PIPELINING TECHNIQUE

The traditional BFT consensus protocols follow a sequential process, where one consensus process must be completed before the next one starts. To speed up the block consensus process, the pipelining technique can be used to pipeline multiple consensus processes. This technique is depicted in Figure 2. Two types of pipelines are considered in the proposed pipelining technique:

1) **Consensus instances pipeline:** Nodes can start the consensus process as soon as they receive a new block if they satisfy the hash chain property between blocks.
2) **BFT consensus steps pipeline:** The BFT consensus protocol includes several voting steps to handle Byzantine faults. These steps can be overlapped as they have the same message pattern and format.

When using the pipelining technique to overlap multiple consensus processes, the hash chain property of a blockchain can be leveraged to enable fast consensus decisions. This property ensures that each block extends the previous block by block hash pointers. With the hash chain property, a node can reach a consensus decision on a block with a higher height is complete. However, we must consider how to handle pipeline hazards that arise due to the interdependence of multiple consensus processes. If a failure occurs in one consensus process for a specific block, all subsequent consensus processes for blocks must be aborted and rolled back. To maintain this overhead within a certain range, it is important to control the maximum number of pipeline consensus processes.

B. OVERLAPPING TECHNIQUE

The overlapping technique aims to optimize this process by overlapping block propagation and consensus. This technique is shown in Figure 3 for the BFT consensus protocol. The block generator simultaneously sends out the hash of a block, which is much smaller in size compared to the entire block, and the block itself. The nodes participating in the consensus process first agree on the block hash before receiving the entire block. Once the block is received, they can then reach a consensus on it.

However, to complete the consensus protocol, all nodes must receive the entire block associated with the block hash agreed between them. It is called block availability. If a block proposer is malicious, that is, the block proposer propagates the block hash but not the block, all nodes cannot complete the consensus even though they agree on the block hash. Hence, we have to guarantee block availability when the overlapping technique is applied to a BFT consensus protocol to guarantee the liveness of the system.

To reduce consensus decision latency, the overlapping technique for the BFT consensus protocol enables concurrent processing of block propagation and consensus. The overlapping technique may be useful since it is enough to reach an agreement on the block hash and the size of the block hash is much smaller than the size of a block.

IV. SECURITY

Applying pipelining and overlapping techniques to the underlying BFT consensus protocol (ex. PBFT [3], AuditChain [21]) still follows the security assumptions of the underlying BFT consensus protocol. Pipelining and Overlapping techniques can cause failures, and the underlying BFT consensus protocol detects and handles those failures. Therefore, Pipelining and Overlapping techniques do not require any additional security assumptions from the underlying BFT consensus protocol.

A faulty node can create pipeline hazards. Fortunately, the underlying BFT consensus protocol takes into account faulty behaviors that cause pipeline hazards during the consensus process. For example, PBFT handles a primary fault where

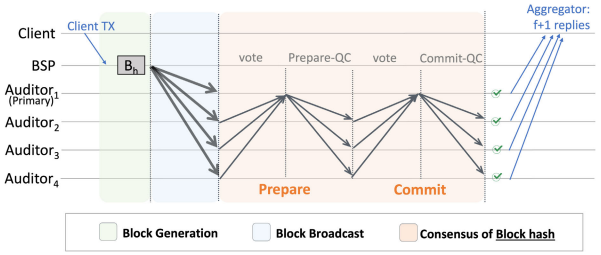

FIGURE 5. Message pattern of AuditChain in a single view.

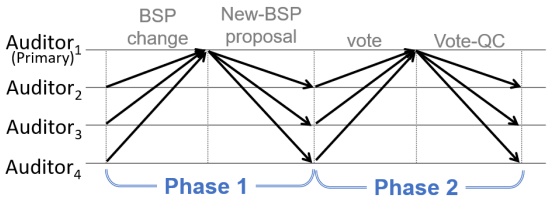

FIGURE 6. Message pattern of BSP change protocol.

the primary node does not disseminate a valid block. The underlying BFT consensus protocol is designed to detect and manage failures to meet safety and liveness of the system, even in the presence of up to N/3 failures. Accordingly, we respond to failures that lead to pipeline hazards by employing pre-established strategies for managing such failures.

Overlapping technology must assure block availability, ensuring that a node consistently receives a valid block corresponding to the received block hash. The underlying BFT consensus protocol already provides the block availability. For instance, if a node fails to receive a block a specified time window, the timer will expire, prompting the activation of the view change protocol. Consequently, faults that disrupt block availability can be effectively addressed using the fault handling mechanisms inherent to the underlying BFT consensus protocol, rendering the development of an entirely new fault handling protocol unnecessary.

V. PROTOTYPING ON AUDITCHAIN

A. OVERVIEW OF AUDITCHAIN

1) SYSTEM ARCHITECTURE

AuditChain [21] is a private blockchain system developed internally for performance evaluation, assuming a partially synchronous network. While the traditional PBFT [3] consensus algorithm is monolithic, block generation steps are separated from the consensus layer in AuditChain. As shown in Figure 4, AuditChain has two types of nodes: A Block Service Provider (BSP) and consensus nodes called auditors. BSP is responsible for creating blocks, while auditors forming a consensus network are responsible for agreement on blocks received from the BSP. The consensus algorithm shown in Figure 5 is like PBFT but with linear $O(n)$ communication complexity. The difference with PBFT is that it is linearized and has an additional node, BSP, as a static block producer.

2) CONSENSUS PROTOCOL

In Figure 5, the consensus protocol proceeds to a series of views ($v = 0, 1, 2, \ldots$) and a series of rounds ($h = 0, 1, 2, \ldots$) within one view. Each view has one active BSP responsible for generating and propagating blocks. One round proceeds with consensus on one block. In each round, the auditor initiates the consensus by creating a consensus instance CI_h corresponding to a block B_h of height h, and there is one primary auditor (referred to as a primary) that collects and propagates messages. Each round of AuditChain proceeds as follows.

① **Propose:** The BSP proposes a block B_h, It broadcasts the proposal message with block B_h to the audit network.

② **Prepare:** On receiving a proposal message containing a valid block B_h in a view v, an auditor sends an $AuditTX_{prepare}$ for B_h to the primary if B_h is valid to vote. The primary collects at least $(n - f)$ $AuditTX_{prepare}$ to form a $prepareQC_h$ in view v and round h. The primary broadcasts the $prepareQC_h$ to all auditors.

③ **Commit:** On receiving a $prepareQC_h$ for B_h, an auditor sends an $AuditTX_{commit}$ for B_h to the primary if the $prepareQC_h$ is valid to vote. The primary collects at least $(n - f)$ $AuditTX_{commit}$ to form a $commitQC_h$ in view v and round h. The primary broadcasts the $commitQC_h$ to all auditors.

④ **Reply:** On receiving a $commitQC_h$ from the primary, an auditor commits a block B_h, executes the client transaction in B_h, and sends a reply to the client. Finally, the auditor creates an audit block of the block B_h, containing $prepareQC_h$ and $commitQC_h$ as proof of the agreement for B_h.

3) BYZANTINE FAILURE HANDLING

In AuditChain, Byzantine nodes can exist: (active) BSP and f auditors. Among these, Byzantine BSP and Byzantine primary auditors can break safety and liveness. Therefore, the auditors monitor the presence of the behavior of the Byzantine BSP and the Byzantine primary auditor during the consensus process. When the auditor detects Byzantine BSP or Byzantine primary auditor, it switches to a new non-byzantine BSP or non-byzantine primary auditor.

The Byzantine BSP may not propagate blocks at all that propagate different blocks to the auditor so that the agreement does not proceed. In this case, the primary cannot generate a QC ($prepareQC_h$ or $commitQC_h$). Auditors that have not received QC from the primary cannot distinguish whether the primary is byzantine or BSP is byzantine. (It may be due to an unstable network, not a fault of a node. However, since it is difficult to distinguish the two, they are all considered to be faulty in node.) Therefore, the auditor firstly handles the situation that does not receive QC as primary auditor failure. In addition, when a situation in which QC is not received even on the (f+1)-th comes, the auditor processes it as BSP failure.

The auditor determines that the BSP is Byzantine when a block is not received or when a QC is not received for (f+1)-th

time and changes the BSP and starts a new view by executing the BSP change protocol. The BSP change protocol ensures that all host auditors eventually enter the same view. Figure 6 shows the message pattern of the BSP change protocol.

① **Phase 1:** An auditor sends a BSP change message m^i_{v+1} to its Primary. BSP change message includes the height h of the highest prepared block and a $prepareQC_h$ as the latest state. The primary determines the highest prepared block height among the received BSP change messages upon collecting at least $(n-f)$ m^i_{v+1}. After then, the primary broadcasts a new-BSP proposal message β_{v+1} prosing the new view $v+1$ to all auditors.

② **Phase 2:** When the auditor receives a valid β_{v+1} from the primary, the auditor sends a Vote message λ^i_v to the primary for the β_{v+1}. Upon receiving the quorum of vote messages, the primary generates a vote QC, which is a QC for the vote message, and propagates the vote QC to the rest of the auditors. The auditor that receives a valid vote QC from the primary starts the corresponding view.

The Byzantine primary in the consensus protocol does not broadcast a QC ($prepareQC_h$ or $commitQC_h$) or broadcasts an invalid QC. If the auditor does not receive a valid QC that must be received from the primary, the auditor determines that the primary is Byzantine and changes it to the primary of the next round and sends audit TX of prepare phase. The primary change in BSP change protocol is the same as the primary change in consensus protocol. In the BSP change protocol, the Byzantine primary does not send a new-BSP proposal or a vote QC, or an invalid new-BSP proposal or an invalid vote QC. If the auditor does not receive a valid new-BSP proposal or a valid vote QC, the auditor determines that the primary is Byzantine and changes to the primary of the next round to send a BSP change message.

B. PIPELINING ON AUDITCHAIN

1) CONSENSUS INSTANCES PIPELINE

In basic AuditChain, an auditor starts a consensus process for a block B_h as soon as it receives the block, even if the previous consensus instance for CI_{h-1} has not yet ended. And the auditor decides to commit a block by gathering two quorums ($prepareQC_h$ and $commitQC_h$) for each block. When pipelining consensus instances, auditors can start the consensus process as soon as a valid chained block is instances for chained blocks, the QC for the higher-height block can commit the lower-height block. This is because the QC for B_h implies that the quorum number of auditors have agreed not only on the blocks of height h but also on all blocks of height h or less that are linked by hash pointers.

2) BFT CONSENSUS STEPS PIPELINE

The basic AuditChain consensus process consists of two steps: Prepare and Commit. To optimize the process, both steps can be pipelined, specifically, the Commit step of consensus instance CI_h and the Prepare step of consensus instance CI_{h+1} can be performed simultaneously. By doing this, an auditor can skip the Commit step of CI_h and determine its commit through the $prepareQC_{h+1}$ in the Prepare step of CI_{h+1}. However, this technique is efficient only if the Prepare step of CI_{h+1} starts before the Prepare step of CI_h ends. If the Prepare step of CI_{h+1} starts later, the consensus process of CI_h may be delayed. To avoid this, the auditor only skips the Commit phase when there is a Prepare step of CI_{h+1} that can pipeline with the Commit step of CI_h. Otherwise, the auditor originally proceeds with the Commit step.

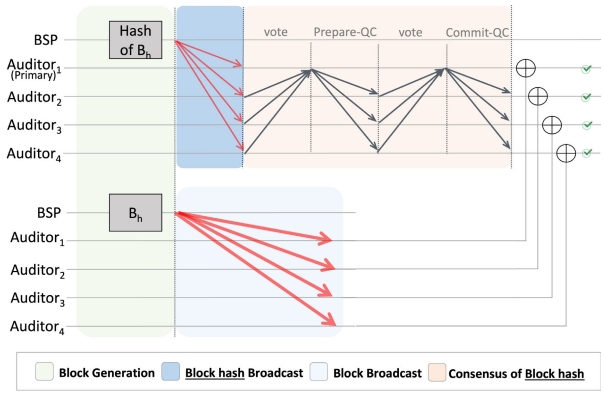

FIGURE 7. Message pattern of AuditChain adopting the overlapping technique.

3) PIPELINE HAZARD

If the BSP or primary is Byzantine, pipeline hazards can occur. Section IV describes how AuditChain detects and handles these failures. After detecting and handling the failure, AuditChain aborts the consensus process and rollbacks from the block height of the failed consensus process to the maximum block height of the pipelined consensus process. To ensure that this overhead remains within acceptable bounds, it is important to control the maximum number of pipeline consensus processes.

C. OVERLAPPING ON AUDITCHAIN

1) OVERLAPPING BLOCK BROADCAST AND CONSENSUS

The basic AuditChain protocol accelerates the consensus process by using an overlapping technique that allows overlapping block broadcasts and consensus. Figure 6 shows the AuditChain message pattern that adopts the overlapping technique. When a BSP finishes generating a block, the BSP broadcast a block proposal message within a block B_h along with a hash of B_h, to the network of auditors. The auditors start the consensus process on a hash of B_h immediately upon receiving the hash, which is smaller in size and quicker to receive than the entire block. However, the auditor cannot commit the block B_h with the QC gathered during the consensus process until they receive the block B_h and verify its validity. Only then can they use the QC to decide whether to commit the block following the commit rule.

2) BLOCK AVAILABILITY

To ensure the liveness of the system in BFT consensus protocols that use the overlapping technique, block availability

TABLE 2. Definitions of sections.

Section	Definition
Block Generation	The interval between the time when the BSP receives a transaction from the client and the time when it creates a block that includes the transaction.
Block (or Hash) Broadcast	The interval between the time when the BSP broadcast the block (or Hash) and the time when the auditor receives it (average among all auditors).
Consensus	The interval between the time when the auditor begins the consensus process and the time when a decision is made to commit to a transaction.
Consensus Delay	The interval between the time when an auditor receives a block (or Hash) and the auditor initiates the consensus process of the block (or Hash).

must be guaranteed. In AuditChain, this is achieved through the handling of BSP failures. Even if the BSP acts maliciously by not broadcasting a valid block or broadcasting an invalid one, block availability is ensured by detecting and handling the malicious BSP. If an auditor does not receive a block within a specified time frame or detects an invalid block, the auditor considers the BSP to be Byzantine and triggers the BSP change protocol to replace it with a new one.

VI. LIMITATIONS

This section describes the scope to which pipelining and overlapping techniques can be employed. The proposed techniques are generally applicable to any variations of PBFT [3] consensus protocols, e.g., Tendermint [7] or HotStuff [4], and including AuditChain [21].

The pipelining technique can be categorized into two primary types: consensus instance pipelining and BFT step pipelining. To begin with, consensus instance pipelining is suitable in cases where the consensus structure operates independently of block generation and the outcomes of consensus. This is because the consensus process is pipelined without reliance on the results from the previous-height block's consensus process. BFT consensus protocols compatible with consensus instance pipelining include PBFT, AuditChain, and Nawal [8]. Conversely, this technique is not applicable to Hotstuff and Tendermint due to their blocks incorporating the outcome of the previous-height block's consensus. On the other hand, BFT step pipelining is suitable when the consensus structure contains multiple steps for acquiring quorum, as is the case with PBFT.

The concept of overlapping becomes feasible within consensus structures characterized by a single block generator node responsible for both block creation and propagation. This encompasses consensus protocols like PBFT, AuditChain, Hotstuff, and Tendermint. Moreover, it is also applicable in consensus structures that involve multiple block generator nodes, such as in the case of Nawal.

VII. PERFORMANCE EVALUATION

We evaluate the effectiveness of pipelining and overlapping techniques on AuditChain [21] deployed in AWS environments. We compare the performance (throughput and latency) of the basic AuditChain and the performance improved AuditChain.

A. EXPERIMENTAL SETUPS

Implementation. We implement the prototype of the optimized AuditChain using Go programming language. We test with three variants: AuditChain (or "none") is the basic AuditChain without any techniques. AuditChain-P (or "P") adopts only the pipelining technique, and AuditChain-PO (or "P+O") adopts the pipelining and overlapping techniques. Workload. Performance (throughput, latency) is measured for 1 minute after the first 30 seconds after the test starts. We use the Hyperledger Caliper benchmarking tool to test the performance. In Every test, for a total of 2 minutes, the client submits a SendPayment transaction to BSP. When referring to latency, we mean the time elapsed from the client submits the transaction to the client receives the f+1 commit events from auditors. Deployment. We run our evaluation on AWS EC2. In our experiments, the nodes for the BSP and auditors are hosted by an EC2 c5d.4xlarge instance with 16 CPU cores, 16 GB of RAM, 400 GB of NVMe SSD, and a 10 Gbps NIC. The nodes for clients are hosted by an EC2 c5d.12xlarge instance with 48 CPU cores, 96 GB of RAM, 900 GB of NVMe SSD, and an 18 Gbps NIC. The nodes form a fully connected graph. We run our experiments on two scenarios: a local-distributed scenario, where every node is deployed in the Seoul region, and a global-distributed scenario, where auditor nodes are distributed across two AWS regions: Seoul (ap-northeast-2), N.Virginia (us-east-1). Profiling. To analyze the latency of experiments, we profile the time for each section. We divide 4 sections as shown Table 2. Consensus delay is an overhead for consensus because the BSP generates blocks quickly regardless of the consensus result, but consensus proceeds sequentially in the order of block height.

B. EFFECT OF PERFORMANCE IMPROVEMENT TECHNIQUES

We first experiment with local and global distributions by varying the batch size to analyze the effectiveness of each performance improvement technique. We conduct experiments where we used batch size = [100, 500, 1000]. A send rate per client of 1100 was used in local deployment,

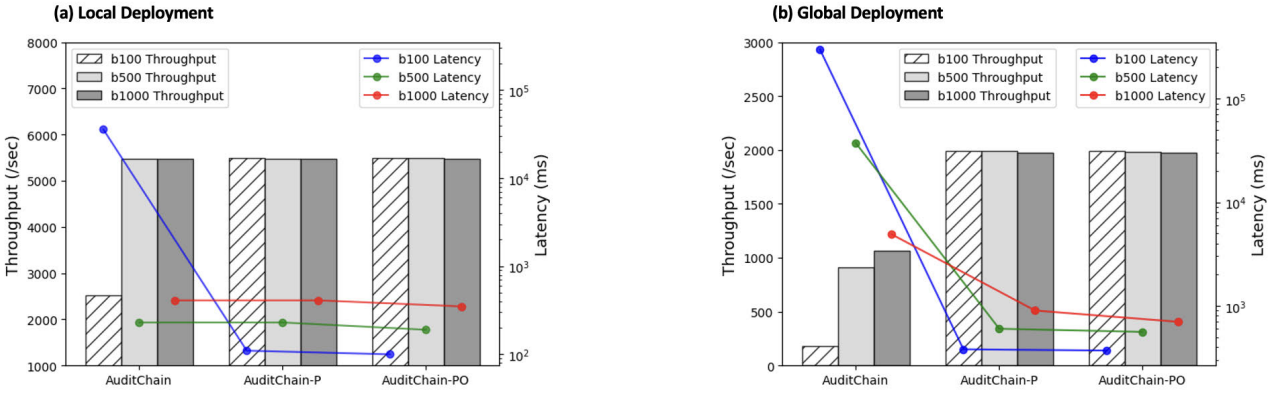

FIGURE 8. Evaluating throughput and average latency (with log scale) at the two types of deployments.

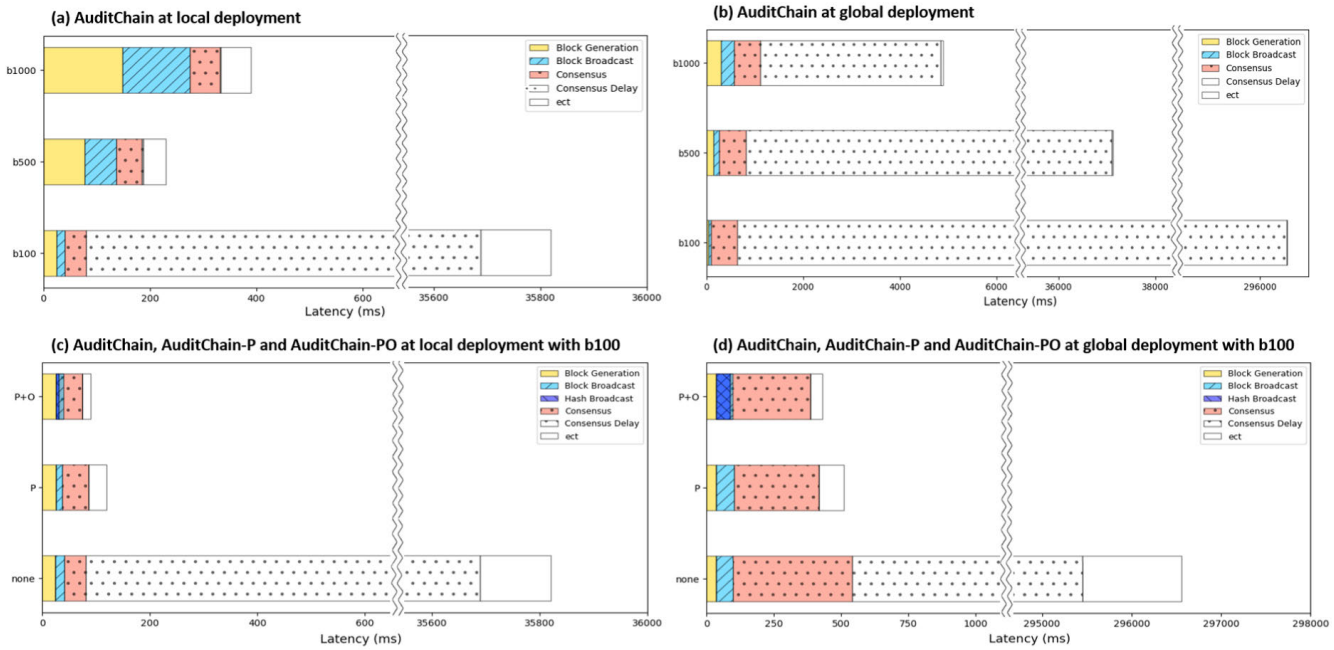

FIGURE 9. Profiling data for the experimental results in Figure 8.

while 400 was used in global deployment. We compare the performance of AuditChain, AuditChain-P, and AuditChain-PO. Figure 8 shows the results of the experiments. To analyze the results in Figure 8, the time for each section was profiled, and the results are shown in Figure 9.

In Figure 8, AuditChain in local deployment saturates faster than AuditChain in global deployment. As seen in Figure 9, the block generation and broadcast time increase as the batch size increases. Moreover, deploying the auditor globally increases block propagation and consensus time. The consensus delay in AuditChain becomes larger as the batch size decreases because the cycle for the BSP to generate blocks becomes shorter, but the consensus speed cannot keep up. The pipelining technique increases throughput and reduces latency by processing consensus instances in parallel and eliminating consensus delay. Figure 8 shows the most significant effect of the pipelining technique with a batch size of 100. AuditChain-P has 117% increased throughput and 99% reduced latency at local deployment and 977% increased throughput and 99% reduced latency at global deployment compared with AuditChain. In Figure 8 (a), for batch sizes of 500 and 1000, the pipelining effect is negligible because performance is already saturated. The overlapping technique reduces latency because they initiate consensus faster by the difference between block broadcast time and hash broadcast time. The difference is larger with larger batch sizes and is larger for the global deployment than for the local deployment. Figure 8 shows the greatest effect of the overlapping technique, with a batch size of 1000 being the largest difference. AuditChain-PO has a 22% reduced latency at the local deployment compared with AuditChain-P.

C. DECENTRALIZATION FACTORS

To analyze the performance improvement techniques that can be achieved through the techniques when adding

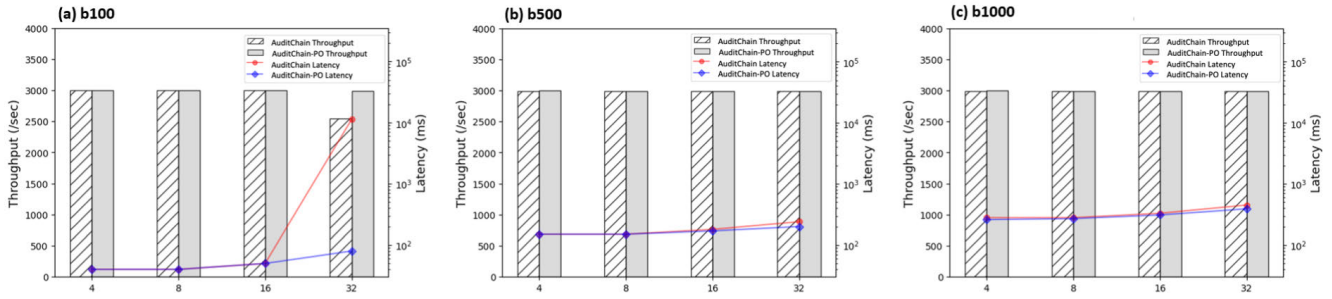

FIGURE 10. Evaluating throughput and average latency (log scale) by varying batch size and the number of auditors.

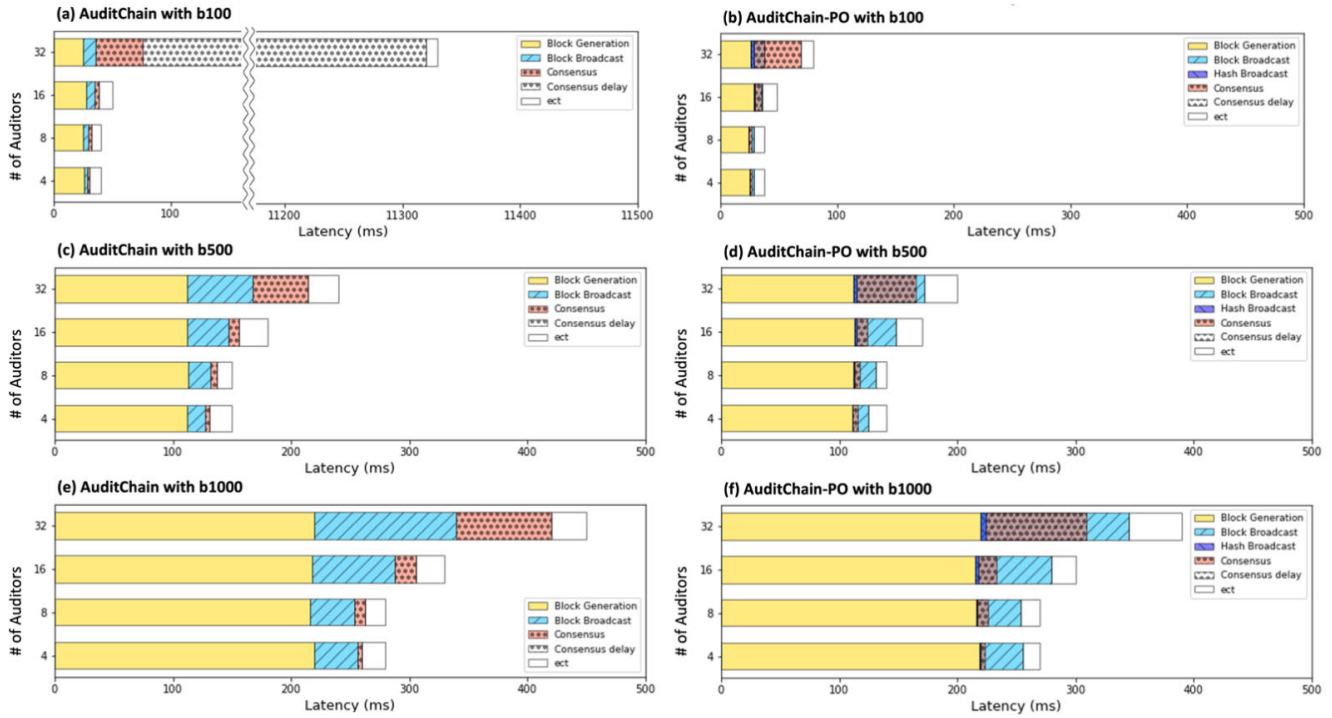

FIGURE 11. Profiling data for the experimental results in Figure 10.

decentralized factors (e.g., reducing the number of consensus nodes or deploying consensus nodes in globally), we perform two experiments and compare the performance of AuditChain and AuditChain-PO.

1) INCREASING THE NUMBER OF CONSENSUS NODES

We conduct experiments where we increased the number of auditors from 4 to 32 and used batch size = [100, 500, 1000], and send rate per client = 600 in local deployment. Figure 10 shows the results of these experiments. To analyze the results in Figure 10, the time for each section was profiled, and the results are shown in Figure 10. With a batch size of 100, AuditChain's performance degraded dramatically as the number of auditors increased from 4 to 32, due to the consensus delay. As mentioned earlier, consensus delay occurs because consensus between nodes takes longer than block generation and broadcast time. For 4, 8, and 16 nodes, the consensus time is shorter than block generation and broadcast time. However, when the number of nodes increases to 32, the consensus time becomes longer than the block generation and broadcast time as the number of nodes increases. However, AuditChain-PO maintained a constant throughput with slightly increased latency as the number of auditors increased to 32. This is because consensus starts faster due to the difference between block broadcast and block hash broadcast time, without causing consensus delays through performance improvement techniques. When using batch sizes of 500 and 1000, both AuditChain and AuditChain-PO have constant throughput with slightly increased latency as the number of auditors increased to 32. This is because the block generation and block broadcast time increase as the block size increases, and they become longer than the consensus time, as shown in Figure 12 (c-f). As with a batch size of 100, there is a point where increasing the number of nodes with the same batch size increases consensus time and incurs consensus overhead. However, AuditChain's performance is

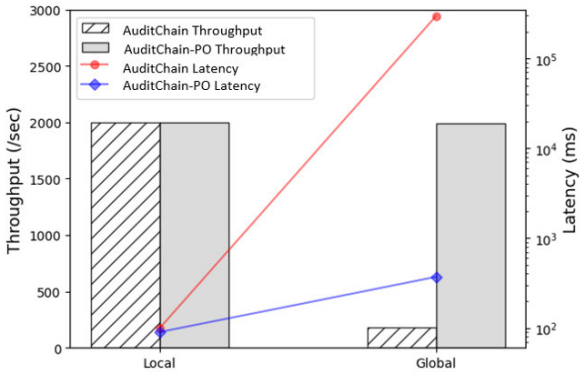

FIGURE 12. Comparing throughput and average latency (with log scale) when expanding the node's deployment.

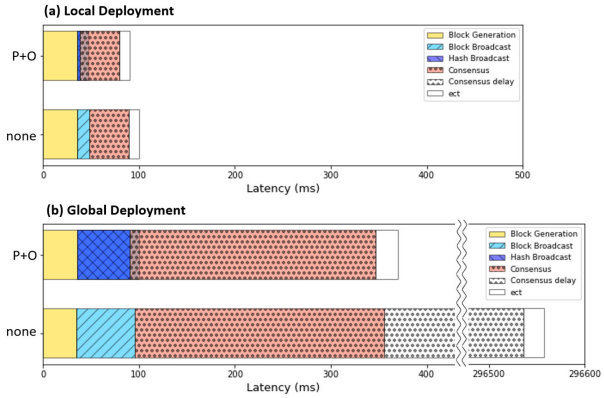

FIGURE 13. Profiling data for the experimental results in Figure 12.

expected to degrade significantly, whereas AuditChain-PO's performance is expected to remain stable, similar to Figure 10 (a).

2) DEPLOYING CONSENSUS NODES GLOBALLY

We compare the performance results of increasing the node dispersion. We use 32 auditors, send rate per client = 400, batch size = 100, and change the auditor deployment from local to global. Figure 12 shows the results of these experiments. To analyze the results in Figure 12, the time for each section was profiled, and the results are shown in Figure 13. When changing the auditor deployment, AuditChain drastically drops in performance. We can expect block broadcast and consensus time to increase as we change node deployment from local to global. In this case, the increase in consensus time is greater than the increase in block broadcast time (Figure 13). However, AuditChain-PO has constant throughput and slightly increased latency. As the distance between nodes increases by changing the auditor deployment from local to global, the latency increases because the consensus and block broadcast time increase. In the case of AuditChain, the latency increases by 2965x, and in the case of AuditChain-PO, it increases by 4x. AuditChain in local deployment achieves around 1997 tx/s with a latency of 0.1 seconds. AuditChain-PO in global deployment still achieves about 1995 tx/s with a latency of 0.3 seconds.

VIII. CONCLUSION

In this paper, we proposed and demonstrated the improvement of performance scalability of BFT for blockchain system. The pipelining technique streamlines the consensus process, increasing throughput by maximizing network resource usage. The overlapping technique processes block broadcast and the consensus process in parallel for faster consensus initiation, reducing the latency. Experiments on AWS showed that both proposed techniques improve both throughput and latency using decentralization factors such as the number of nodes and geographical deployment of nodes.

REFERENCES

[1] C. Natoli and V. Gramoli, "The blockchain anomaly," in *Proc. IEEE 15th Int. Symp. Netw. Comput. Appl. (NCA)*, Oct. 2016, pp. 310–317.

[2] S. Nakamoto. (2008). *Bitcoin Whitepaper*. Accessed: Jul. 17, 2019. [Online]. Available: https://bitcoin.org/bitcoin

[3] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," *Proc. OSDI*, vol. 99, pp. 173–186, Mar. 1999.

[4] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "HotStuff: BFT consensus with linearity and responsiveness," in *Proc. ACM Symp. Princ. Distrib. Comput.*, Jul. 2019, pp. 347–356.

[5] G. Golan Gueta, I. Abraham, S. Grossman, D. Malkhi, B. Pinkas, M. Reiter, D.-A. Seredinschi, O. Tamir, and A. Tomescu, "SBFT: A scalable and decentralized trust infrastructure," in *Proc. 49th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2019, pp. 568–580.

[6] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine agreements for cryptocurrencies," in *Proc. 26th Symp. Operating Syst. Princ.*, Oct. 2017, pp. 51–68.

[7] E. Buchman, "Tendermint: Byzantine fault tolerance in the age of blockchains," Ph.D. thesis, Dept. Eng. Syst. Comput., Univ. Guelph, 2016.

[8] G. Danezis, L. Kokoris-Kogias, A. Sonnino, and A. Spiegelman, "Narwhal and tusk: A DAG-based mempool and efficient BFT consensus," in *Proc. 17th Eur. Conf. Comput. Syst.*, Mar. 2022, pp. 34–50.

[9] C. Natoli, J. Yu, V. Gramoli, and P. Esteves-Verissimo, "Deconstructing blockchains: A comprehensive survey on consensus, membership and structure," 2019, *arXiv:1908.08316*.

[10] L. Lamport, R. Shostak, and M. Pease, "The Byzantine generals problem," in *Concurrency: The Works of Leslie Lamport*, 2019, pp. 203–226.

[11] M. Fischer and Y. Maus, "DISC 2017 review," *ACM SIGACT News*, vol. 48, no. 4, pp. 94–99, Dec. 2017.

[12] C. Berger and H. P. Reiser, "Scaling Byzantine consensus: A broad analysis," in *Proc. 2nd Workshop Scalable Resilient Infrastructures for Distrib. Ledgers*, Dec. 2018, pp. 13–18.

[13] N. Singh and M. Vardhan, "Computing optimal block size for blockchain based applications with contradictory objectives," *Proc. Comput. Sci.*, vol. 171, pp. 1389–1398, 2020.

[14] A. Kiayias, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Proc. Annu. Int. Cryptol. Conf.* Cham, Switzerland: Springer, 2017.

[15] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song, "The honey badger of BFT protocols," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016. pp. 31–42.

[16] I. Kaklamanis, L. Yang, and M. Alizadeh, "Poster: Coded broadcast for scalable leader-based BFT consensus," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2022. pp. 3375–3377.

[17] L. Yang, "DispersedLedger: High-throughput Byzantine consensus on variable bandwidth networks," in *Proc. 19th USENIX Symp. Networked Syst. Des. Implement. (NSDI)*, 2022, pp. 1–21.

[18] S. Peng, Y. Liu, J. Chen, J. He, and Y. Wang, "Komorebi: A DAG-based asynchronous BFT consensus via sharding," in *Proc. IEEE Symp. Comput. Commun. (ISCC)*, Jul. 2023, pp. 1221–1227.

[19] M. M. Jalalzai, "Fast-HotStuff: A fast and robust BFT protocol for blockchains," *IEEE Trans. Dependable Secure Comput.*, early access, Aug. 25, 2023, doi: 10.1109/TDSC.2023.3308848.
[20] D. Zhai, J. Wang, J. Liu, T. Liu, and W. Niu, "Efficient-HotStuff: A BFT blockchain consensus with higher efficiency and stronger robustness," in *Proc. IEEE 28th Int. Conf. Parallel Distrib. Syst. (ICPADS)*, Jan. 2023, pp. 217–225.
[21] (2024). [Online]. Available: https://github.com/sslab-at-postech/Technical-Report/tree/main/AuditChain
[22] X. Yang, "LedgerDB: A centralized ledger database for universal audit and verification," *Proc. VLDB Endowment*, vol. 13, no. 12, pp. 3138–3151, 2020.
[23] T. McConaghy et al., "BigchainDB: A scalable blockchain database," BigChainDB, Berlin, Germany, White Paper, 2016.
[24] (2021). *Postchain*. [Online]. Available: https://postchain-docs.readthedocs.io/en/latest

HANEUL OH received the B.S. degree in computer science and engineering from Chungnam National University, South Korea, in 2017, and the M.S. degree from Pohang University of Science and Technology (POSTECH). Her research interests include distributed systems and blockchain.

CHANIK PARK received the B.S. degree in electronics engineering from Seoul National University, Seoul, South Korea, in 1983, and the M.S. and Ph.D. degrees in electronics and electrical engineering (computer engineering) from KAIST, Daejeon, South Korea, in 1985 and 1988, respectively. Since 1989, he has been a Professor with the Department of Computer Science and Engineering, Pohang University of Science and Technology (POSTECH). He was a Visiting Scholar with the Parallel Systems Group, IBM T. J. Watson Research Center, and a Visiting Professor with the Storage Systems Group, IBM Almaden Research Center. He also visited Northwestern University and Yale University, in 2009 and 2015, respectively. His research interests include storage systems, operating systems, and system security, with the recent addition of blockchain. He has served as a program committee member for a number of international conferences and workshops.