Hindawi
Security and Communication Networks
Volume 2021, Article ID 2507670, 10 pages
https://doi.org/10.1155/2021/2507670

Research Article

Lodestone: An Efficient Byzantine Fault-Tolerant Protocol in Consortium Blockchains

Chen Shan and Lei Fan

School of Cyber Science and Engineering, Shanghai Jiao Tong University, Shanghai 201100, China

Correspondence should be addressed to Lei Fan; fanlei@sjtu.edu.cn

Received 29 October 2021; Accepted 16 November 2021; Published 3 December 2021

Academic Editor: Yuling Chen

We present Lodestone, a chain-based Byzantine fault-tolerant (BFT) state machine replication (SMR) protocol under partial synchrony. Lodestone enables replicas to achieve consensus with two phases of voting and enjoys (1) optimistic responsiveness and (2) linear communication complexity on average. Similar to the state-of-the-art chain-based BFT protocols, Lodestone can be optimized with a pipelining idea elegantly. We implement pipelined Lodestone and deploy experiments to evaluate its performance. The evaluation results demonstrate that Lodestone has a lower latency than HotStuff under various workloads.

1. Introduction

Consensus in blockchain systems, known as state machine replication (SMR), has attracted more and more interest in recent years. When focusing on permissioned blockchains on so-called consortium blockchains, chain-based Byzantine fault-tolerant (BFT) SMR protocols [1–8] under partial synchrony have been widely used to achieve consistency. In general, chain-based BFT SMR protocols follow the conventional propose-vote paradigm where there exists a special role often called leader who is responsible for packing clients' requests into proposals, and then all players achieve consensus on these proposals via multiple (two or three) phases of voting.

PBFT [9], as the first practical BFT SMR protocol under the partial synchronous network [10], achieves safety even under the asynchronous network and liveness when the network gets synchronous. However, the view-change subprotocol in PBFT, with an $O(n)^3$ communication complexity, is too heavy to be practical. Tendermint [6] innovatively employs a lock-commit scheme, similar to the paradigm in [11], that a replica should lock on the proposal he has voted COMMIT for. This allows one replica to decide if voting for one proposal according to his own local states and the leader has no need to prove the safety of his proposal. Casper [7, 8] takes a similar strategy and also implies a pipelining idea for a further improvement. However, both Tendermint and Casper sacrifice optimistic responsiveness in that there needs to be a fixed interval between proposals to guarantee liveness since a new leader has to ensure that he has observed all other nonfaulty replicas' lock state; otherwise, his new proposal may not be accepted. HotStuff [1] creatively introduces another phase of vote to achieve both linear view-change and optimistic responsiveness. The additional phase guarantees that a new leader can construct this proposal safely only with $n - f$ replicas' states. However, in pipelined HotStuff, the three-phase voting scheme not only brings about an increase in latency but also causes an implicit liveness problem. In pipelined HotStuff, there needs to be four consecutive nonfaulty leaders to make a decision which cannot be guaranteed in the $n = 3f + 1$ setting. This means pipelined HotStuff cannot provide liveness in the worst case even under the crash fault-tolerant model, which has also been discussed as the silence attack [12].

We present Lodestone, a novel chain-based BFT SMR protocol, which achieves the following combined properties under partial synchrony:

(1) Two-phase voting with optimistic responsiveness: Lodestone can achieve responsiveness when liveness is guaranteed after GST. That is, the total time of confirmation on one honest leader's proposal only relies on the actual network delay instead of any apriori upper bound assumption of network delay.

(2) Linear average-case communication complexity: a view-change subprocess in Lodestone costs $O(n)$ message complexity on average and $O(n^2)$ in the worst case. Here, we follow the measurement of message complexity in HotStuff [1] which counts the total number of authenticators received by one player in the protocol to achieve a decision.

(3) Deterministic liveness under static corruption: pipelined Lodestone can also achieve deterministic liveness under static corruption where leaders are rotated in a round-robin manner.

The difference in detail between these protocols can be shown in figures, of which Figures 1 and 2 show HotStuff and existing two-phase voting protocols, respectively, while Figure 3 demonstrates our solution.

2. Other Related Works

2.1. BFT SMR Protocols in Alternative Assumptions. There are also many works considering about BFT protocols in alternative assumptions.

Firstly, about the network assumption, many recent protocols under a synchronous network [13–15] or under an asynchronous network [16–18] make efforts to reduce confirmation latency and achieve practical throughput. Secondly, about the corruption assumption, some recent works [19, 20] are aimed to provide higher assurance on blocks even if the adversary corrupts more than f replicas in the future. In this paper, we only concentrate on static security under a partially synchronous network.

2.2. Single-Shot BFT Protocols. There are some related works [21–23] focusing on the single-shot BFT problem which explore the optimal latency bound when the leader is nonfaulty under various resilience assumptions. Though it is still a long way to construct a protocol from single shot to multishots, some results are interesting and may be combined with our protocols in the future work.

2.3. View Synchronization. Another related line of work is about view synchronization [24–26] in the partial synchrony setting. It is also necessary in Lodestone for nonfaulty replicas to stay in the same view for a sufficient long time, and then liveness can be guaranteed. However, view synchronization is not the key point of Lodestone, and we assume that replicas are able to stay in the same view for a sufficient long time after GST.

3. Materials and Methods

3.1. Models

(1) Threat model: we consider a permissioned system consisting of n replicas, indexed by $i \in [n]$, where $[n] = \{1, 2, \ldots, n\}$. We assume a polynomially bounded adversary who can corrupt less than $n/3$ replicas. The replicas corrupted by the adversary can deviate from the prescribed protocol arbitrarily within their capabilities which are also called Byzantine faulty replicas, while the remaining ones are nonfaulty. We only consider a static corruption model in that the adversary chooses which replicas to corrupt prior to the execution.

(2) Network model: we assume that each pair of replicas is connected by a reliable authenticated point-to-point channel. Messages are propagated through a partially synchronous network [10] in that there is an unknown global stabilization time (GST). After GST, a message sent by a nonfaulty replica will be delivered to all nonfaulty replicas with a known bound Δ, though the delivery schedule is determined by the adversary.

(3) Cryptographic primitives: we assume a cryptographic hash function $\mathrm{Hash}(m)$ and a standard digital signature scheme. We also assume a (t, n) threshold signature scheme [27, 28] which provides the following interfaces:

$\mathrm{ThresholdSetup}(1^\lambda)$ generates a pair of key shares $\{pk_i, sk_i\}$ for replica i along with a global public key PK.

$\mathrm{ThresholdSign}_i(m)$ produces a signature share λ_i of message m with sk_i.

$\mathrm{ThresholdVerifyShare}(m, i, \lambda_i)$ verifies if λ_i is a valid signature share of message m.

$\mathrm{ThresholdCombine}(m, i, \lambda_{i,i\in I})$ produces the threshold signature λ of message m from t signature shares where $I \subset [n]$ and $|I| = t$.

$\mathrm{ThresholdVerify}(m, \lambda)$ verifies if λ is a valid threshold signature of message m with PK.

We use an $(n - f, n)$ threshold signature scheme in our following protocols which is assumed to provide robustness and nonforgeability.

(4) Problem definition: we now give the definition of a chain-based BFT SMR protocol. Each replica in a chain-based BFT SMR protocol receives requests from clients and maintains a sequence of blocks called a blockchain. Blocks in a blockchain are chained by hash digest, and thus, each block in a blockchain has its own position denoted as its height. Given a blockchain C and a block $b \in C$, all blocks in C lower than b are ancestor blocks of b, and all blocks in C higher than b are descendant blocks of b. Two blocks b_1 and b_2 are conflicting if and only if b_1 is neither an ancestor nor a descendant block of b_2. Each block includes a batch of requests, and one

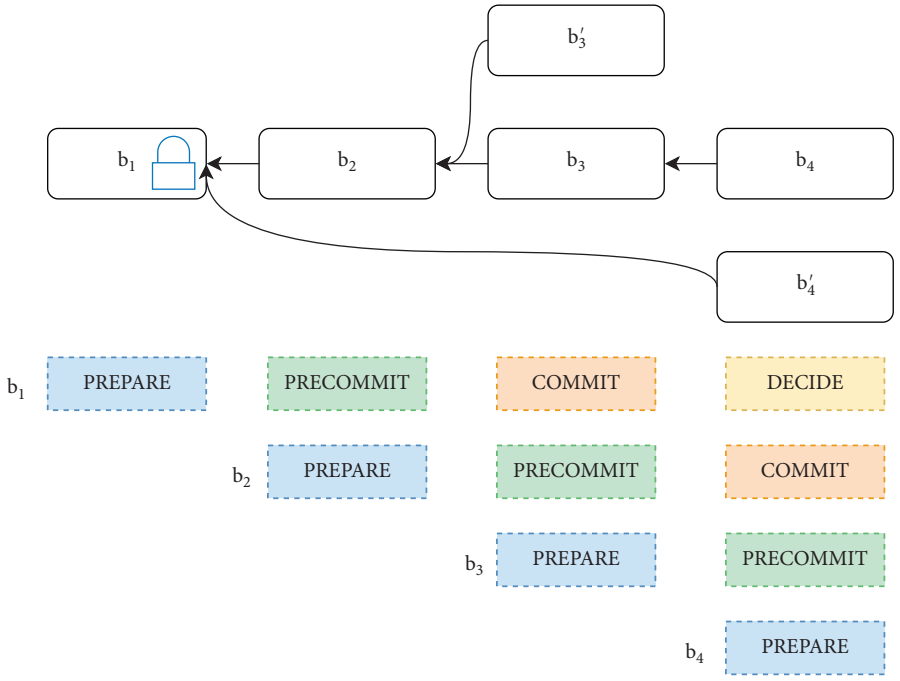

Figure 1: Pipelined HotStuff. b_1, b_2, b_3, and b_4 are in the normal case, while b_1, b_2, b_3', and b_4' are in the timeout case. In the timeout case, nonfaulty replicas who have received b_3' will lock on b_1 but will still vote for b_4' since b_4' also extends from b_1.

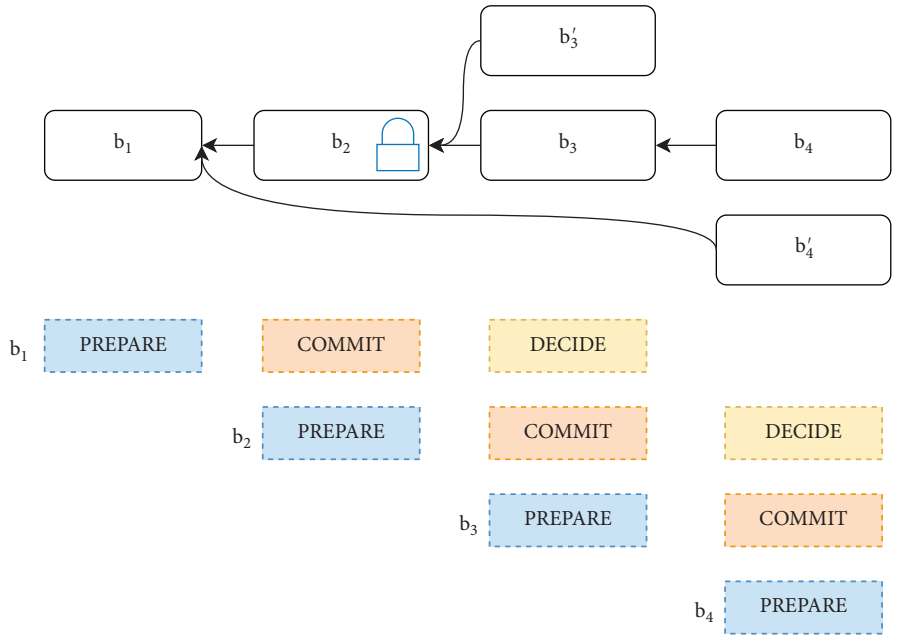

Figure 2: Casper or the two-chain variant of pipelined HotStuff. b_1, b_2, b_3, and b_4 are in the normal case, while b_1, b_2, b_3', and b_4' are in the timeout case. In the timeout case, the leader of b_4' did not receive b_3' in time, while some nonfaulty replicas received b_3' and have locked on b_2. Therefore, they would not vote for b_4' since b_4' conflicts with b_2.

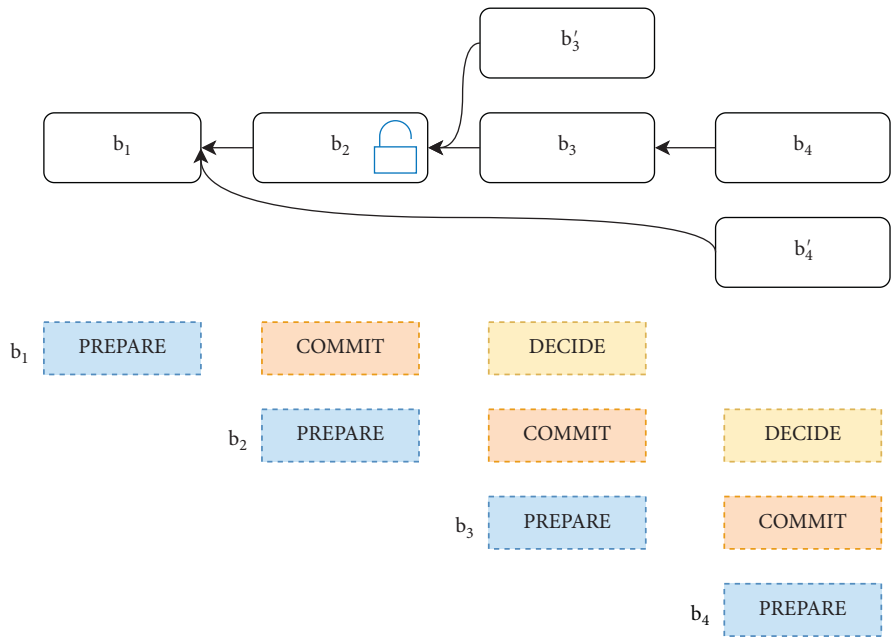

FIGURE 3: Pipelined Lodestone. b_1, b_2, b_3, and b_4 are in the normal case, while b_1, b_2, b_3', and b_4' are in the timeout case. In the timeout case, the leader of b_4' will attach a proof to his proposal which allows nonfaulty replicas who have locked on b_2' to get unlocked and vote for b_4'.

replica's blockchain consists of all blocks which have been finalized. Replicas execute requests in finalized blocks in the sequence of the blockchain and then respond to clients. For simplicity, we do not model clients and assume all requests are sent to all replicas. Thus, we say requests from clients are input to all replicas, and at any time, the output of one replica is his blockchain.

A secure chain-based BFT SMR protocol should satisfy both safety and liveness defined below (in the presence of an arbitrary adversary with all but negligible probability).

Definition 1. (safety). At any time, if two nonfaulty replicas each have their blockchains denoted as C_1 and C_2, then it must be either $C_1 \preccurlyeq C_2$ or $C_2 \preccurlyeq C_1$, where $\preccurlyeq$ means "is a prefix of or equal to." In other words, two nonfaulty replicas will never finalize different blocks at the same height.

Definition 2. (liveness). If a request *req* has been inputted to all replicas at time $t > T_{\text{start}}$, then at time $t + T_{\text{confirm}}$, any nonfaulty replica must output a blockchain which includes *req*, where T_{start} is the time after which the protocol provides liveness and T_{confirm} is a bounded constant.

3.2. Preliminaries

(1) Block: we now format a block b in our protocol as b = parent, view, txs, and hash where parent is the hash digest of the parent block of b, view is the view in which b is proposed, txs is a batch of requests from clients, and hash is the hash digest of b (i.e., b.hash = Hash (b.parent, b.view, b.txs)). In Lodestone, there are three states for one block, namely, PROPOSED, PREPARED, and COMMITTED. Once a block reaches the COMMITTED state, then the block itself and its all ancestor blocks are finalized and can be executed sequentially.

(2) Local states: replicas in Lodestone need to maintain two local states for the protocol execution. The first denoted as *currView* represents the current view number and implies the current leader. It is noted that using the round-robin manner to rotate leaders guarantees that (i) the protocol will be greeted with a nonfaulty leader after at most f consecutive views; (ii) there exist three consecutive views of which leaders are all nonfaulty which will be proved later. The second is *lockedQC* which stores *qc* with the largest view number one replica has voted commit for. Once a nonfaulty replica locks on *qc* (i.e., sets *lockedQC* as *qc*), he will only vote for blocks extending from block b that Hash (b) = qc.hash unless he ensures the majority of nonfaulty replicas have turned to a conflicting branch before locking on *qc*.

(3) Promise and promise-set: during a view-change, each replica additionally sends to the leader a *promise*, evidence that helps the leader locate the highest COMMITTED block and prove that he does so in an honest manner. A *promise* for view v indicates that the replica has not voted commit at view v. If $n-f$ replicas send *promises* for the view v, then there is no block which has a larger view number than v could be COMMITTED. When the leader proposes a block with the $n-f$ *promises*, all the nonfaulty replicas can accept it safely. A *promise* from replica P includes a tuple $v, \hat{v}$ and the corresponding signature share σ = ThresholdSign ($v, \hat{v}$)

from P. We denote $p = v, \widehat{v}, \sigma$ for simplicity. We say p from P for the current view $\widehat{v}$ represents that he did not vote commit when he was in the view v. Let $\widetilde{v}$ be the last view in which P has voted commit. P will generate a set of *promises* for each view from $\widetilde{v}+1$ to *currView*. We denote it as a *promise-set*.

(4) NullQC: before entering a new view, each nonfaulty replica P_i will send his promise-set *promise-set* to the new leader, along with his *lockedQC*. Upon receiving valid promise-sets from $n-f$ distinct replicas, the new leader will select *lockedQC* with the largest view number as *highQC* and propose his new block following the block b where Hash(b) = highQC.hash. Let I be the set of indexes of the $n-f$ replicas; we have $|I| = n-f$. We use $\widehat{v}$ to denote *currView* and $\widetilde{v}$ to denote *highQC.view*. Each promise-set must contain a *promise* that promise$_i^{v,(v+1)} = \widetilde{v}+1, \widehat{v}, \lambda_i^{v,(v+1)}$ where $i \in I$. The new leader can combine these $n-f$ signature shares as a threshold signature to prove that *highQC* he selected is exactly the one with the largest view number among $n-f$ replicas. This extra proof is denoted as *nullQC* in Lodestone. We have nullQC = ThresholdCombine$(\widetilde{v}+1, \widehat{v}, \{\lambda_i^{\widehat{v},(\widetilde{v}+1)}\})$, where $i \in I$.

3.3. Pipelined Lodestone. We are now ready to describe our protocol pipelined Lodestone. We first define some utilities as shown in Figure 4.

Then, we formalize pipelined Lodestone with algorithms. The protocol runs in a succession of views denoted as *currView*. Each view number is mapped into a leader in a round-robin manner. The leader will execute Algorithm 1, all replicas will execute Algorithm 2, and then the next leader will execute Algorithm 3. When timeout triggers during any wait-for procedure in one replica's local view, he will execute Algorithm 4. We omit any check for brevity.

4. Safety, Liveness, and Communication Complexity

4.1. Safety

Lemma 1. *For any two valid quorum certificates qc_1 and qc_2, if $qc_1 \cdot view = qc_2 \cdot view$, then $qc_1 \cdot block = qc_2 \cdot block$.*

Proof. For any valid qc, at least $n-2f$ nonfaulty replicas have sent their relevant signature shares, namely, at least $n-2f$ nonfaulty replicas have voted for the block which *qc.block* represents in *qc.view*. Suppose qc_1 and qc_2 are two valid qc such that $qc_1 \cdot$ view $= qc_2 \cdot$ view, but $qc_1 \cdot$ block $\neq qc_2 \cdot$ block. We must have that at least $n-2f$ nonfaulty replicas voted for the $qc_1 \cdot$ block and also at least $n-2f$ nonfaulty replicas voted for the $qc_2 \cdot$ block in the same view. Thus, the intersection of the two sets at least includes one nonfaulty replica since $2 \times (n-2f) > n-f$, a contradiction to Algorithm 2 that a nonfaulty replica can only vote once in a view.

Theorem 1. *In pipelined Lodestone, two conflicting blocks cannot be both COMMITTED.*

Proof. Let b_1 and b_2 be two conflicting blocks which are both COMMITTED, namely, there exists a two-chain $\langle b\_1, b\_1', b\_1'' \rangle$ and also one in the form of b_2, $b\_2\prime$, $b\_2\prime$. W.l.o.g, we can assume that $b_2 \cdot$ view $> b_1' \cdot$ view with Lemma 1. Since b_1 is COMMITTED, at least $n-2f$ nonfaulty replicas have voted for b_1' and then locked on b_1. Let $\widehat{b}$ be the PREPARED block with the smallest view number that satisfies $\widehat{b}$.view $> b_1'$.view, and $\widehat{b}$ conflicts with b_1. Such $\widehat{b}$ must exist since b_2 satisfies all these conditions. Now, consider the proposal m of $\widehat{b}$. At least $n-2f$ nonfaulty replicas have locked on b_1 in $b_1' \cdot$ view. These $n-2f$ nonfaulty replicas will only promise on view number larger than $b_1 \cdot$ view since then. Therefore, $m \cdot$ proof $\cdot$ view $> b_1 \cdot$ view. And due to the minimality of $\widehat{b}$, we must have $m \cdot qc$.view $< b_1 \cdot$ view. Then, m.proof.view $> m.qc$.view $+ 1$, a contradiction to the validity of *nullQC*.

4.2. Liveness

Lemma 2. *Under an $n = 3f+1$ setting where leaders are rotated in a round-robin manner, in any consecutive $n+2$ views, there are three consecutive views all leaded by nonfaulty replicas.*

Proof. Consider any n consecutive views denoted as $v, v+1, v+2, \ldots, v+n-1$. There are three cases:

(i) The leader of v is faulty. Then, for the view from $v+1$ to $v+n-1$, there are $f-1$ faulty leaders while $2f+1$ nonfaulty leaders. The $f-1$ faulty leaders can at most separate $2f+1$ nonfaulty leaders into f segments, with at least one segment owning 3 nonfaulty leaders.

(ii) The leader of v is nonfaulty, while the leader of $v+1$ is faulty. Then, for the view from $v+2$ to $v+n$, there are $f-1$ faulty leaders while $2f+1$ nonfaulty leaders. Then, the situation becomes the same as the first case.

(iii) Both v and $v+1$ are leaded by nonfaulty leaders. Then, if the leader of $v+2$ is nonfaulty, three consecutive nonfaulty leaders exist. Otherwise, for the view from $v+3$ to $v+n+1$, there are $f-1$ faulty leaders while $2f+1$ nonfaulty leaders. Then, the situation becomes the same as the first case.

Lemma 3. *If a nonfaulty leader of view v collects $n-f$ valid view-change messages, he can always propose his new block, along with valid nullQC that nullQC $\cdot$ view = highQC$\cdot$ view + 1 where highQC points to the parent block of his new block.*

Proof. Let P_i be the nonfaulty leader of view v. If P_i collects $n-f$ valid view-change messages, he will select qc with the largest view number among them as *highQC*. Each of the $n-f$ valid view-change messages must also include a promise on highQC.view + 1; then, P_i can combine a threshold signature with these $n-f$ signature shares and construct *nullQC* that nullQC $\cdot$ view = highQC $\cdot$ view + 1.

```
function VOTE (view,hash,id)
    v.id=i  d
    v.view=view
    v.hash=hash
    v.sigShare=ThresholdSigni (v)
    return v
end function

function MSG (block, qc, proof, view)
    m.block=block
    m.qc=qc
    m.proof=proof
    m.view=view
    return m
end function

function BLOCK (view,parent)
    b.view=view
    b.requests=requests
    b.parent=parent
    b.hash= Hash (b)
    return b
end function

function QC (view, hash, V)
    qc.view=view
    qc.block=hash
    qc.proof=ThresholdCombine (qc, {⟨v. id, v. sigShare⟩| v ∈ V})
    return qc
end function

function NULLQC (view, currView, T)
    nullQC.view=view
    Σ = {⟨i, t. σ_i^{view,currView}⟩|∀ T_i ∈ T, t ∈ T_i
    nullQC.proof=ThresholdCombine (⟨view, currView⟩ Σ)
    return nullQC
end function

function PROOF (startView, currView, id)
    T=Ø
    for v=startView to currView do
      σ_i^{v,v̂} = ThresholdSign_i (⟨v,v̂⟩)
      T = T ∪ {⟨⟨v,v̂⟩, σ_i^{v,v̂}⟩}
    end for
    return T
end function
```

Figure 4: Utilities in the pipelined Lodestone protocol.

(1) Wait for $n - f$ VIEW-CHANGE message $m \in M$ of currView-1
(2) $highQC \leftarrow \text{argmax}_{m \in M}\{m.qc.view\}$
(3) if $lockedQC.view < highQC.view$
(4) $\quad lockedQC \leftarrow highQC$
(5) $\widehat{T} \leftarrow \{m.proof | m \in\}$
(6) $nullQC \leftarrow NullQC(highQC.view + 1, currView, \widehat{T})$
(7) $b \leftarrow BLOCK(currView, lockedQC.hash)$
(8) Broadcast MSG (PROPOSE, b, *lockedQC, NullQC*)

Algorithm 1: Pipelined Lodestone protocol: as the leader of *currView*.

(1) Wait for RROPOSE message m from LEADER ($currView$)
(2) $b \leftarrow m.block$
(3) **if** $m.qc.view + 1 = m.proof.view$.
(4) $\quad v \leftarrow VOTE(currView, b, hash, i)$
(5) $\quad$ send MSG (***GENERIC***, b, $hash$, $\perp$, v) to LEADER (currView + 1).
(6) **if** $m.qc.view > lockedQC.view$
(7) $\quad lockedQC \leftarrow m.qc$
(8) $b' \leftarrow b.parent, b'' \leftarrow b'.parent$
(9) **if** $b''.view + 1 = b'\,parent$
(10) $\quad$ DECIDE (b'')

Algorithm 2: Pipelined Lodestone protocol: as a replica P_i of $currView$.

(1) Wait for $n - f$ matched GENERIC messages $\widehat{M}$ of $CurrView$
(2) $\quad hash \leftarrow \hat{m}.block, \forall \hat{m} \in \widehat{M}$
(3) $\quad \widehat{V} \leftarrow \{\hat{m}.proof \mid \hat{m} \in \widehat{M}\}$
(4) $\quad lockedQC \leftarrow QC(currView, hash, \widehat{V})$

Algorithm 3: Pipelined Lodestone protocol: as the next leader of $currView$.

Jump here if TIMEOUT triggers during any wait procedure or before entering currView+1
(1) $T \leftarrow PROOF(lockedQC.view + 1, currView + 1, i)$
(2) send MSG (VIEW-CHANGE, $\perp$, $lockedQC$, T) to LEADER ($currView + 1$)

Algorithm 4: Pipelined Lodestone protocol: TimeOut $currView$.

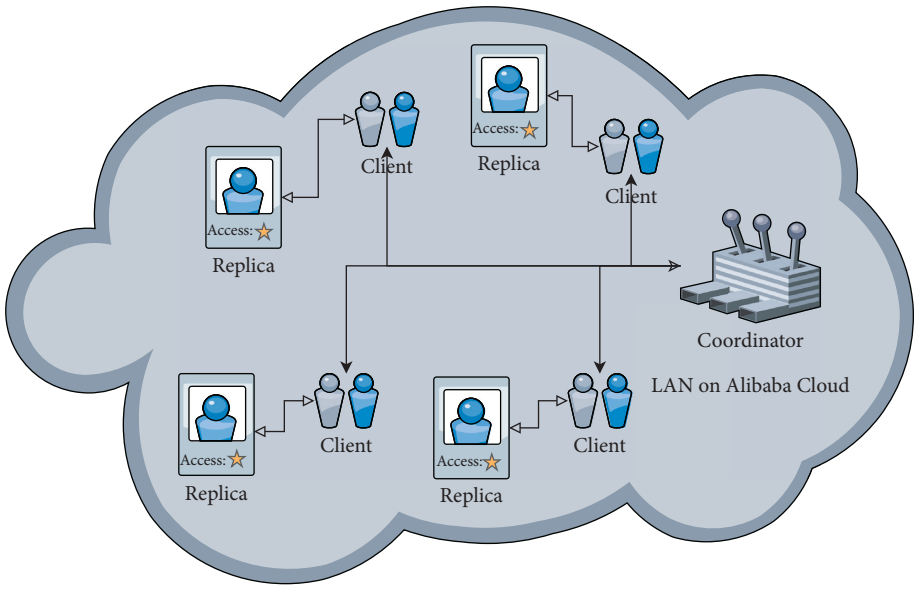

Figure 5: The architecture diagram of experiments.

Theorem 2. *In pipelined Lodestone, after GST, any request from clients will be included in the finalized block in a bounded time.*

Proof. According to Lemma 2, there are three consecutive views leaded by nonfaulty replicas in any consecutive $n + 2$ views, denoted as $v, v + 1, v + 2$. There exists a bounded time

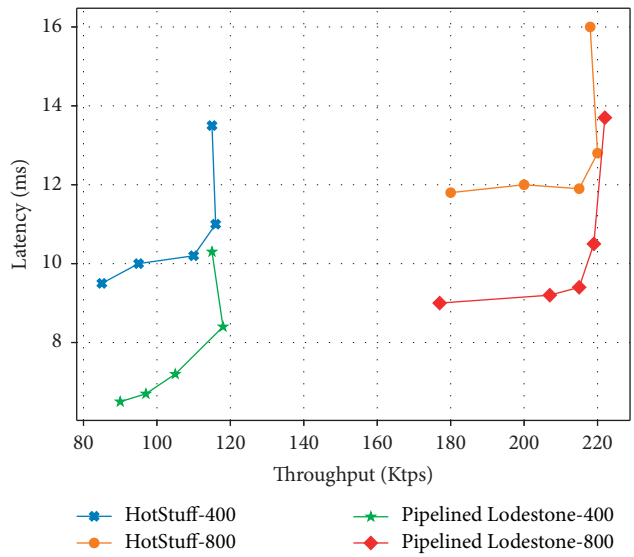

Figure 6: Throughput vs. latency with different choices of batch size, 10 replicas, and zero-sized payload.

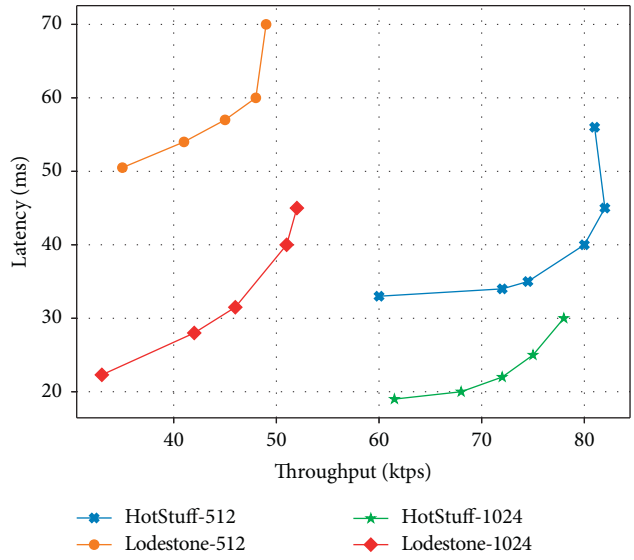

Figure 7: Throughput vs. latency with different choices of payload size, 10 replicas, and batch size of 800.

duration T that if all nonfaulty replicas stay in the same view during T, then the leader can propose his new block with nullQC · view = highQC · view + 1 according to Lemma 3. Thus, the first nonfaulty leader of v will propose a block b, and all nonfaulty replicas will vote for b. Then, the second nonfaulty leader of $v+1$ will collect $n-f$ votes for b and propose a block b' with highQC$\prime$ · view = $v\wedge$ nullQC$\prime$ · view = $v+1$. All nonfaulty replicas will also vote for b'. Then, the third nonfaulty leader of $v+2$ will collect $n-f$ votes for b' and propose a block b" with highQC$\prime\prime$.view = $v+1$. Therefore, b, b', and b" form a two-chain $b, b\prime, b\prime$, and all nonfaulty replicas will consider b as COMMITTED after receiving b". For any time t, the three consecutive views leaded by nonfaulty players will come within a bounded time duration after t. Therefore, there exists a bounded constant T_{confirm} in that for any request input to all replicas at time t, it will be included in the finalized block at $t + T_{\text{confirm}}$.

4.3. Communication Complexity. We now discuss the communication complexity of pipelined Lodestone. It is noted that we only consider the communication overhead when liveness can be guaranteed after GST. When the network is asynchronous, there may be unbounded views without making a decision. In fact, we can allow replicas at

most send f promises during a view-change which has no effect on the safety property since it is impossible to guarantee liveness under an asynchronous network [29].

Theorem 3. *After GST, pipelined Lodestone achieves a linear communication complexity on average while $O(n^2)$ in the worst case.*

Proof. When liveness can be guaranteed after GST, whenever two consecutive views are leaded by nonfaulty replicas, the second leader can generate *qc*, and all nonfaulty replicas will update their *lockedQC*. Therefore, in the worst case, there are $2f + 1$ consecutive views without generating new *qc*. And then, one replica needs to send $2f + 1$ promises on these $2f + 1$ views. And thus, the total complexity of a view-change is $O(n^2)$ in the worst case.

Now, we discuss the communication complexity of a view-change in the average case. The probability of one view leaded by a faulty replica can be considered as 1/3 while 2/3 by a nonfaulty leader approximately with the assumption $n = 3f + 1$. Given any view v, let X be the length of views before v without any two consecutive views leaded by nonfaulty replicas. Let $P(X)$ be the probability distribution of X. For any view v, $X = k$ occurs when

(i) Either the leader of $v - 1$ is faulty and $X = k - 1$ holds for $v - 1$

(ii) Or the leader of $v - 1$ is nonfaulty while the leader of $v - 2$ is faulty and $X = k - 2$ holds for $v - 2$

As the assumption, $v - 1$ and $v - 2$ also follow the same probability distribution which is denoted as $P(X = k - 1)$ and $P(X = k - 2)$, respectively. Then, $P(X = k)$ can be expressed as

$$P(X = k) = \frac{1}{3}P(X = k - 1) + \frac{2}{9}P(X = k - 2). \quad (1)$$

The expectation $E(X)$ is the length of views before v without any two consecutive views leaded by nonfaulty players on average. $E(k)$ is also the total number of signatures in the promise-set of one replica on average. We have $E(X) = 7/4$ after simplification. Therefore, the expectation of the total communication overhead of a view-change is $O(n)$ for all n replicas in a view.

5. Results and Discussion

5.1. Implementation and Setup. We have implemented both pipelined Lodestone and pipelined HotStuff in C++ language with the same codebase for a fair comparison, taking the implementation (https://github.com/hot-stuff/libhotstuff) in HotStuff's paper [1] as a reference. We use Ed25519 for common digital signatures and BLS threshold signatures (https://github.com/herumi/bls) for combining signatures in our protocol.

We deploy our experiments on Alibaba Cloud using ecs.ic5.4xlarge instances. The round-trip delay between two instances is less than 1 millisecond, with the bandwidth about 5 Gbps.

In all experiments, besides all players and clients, we develop a coordinator who is responsible for notifying all clients sending requests to players. The coordinator collects measurement data to compute the throughput and end-to-end latency of clients. Figure 5 shows the architecture in our experiments.

We first evaluate these two protocols with zero-sized payload and different choices of batch sizes to get rid of the effects of payload size. Figure 6 shows that Lodestone has a prominent lower latency compared with HotStuff under the batch size of both 400 and 800 benefiting from conserving one phase of the vote.

Figure 7 depicts different payload sizes for both systems as 512 bytes and 1024 bytes, with a fixed batch size of 800. In such settings, Lodestone still enjoys a remarkable lower latency compared with HotStuff and provides comparable throughput.

6. Conclusions

We presented pipelined Lodestone, a chain-based BFT SMR protocol, which achieves linear view-change on average and optimistic responsiveness with only two phases of voting. Through the experimental results, pipelined Lodestone provides a lower latency and comparable throughput compared with HotStuff in various workloads and network scales.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.

Conflicts of Interest

The authors declare that there are no conflicts of interest regarding the publication of this paper.

References

[1] M. Yin, D. Malkhi, K. Michael, M. K. Reiter, and G. G.. Gueta, "HotStuff: BFT consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pp. 347–356, Toronto ON Canada, July 2019.

[2] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and M. Yin, "Sync HotStuff: synchronous SMR with 2Δ latency and optimistic responsiveness," *IACR Cryptol. ePrint Arch*, vol. 2019, 270 pages, 2019.

[3] T. H. C. Hubert, R. Pass, and E. Shi, "PiLi: an extremely simple synchronous blockchain," *IACR Cryptol. ePrint Arch*, vol. 2018, 980 pages, 2018.

[4] T. H. C. Hubert, R. Pass, and E. Shi, "PaLa: a simple partially synchronous blockchain," *IACR Cryptol. ePrint Arch*, vol. 2018, 981 pages, 2018.

[5] B. Y. Chan and E. Shi, "Streamlet: textbook streamlined blockchains," in *Proceedings of the second ACM Conference on Advances in Financial Technologies*, pp. 1–11, New York, NY, USA, October 2020.

[6] E. Buchman, J. Kwon, and Z. Milosevic, "The latest gossip on BFT consensus," 2018, https://dblp.org/rec/journals/corr/abs-1807-00734.html?view=bibtex, Article ID 04938.

[7] V. Buterin and V. Griffith, "Casper the Friendly Finality Gadget," 2017, https://arxiv.org/abs/1710.09437, Article ID 09437.

[8] V. Buterin, D. Reijsbergen, S. Leonardos, and G. Piliouras, "Incentives in ethereum's hybrid casper protocol," in *Proceedings of the IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pp. 236–244, Seoul,, May 2019.

[9] M. Castro and B. Liskov, "Practical Byzantine fault tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, pp. 173–186, New Orleans, NO, USA, February1999.

[10] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM*, vol. 35, no. 2, pp. 288–323, 1988.

[11] L. Lamport, "Paxos made simple, fast, and byzantine," in *Proceedings of the Sixth International Conference on Principles of Distributed Systems. OPODIS*, pp. 7–9, Reims, France, 2002.

[12] F. Gai, F. Ali, J. Niu, C. Feng, F. Beschastnikh, and H. Duan, "Dissecting the performance of chained-BFT," in *Proceedings of the dICDCS*, pp. 595–606, Washington DC, USA, July 2021.

[13] T. Hanke, M. Movahedi, and D. Williams, "Dfinity Technology Overview Series, Consensus system," 2018, https://arxiv.org/abs/1805.04548.

[14] T. H. H. Chan, R. Pass, and E. Shi, "PiLi: An Extremely Simple Synchronous Blockchain," *Cryptology ePrint Archive*, vol. 2018, 2018.

[15] I. Abraham, D. Malkhi, K. Nayak, L. Ren, and M. Yin, "Sync hotstuff: simple and practical synchronous state machine replication," in *Proceedings of the 2020 IEEE Symposium on Security and Privacy (SP)*, pp. 106–118, IEEE, San Francisco, CA, USA, May2020.

[16] A. Miller, Y. Xia, K. Croman, and E. Shi, "The Honey badger of BFT protocols," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pp. 31–42, Vienna, Austria, October 2016.

[17] I. Abraham, D. Malkhi, and A. Spiegelman, "Asymptotically Optimal Validated Asynchronous Byzantine agreement," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pp. 337–346, Toronto, Canada, July 2019.

[18] A. Gągol, D. Leśniak, and D. Straszak, "Aleph: efficient atomic broadcast in asynchronous networks with byzantine nodes," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pp. 214–228M. Świętek, Zurich, Switzerland, October 2019.

[19] D. Malkhi, K. Nayak, and L. Ren, "Flexible Byzantine Fault tolerance," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1041–1053, London, UK, November 2019.

[20] Z. Xiang, D. Malkhi, K. Nayak, and L. Ren, "Strengthened Fault Tolerance in Byzantine Fault Tolerant replication," 2021, https://arxiv.org/abs/2101.03715.

[21] J. Martin and L. Alvisi, "Fast byzantine consensus," *IEEE Transactions on Dependable and Secure Computing*, vol. 3, no. 3, pp. 202–215, 2006.

[22] I. Abraham, K. Nayak, L. Ren, and Z. Xiang, "Brief Announcement: Byzantine Agreement, Broadcast and State Machine Replication with Optimal Good-Case Latency," in *Proceedings of the 34th International Symposium on Distributed Computing (DISC 2020)*, Schloss Dagstuhl-Leibniz-Zentrum für Informatik, Research institute in Wadern, Germany, October 2020.

[23] I. Abraham, K. Nayak, L. Ren, and Z. Xiang, "Good-case Latency of Byzantine Broadcast: A Complete Categorization," 2021, https://arxiv.org/abs/2102.07240.

[24] O. Naor, M. Baudet, D. Malkhi et al., "Cogsworth: Byzantine view synchronization," 2019, https://arxiv.org/abs/1909.05204.

[25] M. Baudet, A. Ching, A. Chursin et al., "State machine replication in the libra blockchain," Technical Report D, 2019, http://libra.axuer.com/docs/state-machine-replication-paper/.

[26] O. Naor and I. Keidar, "Expected Linear Round Synchronization: The Missing Link for Linear Byzantine SMR," 2020, https://arxiv.org/abs/2002.07539.

[27] V. Shoup, "Practical threshold signatures," in *Proceedings of the Advances in Cryptology - EUROCRYPT 2000*, pp. 207–220, Zagreb, Croatia, May 2000.

[28] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme," in *Proceedings of the International Workshop on Public Key Cryptography*, pp. 31–46, Springer, Miami, FL, USA, January 2003.

[29] M. J. Fischer, N. A. Lynch, and M. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM*, vol. 32, no. 2, pp. 1–7, 1983.