Chasing the Speed of Light: Low-Latency Planetary-Scale Adaptive Byzantine Consensus

Christian Berger[†] Lívio Rodrigues[⋆] Hans P. Reiser[‡] Vinícius Cogo[⋆] Alysson Bessani[⋆]

[⋆]*LASIGE, Faculdade de Ciências Universidade de Lisboa, Portugal*
[†]*University of Passau, Germany* [‡]*Reykjavik University, Iceland*

Abstract

Blockchain technology has sparked renewed interest in planetary-scale Byzantine fault-tolerant (BFT) state machine replication (SMR). While recent works have mainly focused on improving the scalability and throughput of these protocols, few have addressed latency. We present FLASHCONSENSUS, a novel transformation for optimizing the latency of quorum-based BFT consensus protocols. FLASHCONSENSUS uses an adaptive resilience threshold that enables faster transaction ordering when the system contains few faulty replicas. Our construction exploits *adaptive weighted replication* to automatically assign high voting power to the fastest replicas, forming small quorums that significantly speed up consensus. Even when using such quorums with a smaller resilience threshold, FLASHCONSENSUS still satisfies the standard SMR safety and liveness guarantees with optimal resilience, thanks to the judicious integration of *abortable SMR* and *BFT forensics* techniques. Our experiments with tens of replicas spread in all continents show that FLASHCONSENSUS can order transactions with finality in less than 0.4s, half the time of a PBFT-like protocol (with optimal consensus latency) in the same network, and matching the latency of this protocol running on the theoretically best possible internet links (transmitting at 67% of the speed of light).

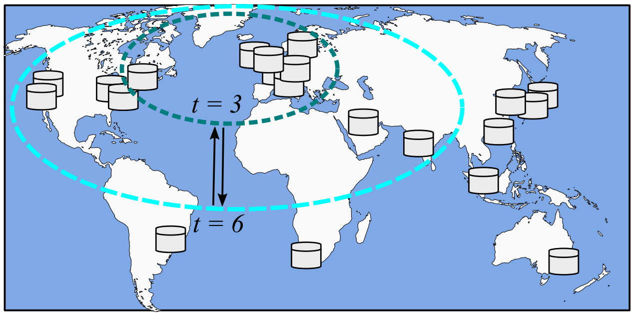
1 Introduction

State machine replication (SMR) is a general approach for achieving fault tolerance in distributed systems by coordinating client interactions with a set of n independent server replicas [48]. As of recently, many scalable (BFT) SMR protocols have been proposed for usage in blockchain infrastructures, such as HotStuff [60], SBFT [30], Tendermint [15], Mir-BFT [55], RedBellyBC [21], DisperseLedger [59], and Kauri [43]. These protocols employ either some dynamically elected leader [15, 30, 43, 56, 60], use multiple leaders [1, 55], or are leaderless [4, 21, 59].

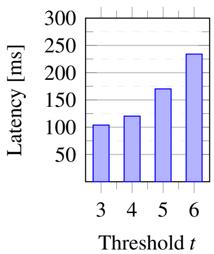
Nevertheless, consensus in all these cases requires communication steps that involve a Byzantine majority of replicas under the assumption of a Byzantine adversary that controls up to a *fixed* resilience threshold of $t = \lfloor \frac{n-1}{3} \rfloor$ replicas. Often, the quorum size for proceeding to the next protocol stage depends on this threshold, a Byzantine *t-dissemination quorum* with $\lceil \frac{n+t+1}{2} \rceil$ replicas [39]. This size equals roughly $\frac{2}{3}$ of all replicas if an *optimal* resilience threshold is used.

As for geo-replicated or planetary-scale systems, such as permissioned blockchains (e.g., [3, 21]) with tens of nodes distributed worldwide, a relevant optimization goal is lowering the end-to-end latency clients observe. Employing smaller quorums of closer replicas can significantly decrease SMR latency [34, 54]. The challenge in using such smaller, faster quorums is to ensure they intersect in sufficiently many replicas with all other quorums of the system. Such smaller, intersecting quorums can be built using *weighted replication*, where faster replicas have more voting power. However, this approach requires more replicas than necessary for optimal resilience [54]. In fact, there is a trade-off between resilience and performance, as a smaller, faster quorum requires more spare replicas [9].

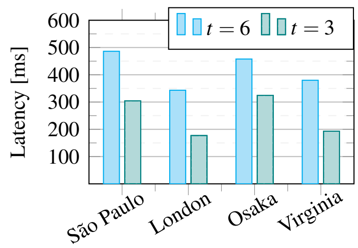
Smaller quorums for better latency. To illustrate how a geo-replicated system can progress faster by accessing a smaller quorum of replicas, we consider a weighted system [54] with $n = 21$ replicas dispersed across all 21 AWS regions (see Figure 1a). When the system is configured for maximum resilience, it tolerates up to $t = 6$ Byzantine replicas (the highest integer satisfying $t < \frac{n}{3}$) and has $\Delta = 2$ spare replicas, while the smallest weighted consensus quorum Q_v^6 contains 13 replicas (see §B for details on these calculations). This number corresponds to only one replica less than using non-weighted replication. If we instead configure the system for tolerating just $t = 3$ failures, the smallest weighted quorum Q_v^3 contains only 7 replicas, with $\Delta = 11$. Furthermore, this quorum can be composed of closer replicas that can exchange votes with each other faster, thus more swiftly proceeding through the stages of the consensus protocol (see Figure 1b), and ultimately leading to latency gains that clients around the globe can benefit from (see Figure 1c).



(a) Weighted quorums sizes with $t = 6$ and $t = 3$.



(b) Consensus latency vs. resilience threshold.



(c) End-to-end transaction latencies observed by clients in different regions.

Figure 1: Weighted quorums composition and resulting BFT SMR latency for different resilience thresholds (t) in our $n = 21$ setup (see details in §6).

In this paper, we propose FLASHCONSENSUS, an adaptive transformation for building protocols that aim for the best of both worlds. It ensures the same resilience as if the optimal threshold $t = \lfloor \frac{n-1}{3} \rfloor$ is used while driving consensus instances faster by optimizing the system for the expected common case, in which the number of faulty replicas is much lower than the BFT protocol's resilience threshold.

Challenges and the big picture. The main problem when using a lower resilience bound $t_{fast} < t$ for consensus is that a Byzantine attacker that controls f replicas (such that $t_{fast} < f \leq t$) can cause equivocations. In this case, the attacker can convince two correct replicas to decide different batches of transactions for the same consensus instance, as quorums for the lower threshold t_{fast} do not necessarily overlap in at least one correct replica.

A key insight of our work is that we can utilize *BFT protocol forensics or accountability* [49,50] in a novel way. Instead of using it as a forensic tool to investigate the "day after", we leverage it as a protective countermeasure against Byzantine attackers. In BFT protocol forensics, the client is the one who detects conflicting values based on replicas' logged signed messages and pinpoints equivocating replicas. In our solution, the responsibility of detecting faulty replicas is imposed on all correct replicas so the system can autonomously detect and expel equivocating parties. In a system tolerating up to t_{fast} faulty replicas, audits can detect agreement violations and identify $t_{fast} + 1$ faulty replicas if there are no more than $2t_{fast}$ faulty replicas [50]. Using $t_{fast} = \lceil \frac{t}{2} \rceil$ guarantees that audits are always supported for up to t faulty replicas. The system can recover from these violations by purging the detected violators from the system and rolling back the divergent decisions of correct replicas to a consistent state. This continuous auditing is important not only as a recovery mechanism but also as a *deterrent to attacks* since any perpetrator will be identified and expelled from the system.

The fact that a decision can be rolled back on replicas may lead to transaction outcomes observed by clients being undone, affecting the finality/durability of these transactions. Therefore, we have to modify the matching replies requirements on clients to ensure they can know when an operation *completes* in the system, ensuring linearizability as in standard SMR [16]. Another important contribution of our work is deriving the exact number of matching replies a client needs to expect to preserve linearizability even when consensus agreement violations are possible.

Minimizing consensus latency, but then letting clients wait for *more-than-usual* replies from all over the world to preserve linearizability counteracts our goal of reducing the end-to-end request latency. For this reason, we extend the programming model of BFT SMR with *correctables* that allow client applications to use incremental consistency guarantees [29], which simplifies and abstracts client-side speculation. We show that clients can lower their transaction latency even further using this abstraction.

Lastly, we need to take *liveness* into special consideration. SMR liveness requires requests issued by correct clients to be eventually completed. This property can be endangered if the protocol operates with a lower threshold $t_{fast} < t$ and there are $f > t_{fast}$ Byzantine replicas that stay silent, i.e., do not reply to the client or participate in consensus quorums. To avoid this scenario, we reuse the idea of *abortable state machine replication* [5]. If the system blocks or equivocates, we abort the execution of the fast mode of FLASHCONSENSUS and start a more resilient protocol instance, which uses the maximum resilience threshold t.

Our experimental evaluation with up to 51 replicas around the globe shows that FLASHCONSENSUS can order transactions with finality in less than 0.4s, which is half of the time required for BFT-SMART [11] (which implements a PBFT-like protocol) in the same network. Interestingly, our observed latencies are close to the theoretical optimum for BFT-SMART, considering the physical location of replicas and links transmitting at $\frac{2}{3}$ of the speed of light, which is accepted as the upper bound on data transmission speed for the internet [13,36].

Contributions. In this paper, we present a threshold-adaptive BFT protocol, FLASHCONSENSUS, that strives for continuous self-optimization during runtime by tuning the threshold used in consensus quorums. FLASHCONSENSUS was implemented on top of AWARE [9], a BFT-SMART extension that provides automated and dynamic voting-weight tuning and leader placement for supporting the emergence of fast quorum systems with a fixed resilience threshold t. Our ultimate goal is to significantly reduce latency in planetary-scale BFT SMR in the expected common case of having few failures. In summary, we make the following contributions:

- We explore how to detect malicious behavior under an underestimated threshold t_{fast} by auditing the system and repairing the correct replicas' state after an agreement violation happens.
- We show that it is possible to preserve the usual SMR guarantees, *linearizability* and *termination*, under the larger resilience threshold t, even if the agreement quorums are formed using a smaller threshold $t_{fast} < t$.
- We equip clients with *correctables* to allow for client-side speculation, thus enabling a client application to minimize the observed transaction latency even further by selecting the desired consistency level.
- We conduct an extensive evaluation based on real and simulated networks to reason about possible latency gains that clients dispersed over the globe can achieve when using FLASHCONSENSUS.
- We show that the principles underlying FLASHCONSENSUS are generic enough to be used in other quorum-based protocols such as HotStuff [60], leading to even higher latency gains.

2 Byzantine State Machine Replication

Assuming a deterministic service starting in the same state [48], a BFT (or Byzantine) SMR service aims to satisfy the following standard properties [8, 16]:

1. *SMR Safety:* all correct replicas *durably execute* the same sequence of operations;
2. *SMR Liveness:* all operations issued by correct clients are eventually completed.

PBFT. Castro and Liskov presented the Practical Byzantine Fault Tolerance (PBFT) [16] SMR algorithm in 1999, which became widely famous as the first practical method for tolerating Byzantine faults. PBFT is optimal in terms of best-case latency (3 communication steps), resilience ($t < \frac{n}{3}$), and in terms of assumptions (SMR safety under asynchrony and liveness in a partially synchronous system model).

The idea of PBFT is to order requests by relying on a single leader that assigns sequence numbers to batches of requests. If the leader is correct and there is enough synchrony in the system, PBFT's *normal case operation* is repeatedly successfully executed. The normal case operation is an agreement pattern that consists of the leader proposing a batch of operations to all replicas, followed by two phases of all-to-all message exchanges (PREPARE and COMMIT) in which replicas try to commit the messages consistently despite Byzantine failures by the use of quorums. Quorums are sufficiently large to guarantee that any intersection of two quorums Q_i and Q_j contains at least one correct replica, e.g., for any pair of replicas r_i and r_j, it holds for the quorums Q_i and Q_j they observe in a phase, that $|Q_i \cap Q_j| \geq t+1$.

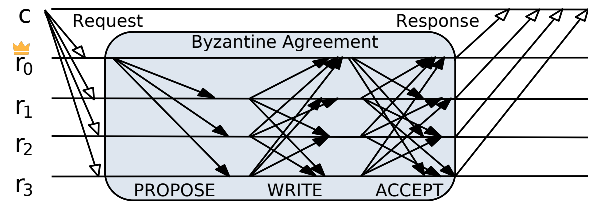


Figure 2: BFT-SMART normal case operation [11].

The problem of a faulty leader is solved by the *view change* sub-protocol if $t+1$ replicas support the change of leader. During a view change, the newly elected leader collects the current status from a quorum of replicas and defines consistent decisions for pending instances.

BFT-SMART. The most popular implementation of a PBFT-like protocol is the BFT-SMART replication library [11]. BFT-SMART works just like PBFT, running a three-phase *consensus instance* for defining the batch of transactions to be processed at slot i, as illustrated in Figure 2. The analogous of a view change in BFT-SMART is its *synchronization phase*, where a new *regency* (view) starts with a newly-elected leader responsible for concluding pending consensus instances and starting instances for future slots.

BFT-SMART was recently extended to support weighted geo-replication. The resulting system, AWARE [9], supports the autonomic reconfiguration of replicas weights based on the observed latency between them (see more about AWARE in Appendix B).

Abortable state machine replication. Abstract [5] is a design methodology to simplify the development and reconfiguration of *abortable* replicated state machines, which are, in contrast to traditional replicated state machines, allowed to abort the execution of client requests, and then re-direct this responsibility to another Abstract instance.

The design is flexible since the decision of what might be the "common case", i.e., the criteria for an optimization (such as speculating on the absence of faulty replicas or request contention) is left to the system designer. The only aspect that needs to be ensured is the safety of operations, while liveness is ensured by eventually falling back to a backup instance that never aborts, e.g., PBFT. Novel Abstract instances can be composed by linking existing ones, i.e., specifying which one takes over if the other aborts. This method is implemented by a *switching mechanism* that glues together the different Abstract instances. To make this work, each Abstract instance – except the backup – needs to implement an abort sub-protocol that is exposed to the next Abstract instance. In this way, the next Abstract instance can be safely initialized by utilizing the abort indications of the last aborting Abstract instance.

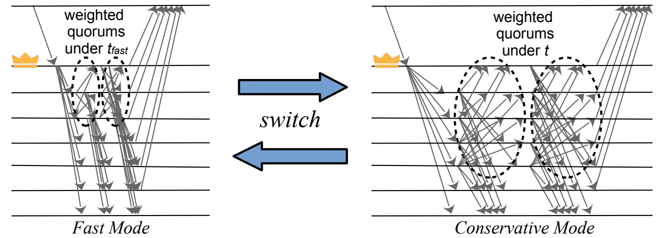


Figure 3: FLASHCONSENSUS two modes of operation.

3 Assumptions and Goals

We consider the same system model used in BFT-SMART [11] and in several other important protocols such as HotStuff [60] and PBFT [16]. In our system, there is a set of n replicas, $\mathcal{R} = \{r_0, .., r_{n-1}\}$, and an arbitrarily large set of clients $\mathcal{C}$.

Communication between all processes in the system is done through point-to-point channels that are authenticated and reliable. Reliable communication channels can be built on top of unreliable channels, as long as they are fair-loss, i.e., eventually, deliver messages if re-transmission attempts are used. To ensure liveness, we require a weak *partial synchrony* [23] where the system may initially behave asynchronously and, after an *unknown* GST (*Global Stabilization Time*), some upper bound holds for all message transmission delays.

We assume there are at most t Byzantine replicas and an unbounded number of Byzantine clients. Byzantine processes may behave arbitrarily and even collude under the orchestration of an adversary. However, they are still limited in their computational capabilities, i.e., they can not break cryptographic primitives.

We assume a trusted setup in which public key material is distributed among the processes. Every replica or client possesses secret and public keys, and all public keys are known to all processes. Additionally, replicas are equipped with `sign(·)` and `verify(·)` primitives to create and verify message signatures, respectively. Finally, we assume a collision-resistant hash function `hash(·)`.

Our goal is to design a fast BFT protocol for wide-area deployments that satisfy the standard SMR safety and liveness (see §2) for the optimal resilience bound $t = \lfloor \frac{n-1}{3} \rfloor$ and can tune itself to achieve fast commit latency in the expected common-case when there are no more than $t_{fast} = \lceil \frac{t}{2} \rceil$ faulty replicas and a stable, correct leader.

4 FLASHCONSENSUS

FLASHCONSENSUS provides continuous self-optimization at runtime by adapting the resilience threshold and changing weights to enable the emergence of smaller consensus quorums for low-latency transaction execution. To achieve this, it aims for the best of both worlds since it maintains the maximum resilience of the system for supporting diagnosis and repairs while continuously attempting to run consensus instances faster by optimizing the system for the expected common case with few or no failures.

This dual approach is implemented using abortable state machines, as previously done for performance [5] and resource efficiency [22]. The system starts in *conservative mode* by running instances of a quorum-based consensus protocol tolerating t failures. If nothing bad happens for a certain number of consensus instances, the system switches to the optimistic *fast mode* that tolerates only t_{fast} failures. While the leader is correct and the number of actual failures does not surpass t_{fast}, FLASHCONSENSUS stays in this configuration and uses smaller quorums to accelerate the termination of consensus. If latency gains do not match expectations, the leader is found to be faulty, or if correct replicas detect equivocations, FLASHCONSENSUS switches back to the conservative mode. Figure 3 illustrates the two modes of operation.

Even building upon the features of AWARE [9], allowing us the use small quorums with reconfigurable weights based on the observed latency between replicas, FLASHCONSENSUS design encompasses several challenges. First and foremost, we need mechanisms to detect and diagnose the system when there are more than $f > t_{fast}$ failures. Second, we need a robust reconfiguration mechanism to safely abort the fast configuration and move to the conservative one in such situations. Finally, the client-replica contract needs to ensure linearizability in our dual fault-threshold approach. These issues are discussed in the following subsections.

4.1 Obtaining Smaller Quorums

Decreasing t brings modest benefits in terms of quorum size when using normal dissemination quorum systems. For instance, in a system with 21 replicas, tolerating at most 6 failures, if we configure the system to tolerate up to $t_{fast} = 3$ failures, the quorum size will drop from 14 to 13 replicas. We can do much better than that by resorting to weighted replication, allowing us to build quorums as small as 7 replicas in this setting ($n = 21$, $t_{fast} = 3$).

The challenge in using weighted replication is to distribute voting power to enable small quorums without giving up required quorum guarantees, i.e., all quorums intersect in at least one correct server (consistency), and there is always at least one quorum available (availability).

The weight distribution scheme proposed for WHEAT [54] satisfies these requirements and ensures there is at least one minimal quorum with only $2t_{fast}+1$ replicas, independently of n. To see why this quorum is minimal, notice that the availability property requires the biggest quorum in the system to contain at most $n-t_{fast}$ replicas. To satisfy consistency, this quorum must intersect all other quorums in at least $t_{fast}+1$ replicas. As a consequence, the smallest possible quorum must have at least $2t_{fast}+1$ replicas. WHEAT's weight choice for the replicas is nonarbitrary. There are $2t_{fast}$ replicas with voting power $V_{max}>1$ and the rest remains with voting power 1. Appendix B contains more details about WHEAT.

An additional challenge in using weighted replication is how to (re)assign the weights in accordance with the current system conditions. In our case, we must select the $2t_{fast}$ replicas that will receive maximal voting power. We resort to the latency measurement and weight reassignment scheme of AWARE [9] and focus instead on the problem of running the protocol optimistically, tolerating few failures.

4.2 Dealing with $f>t_{fast}$ Failures

The key challenge in devising FLASHCONSENSUS is ensuring SMR safety and liveness when the system is in fast mode and $f>t_{fast}$. In this section, we detail how to detect and deal with such situations.

4.2.1 Safety

When running in fast mode, the adversary can control more than t_{fast} replicas and cause equivocations in the system. This situation might lead correct replicas to decide different transaction batches in a consensus instance since fast mode's smaller quorums are not guaranteed to overlap in at least one correct replica.

To limit this scenario, the system must detect if the actual number of faults (f) surpassed the resilience threshold of the fast mode (t_{fast}) and, if so, revert the system to its more resilient configuration. For detecting faults, we need to check the state of the replicas periodically to ensure they are consistent, which is done through periodic *checkpoint messages*, as in PBFT [16]. After every k completed consensus instances, each replica takes a snapshot of the service state and broadcasts to all replicas a signed message containing the digest of this snapshot h and the highest consensus instance i for which decided requests affect this checkpoint. Every replica waits for $n-t$ matching checkpoint hashes for the same consensus instance to define the checkpoint as *stable*. If during this process, a correct replica (the auditor) detects non-matching checkpoints, it runs the lightweight forensics procedure of Figure 4 to identify and obtain a non-repudiable *Proof-of-Culpability* (PoC) for the protocol violators.

Replica

F1. Find evidence: Let S and S' be the two sets of replicas with diverging checkpoint digests. The auditor tries to collect signed lists of decision proofs from consensus instances $i-k+1$ to i from at least one of the replicas of each of these two sets.

F2. Produce PoC: When such logs are obtained, the auditor checks the logs to find the first consensus instance with diverging decisions. Once such an instance is found, the auditor checks the proofs of decisions.

1. If any of the proofs of decision is invalid, the log signed by the replica that provided it is a PoC for the replica.
2. If both proofs are valid, the auditor finds the $t_{fast}+1$ replicas that provided signed ACCEPT messages for both decisions. These two conflicting proofs are the PoC for the $t_{fast}+1$ misbehaving replicas.

Figure 4: Lightweight forensics procedure.

If $n-t$ matching checkpoint hashes are received during the procedure execution, the auditor stops it. This action is important to avoid faulty replicas blocking correct replicas in forensics procedures since a faulty replica can send a non-matching checkpoint but never send the corresponding log.

After concluding the lightweight forensics procedure, if a PoC is produced for one or more replicas, the auditing replica broadcasts this PoC to all replicas making the system switch to the conservative mode and expel the misbehaving replicas, as described in §4.2.4.

The lightweight forensics protocol might also be triggered if a client detects non-matching signed replies for one of its requests. When this happens, the client sends a *panic message* with conflicting replies to the replicas. A replica that receives a panic message with correctly signed conflicting replies starts the lightweight forensics protocol, but fetches logs from the last checkpoint until the consensus instance that decided the problematic request.

4.2.2 Liveness

Besides equivocations, $f>t_{fast}$ adversary-controlled replicas can stay silent and negatively affect the liveness of the system. In such situations, there will be less than $n-t_{fast}$ correct replicas in the system, subverting a key liveness assumption of the consensus protocol operating with no more than t_{fast} failures. It can lead to two unfavorable situations. First, client requests might not be ordered, triggering a timeout and causing the system to initiate a leader change (i.e., BFT-SMART's synchronization phase). As explained in the next section, this sub-protocol reverts the system to the conservative configuration in which up to t failures are tolerated.

The second situation is more tricky since the request might be ordered but faulty replicas might not send replies to the client, preventing the client from consolidating the request result. In this case, the client could send a *panic* message to the replicas asking it to switch to the conservative mode. Replicas might trigger a leader change to switch the system's mode. However, it must be done carefully since a malicious client can create such a message to disallow the system to operate in fast mode. This type of weakness is inherent to many optimistic protocols [5]. Since the use of this mechanism can make FLASHCONSENSUS very fragile, as a single malicious client can undermine our latency improvements, we propose an alternative approach. If the client does not receive the required number of replies, it needs to periodically check the log of decisions until the next checkpoint to see in which position on the finalized request log its request appears. It is similar to what blockchain clients do, inspecting the blockchain until their requests are included in a block a certain number of blocks before the blockchain head. This approach ensures clients would benefit from the extremely low latency of FLASHCONSENSUS as long as there are no more than t_{fast} faulty replicas in the system.

4.2.3 Performance Degradation

To ensure FLASHCONSENSUS does not lead to performance degradation when compared to the performance of the conservative mode, all replicas periodically monitor their observed performance and compare it with expectations they have on the conservative mode. Replicas retrieve their expectations from AWARE's underlying latency prediction model [9]. Using this model, replicas can predict their consensus latency for the conservative mode using the network latency map and set their *consensus latency expectation threshold*. If they find that the consensus latency they currently observe exceeds this threshold, they stop their execution and ask for a synchronization phase. When $t+1$ replicas ask for a synchronization phase, the system switches to the conservative mode (see next section). In the end, FLASHCONSENSUS only runs in fast mode if replicas observe the consensus latency to be lower than the expected latency in the conservative mode.

4.2.4 Reconfiguration of the System

As explained before, FLASHCONSENSUS operates in two regimes: *fast*, in which smaller quorums are used and t_{fast} failures are tolerated, and *conservative*, in which standard-size quorums are employed and t failures are tolerated.

The system starts in the conservative mode, and after finishing a predefined number of θ consecutive consensus instances, it switches to the fast mode. Such reconfiguration is very simple because it is done deterministically at a certain point in the execution, i.e., after a certain consensus instance is decided. At this point, it simply requires each replica locally change the fault threshold to t_{fast} and recalculate its quorum size before executing the next consensus instance.

A replica stays in the fast configuration until the underlying consensus' synchronization phase [53] is triggered. This approach can be done deliberately due to either safety or liveness issues, as discussed in previous subsections. In both cases, we require the participation of $t+1$ replicas to start the synchronization phase, which always runs considering threshold t, not t_{fast} (which might already be violated).

During the synchronization phase, the newly elected leader will receive the log of decided instances since the last checkpoint from $n-t$ replicas and will verify if there are diverging decisions, using the lightweight forensics procedure (Figure 4). If this is the case, when a new leader is elected, the first transaction of its regency after repairing the system to a single transaction history is a reconfiguration request [11] asking for the removal of the Byzantine replicas that participated in the detected equivocation. This request contains the PoC generated during the forensics procedure. Every correct replica removes these compromised replicas from the system after processing reconfiguration requests with valid PoCs.

Further, in the case of equivocations, some replicas might need to roll back their states to a previous *stable checkpoint* (as discussed in §4.2.1), reapplying the correct transaction history as defined by the new leader on this state.

4.3 Ensuring Linearizability

In typical BFT SMR systems, a client waits for $t+1$ matching replies to ensure the replicated system perfectly emulates a centralized server, i.e., it satisfies linearizability [31]. This quorum size becomes $\lceil \frac{n+t+1}{2} \rceil$ if one wants to avoid running a consensus when performing read-only operations [8, 16]. These quorum sizes are still valid in FLASHCONSENSUS while in conservative mode; however, when the system is in the fast mode, the existence of equivocations and the possibility of divergent decisions (that will be later detected and punished) requires revisiting the matching replies quorum expected by clients.

Figure 5 illustrates the situation where two clients received conflicting replies (v and v') from two different quorums (Q and Q', respectively) for some instance i (ignore the leader change quorum for now). Even if t malicious replicas are present in the intersection of the quorums, a client can assume that its request has been committed and will not be rolled back by waiting for $n-t$ matching replies. Due to the $n>3t$ assumption, $(n-t)+(n-t)>n+t$. It means that two quorums with $n-t$ elements intersect in more than t replicas and thus the intersection must contain not only faulty replicas. By waiting for $n-t$ matching replies, responses accepted by clients will never be rolled back, even with divergent decisions for consensus instance i, *as long as there are no leader changes*.

Now, consider the same scenario in which the replies for v were deemed final by the client, but there was a leader change, and the new leader needs to define the result of consensus instance i. In this scenario, the elected leader waits for $n-t$ replicas to inform their status as indicated in the quorum of Figure 5, receiving replies from every replica but t slow replicas that decided v.

In this setting, the decision for v will be preserved as long

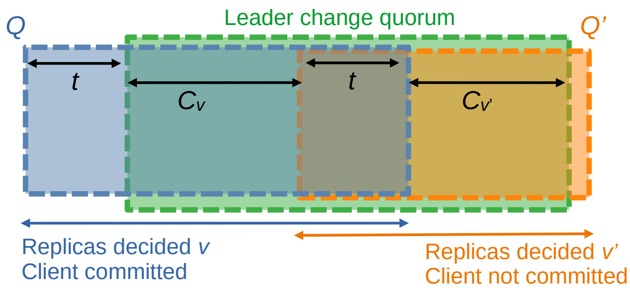


Figure 5: Quorum reasoning in FLASHCONSENSUS.

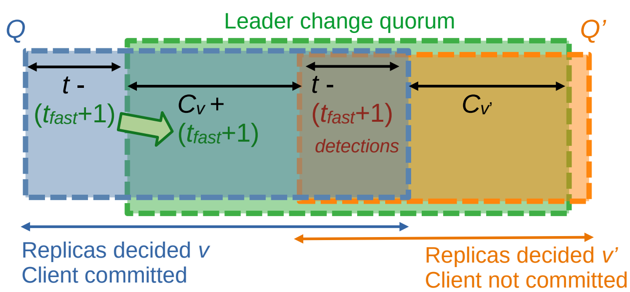


Figure 6: Waiting for information from $t_{fast}+1$ more replicas during a leader change in FLASHCONSENSUS.

as such value is the majority value among the ones informed by replicas, i.e., $C_v > C_{v'} + t$ in the figure. Considering $n = 2t + C_v + C_{v'}$ and $Q = C_v + 2t$ (both directly from the figure), we can reach that $C_v > \frac{n}{3}$, leading to $Q = n$. Therefore, *waiting for $n-t$ matching replies is insufficient to ensure a value is not rolled back during a leader change that switches the system to the conservative mode.* The only quorum big enough to ensure that is waiting for matching replies from *all replicas*.

Fortunately, by integrating BFT forensics [50] in the leader change sub-protocol (view change in PBFT or synchronization phase in BFT-SMART), we can make such a quorum smaller. More specifically, we observe that to produce equivocations that lead some correct replicas to decide v and v', and later force a committed value to be rolled back, the $t_{fast}+1$ equivocators must participate in the three quorums (for v, for v', and leader change). Therefore, if the new leader executes the forensics protocol during the leader change, it is possible to identify up to $d \leq t_{fast}+1$ equivocators. Consequently, it is possible to discard the contributions of such malicious replicas and wait for messages from d additional replicas. This situation is illustrated in Figure 6.

In this scenario, instead of assuming $C_v > C_{v'} + t$, we have $C_v + (t_{fast}+1) > C_{v'} + t - (t_{fast}+1)$. By developing this equation like before, we find that by *waiting for $n - t_{fast} - 1$ matching replies in fast mode, a client knows the result of its operation is durably committed, ensuring the linearizability of the replicated service*.

5 Implementation and Optimizations

FLASHCONSENSUS was implemented on top of the AWARE prototype [9] which is based on BFT-SMaRt. We stress that all mechanisms employed in AWARE (e.g., latency measurement and weights reassignment) could be implemented in any quorum-based SMR protocol. This implementation uses TLS to secure all communication channels and the elliptic curve digital signature algorithm (ECDSA) and SHA256 for signatures and hashes, respectively.

Most of our modifications were related to the switching between two modes of operations (with different resilience thresholds) and implementing BFT forensics, which requires the signing of WRITE and REPLY messages (see Figure 2).

5.1 Consolidated Algorithm

The consolidated algorithm for FLASHCONSENSUS is presented in Figure 7. This figure summarizes all the modifications we did on AWARE to accommodate the required mechanisms described in the previous section. The correctness argument of the protocol is presented in Appendix A.

5.2 Improving Latency with Speculation

Operating in fast mode requires clients to collect $n - t_{fast} - 1$ matching replies to preserve linearizability. It is considerably more than the $t+1$ replies typically required in BFT SMR. This result is expected to negatively impact the latency observed by clients in the fast mode.

FLASHCONSENSUS can further lower clients' observed latencies using client-side speculation. For this purpose, we implemented *correctables* [29] in the client shim of our BFT protocol. A correctable is a programming abstraction that allows a client application to work with incremental consistency guarantees and thus accelerates the application by allowing it to speculate with intermediate results. The state of a correctable can be updated multiple times, depending on the replies received by the client, strengthening the consistency guarantee each time, until it reaches the *final* state, which corresponds to the strongest consistency guarantee.

We design the FLASHCONSENSUS correctable following two principles: (1) to ensure the same safety guarantee as other BFT SMR protocols, the final consistency guarantee should satisfy linearizability under the maximum resilience threshold t, and (2) less safe consistency guarantees may relax either the assumptions on the number of Byzantine replicas or trade linearizability for a weaker consistency model.

We define incremental consistency levels for FLASHCONSENSUS as follows (see Figure 8). *First* is the speculative result a client can access as soon as the first response arrives and does not provide any correctness guarantee. *Weak* demands responses from replicas totaling $Q_v = t_{fast} \cdot V_{max} + 1$ votes, being V_{max} the maximum weight assigned to a replica, and

Client

C1. Invocation: When a new operation o is invoked, send $\langle \text{REQUEST}, o \rangle$ to all replicas.

C2. Acceptance: Accept a result *res* for o if received a set of matching replies $rep = \{\langle \text{REPLY}, h(o), fast, res \rangle\}$ such that:

1. $fast \wedge |rep| \geq n - t_{fast} - 1$ OR
2. $\neg fast \wedge \sum_{r \in rep} weight(r) \geq 2t \cdot V_{max} + 1$.

C3. Panic: Send a message $\langle \text{PANIC}, o, rep \rangle$ to all replicas if the set of all received replies $rep = \{\langle \text{REPLY}, h(o), \texttt{true}, * \rangle\}$ contains diverging results for operation o.

Replica

State

fast	mode of operation	boolean	`false`
r	current regency	Set of replicas	`initRegency()`
chkp	last stable checkpoint	bytes	`null`

Building Blocks

AWARE-NC	normal case operation of AWARE (conservative)
AWARE-NCF	normal case operation of AWARE in fast mode
AWARE-SYNC	replaces leader, synchronizes decision log
AUDIT	lightweight forensics procedure of Figure 4

S1. Request Processing: Start a *timer* for each received client request. If leader, create a batch of requests and propose it using AWARE-NCF, if *fast*, or AWARE-NC, otherwise.

S2. Client Panic: If *fast* and received a message $\langle \text{PANIC}, o, rep \rangle$ with diverging signed replies for o in *rep* from a client, do **Forensics**.

S3. Periodic Checkpoint: When a snapshot of the service state $chkp'$ is created after processing consensus j, broadcast signed message $\langle \text{CHECKPOINT}, h(chkp'), j \rangle$. If $n - t$ matching checkpoint hashes $h(chkp')$ for j are received, update the last stable checkpoint *chkp* to $chkp'$. If there are no $n - t$ matching checkpoints, do **Forensics**.

S4. Forensics: Run the forensics protocol in AUDIT. If a proof of culpability *poc* is produced, broadcast $\langle \text{POC}, poc \rangle$.

S5. Switch: After deciding θ successful consensus instances in a row, set *fast* to `true` (the optimization interval θ is inherited from AWARE).

S6. Abort: Broadcast a STOP message to enter **Synch. Phase** if *fast* and either (1) a request *timer* expires, (2) a message $\langle \text{POC}, poc \rangle$ with a valid PoC is received from some replica, or (3) upon *consensus latency disappointment* (the fast mode is slower than the predicted latency for the conservative mode – see §4.2.3).

S7. Synch. Phase: Run the synch. phase as follows:

1. Upon receiving $t+1$ matching STOP messages, use AWARE-SYNC to replace the current leader and synchronize decision history. If *fast*, set *fast* to `false` and do **Forensics** (except if triggered due to **S6.B**).
2. In case a PoC is produced, all replicas roll back to *chkp* and use the decisions (with proofs) obtained from the new leader to re-execute decided operations. The new leader proposes $\langle \text{RECONFIGURE}, culprits, poc \rangle$ using AWARE-NC.
3. Upon deciding $\langle \text{RECONFIGURE}, culprits, poc \rangle$ proposed by the new leader, a replica verifies the *poc* using AUDIT, and removes the replicas *culprits* from the system.

Figure 7: A summary of FLASHCONSENSUS.

thus must have been confirmed by at least one correct replica if $f \leq t_{fast}$. This result can be stale, leading to the satisfaction of only sequential consistency under t_{fast} failures. *Strong* demands $Q_v = 2t_{fast} \cdot V_{max} + 1$ weights and satisfies linearizability if $f \leq t_{fast}$. Since the read-only optimization requires

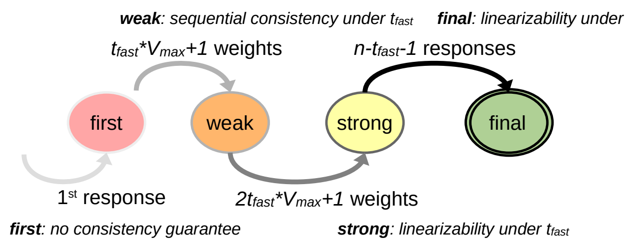


Figure 8: Incremental consistency levels that can be accessed through the correctable programming interface.

such a larger quorum, a correctable in a strong state ensures linearizability as long as $f \leq t_{fast}$. Finally, the *Final* level satisfies linearizability under t (just like any typical SMR with the read-only optimization enabled) by waiting for $n - t_{fast} - 1$ replies, as explained in the previous section. Since BFT SMR requires linearizability, only *Strong* and *Final* give the typical safety guarantee for their respective resilience thresholds.

6 Evaluation

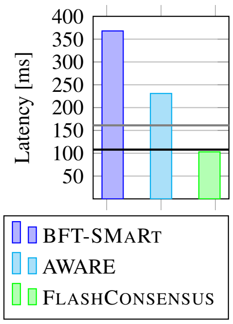
We use our prototype implementation to compare the latency of FLASHCONSENSUS with AWARE and BFT-SMART, on the AWS cloud as well as a simulated network of 51 replicas which is based on real data from the internet. Parts of our experiments were conducted in our local data center using high-fidelity tools for network emulation and simulation [28, 33]. We validated the fidelity of these emulated/simulated setups as described in Appendix C. Our experiments focus mostly on measuring latency, which is fundamentally limited by the quality of the links and quorum formation rules in a wide-area network.

6.1 AWS Network

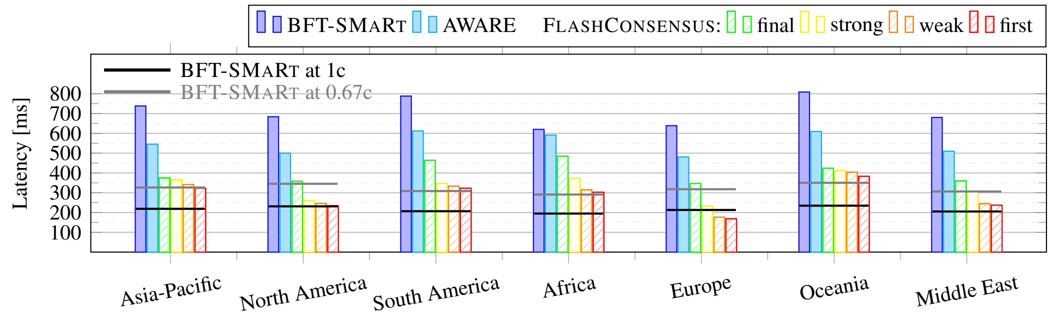
To begin with, we investigate the potential performance gains of FLASHCONSENSUS, comparing it to AWARE and BFT-SMART as baselines. Later, we reason about FLASHCONSENSUS's runtime behavior (particularly its adaptiveness) in the presence of faults or unfavorable network conditions.

Setup. We use `c5.xlarge` instances on the AWS cloud for deploying a client and a replica in each of the $n = 21$ AWS regions (depicted in Figure 1a).

All clients send 400-bytes requests simultaneously and continuously to the replicas (2000 per client) until each has finished its measurements. A client request arriving at the leader may wait until it gets included in a batch when there is currently a consensus instance running. We employ synchronous clients that block until a result is obtained and send the next request after randomly waiting for up to 1s. Finally, *request latency* is the average end-to-end protocol latency computed by a client after completing all operations.



(a) Consensus latency.



(b) Clients' observed end-to-end latencies for protocol runs with BFT-SMaRt, AWARE and FlashConsensus. The client results are averaged over all regions per continent.

Figure 9: Achievable latency gains for the $n = 21$ AWS setup.

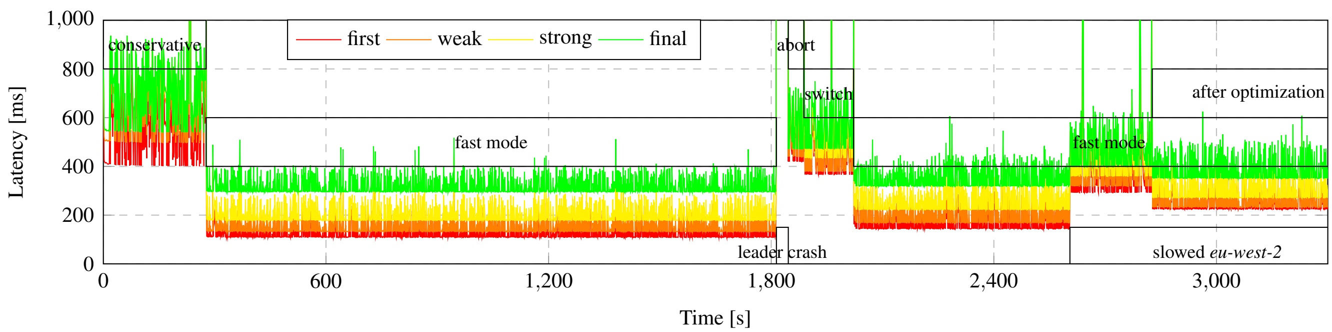


Figure 10: Runtime behavior of FlashConsensus.

6.1.1 FlashConsensus Acceleration

For a better exposition, we group the 21 clients' results by the continent they are located in, reporting only their regional averages (see Figure 9). We observe that FlashConsensus significantly accelerates consensus, leading to a speedup of 3.57× faster decisions (see Figure 9a). This result also surpasses the speedup of 2.29×, achievable if the speed of the links employed by BFT-SMaRt approximated the speed of light.[1] Second, accelerating consensus decisions also leads to faster request latencies observed by clients worldwide (see Figure 9b). Averaged over all client regions, FlashConsensus leads to a speedup of 1.87× over BFT-SMaRt for clients' observed end-to-end request latencies with finalization (AWARE leads to 1.33× only).

Our results also show that even higher speedups can be achieved by employing the incremental consistency levels of FlashConsensus's correctables. The *strong* consistency level, which guarantees linearizability if $f < t_{fast}$, achieves a speedup of 2.38×, while the speculative levels *weak* and *first* achieve speedups of 2.76× and 2.90×, respectively (results are averaged over all regions).

[1] Assuming all links transmit at the speed of 0.67c [13, 36].

6.1.2 FlashConsensus Runtime Behavior

For this experiment, we create an emulation of the AWS network in our local cluster to be more flexible with the induction of events. The emulated network uses latency statistics from cloudping,[2] and, the Kollaps network emulator, which was validated for realistic WAN experimentation with BFT-SMaRt and WHEAT [28], to construct the emulated network.

We launch FlashConsensus in the same $n = 21$ AWS regions to observe its runtime behavior during the system's lifespan. Noticeably, clients' request latencies show high variations, which are caused by a random waiting time of a request at the leader before getting included in the next batch, which takes a varying time depending on how shortly the request arrived before the next consensus can be started. Moreover, we induce the following events to evaluate FlashConsensus's reactions and plot the latency observed by a representative correct client in Figure 10.

Configuration switch. FlashConsensus starts in the conservative configuration, displaying low performance. Later, around time 277s, the system switches to the fast configuration (such switches are attempted every 400 consensus instances), leading to a significant latency improvement.

Leader crash. At the time 1814s, we crash the leader. Subse-

[2] https://www.cloudping.co/grid.

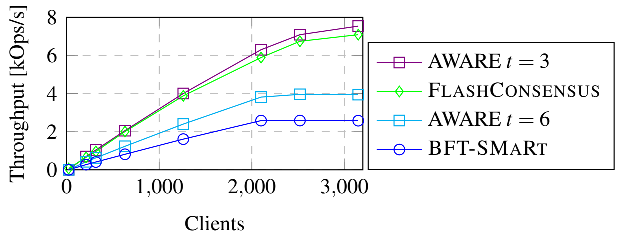


Figure 11: Throughput comparison for $n = 21$ replicas.

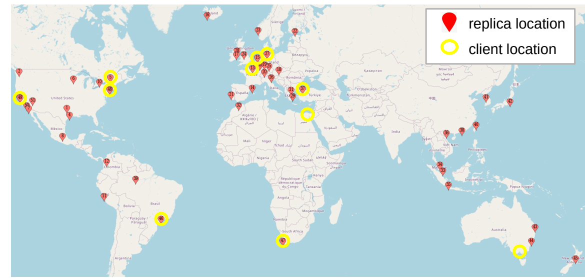


Figure 12: Map showing the locations of the $n = 51$ replicas.

quently, replicas perform a leader change and abort (at 1846s), switching back to the conservative mode. This blocking time is similar to what a client would experience if an equivocation is discovered. At the time 1889s, after finishing 400 consensus instances again, replicas return to the fast mode. This leads only to a modest performance improvement because the weights have not been optimized yet. At the time 2022s, the system self-optimizes to its best weight distribution and leader placement (using the mechanisms inherited from AWARE), reaching again the low latencies experienced before.

Slowed network. At the time 2640s, we artificially make the current leader slower by using the *tc* command. After approx. 185s, which is the time required for a measurement round and self-optimization [9], FLASHCONSENSUS detects it is running in a sub-optimal configuration and changes replicas' weights and leader location to accelerate performance (the leader is changed from *eu-west-2* to *us-east-1*).

6.1.3 FLASHCONSENSUS Throughput

Although FLASHCONSENSUS aims to optimize latency, we conducted a simple 0/0-microbenchmark in our emulated AWS network with an increasing number of clients evenly distributed among all AWS regions while measuring the throughput of BFT-SMART, AWARE ($t = 6$), AWARE ($t = 3$), and FLASHCONSENSUS. In this experiment, we used 0-byte requests to avoid the saturation of links bandwidth. Figure 11 presents the results. These results show two main insights. First, faster consensus instances can achieve higher throughput, provided the available network bandwidth is not exhausted. Second, FLASHCONSENSUS displays a minor penalty compared to AWARE ($t = 3$), which uses the same quorums but offers half of our resilience. This difference stems from the costs of additional signatures needed to integrate BFT forensics into our protocol.

6.2 Larger Network

In our next experiment, we assess if the performance gains of FLASHCONSENSUS are sustained in a different scenario, with a larger number of $n = 51$ replicas approaching a permissioned blockchain deployment. Since this number exceeds the number of available AWS regions, we sample locations from a publicly available dataset provided by *Wonderproxy*.[3] We choose 51 locations for distributing a replica in each of them, and also distribute 12 clients, as depicted in Figure 12.

Setup. For simulating this network, we use *Phantom* [33], which employs a hybrid simulation-emulation architecture, in which real, unmodified applications execute as native Linux processes that are hooked into a high-performance network simulation. Further, it has been shown that Phantom can be used to faithfully evaluate the performance of BFT protocol implementations [10]. For validity, we repeated the $n = 21$ AWS experiment depicted in the previous section in this simulator and observed a low deviation with the results obtained from the real AWS network and Kollaps emulator (see Appendix C).

Speedup achieved by FLASHCONSENSUS. In our experiment, we measure the speedup that FLASHCONSENSUS achieves in direct comparison with BFT-SMART. We measure both consensus latency and client end-to-end latencies using the same method described before. Phantom bootstraps replicas and clients in their host locations with the initial protocol leader in Cape Town. As before, clients run simultaneously and send requests with a payload of 400 bytes with a randomized waiting interval of up to 4s between two requests. When running FLASHCONSENSUS replicas are started in a configuration with optimal resilience threshold ($t = 16$), but FLASHCONSENSUS optimizes this threshold to $t_{fast} = 8$ before clients collect their measurement samples.

Results. Figure 13 shows similar latency improvements as in the previous experiment for FLASHCONSENSUS when compared with BFT-SMART. The consensus latency decreases from 350ms in BFT-SMART to only 88ms in FLASHCONSENSUS, corresponding to a consensus execution speedup of 3.98×. For request latencies with *final* consistency, the highest speedup observed was in Frankfurt (1.95× from 615ms down to 314ms), and the lowest speedup was observed in Cape Town (1.41× from 618ms to 440ms).

[3] https://wonderproxy.com/blog/a-day-in-the-life-of-the-internet/.

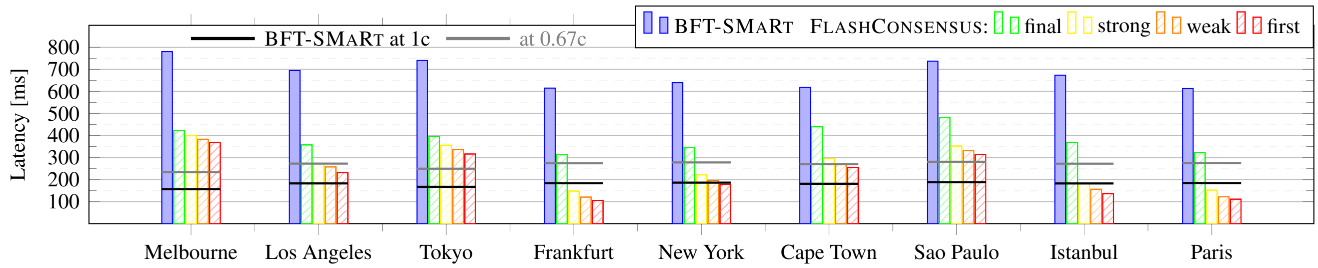


Figure 13: Latencies of BFT-SMaRt and FlashConsensus for $n = 51$ replicas, observed from different client locations.

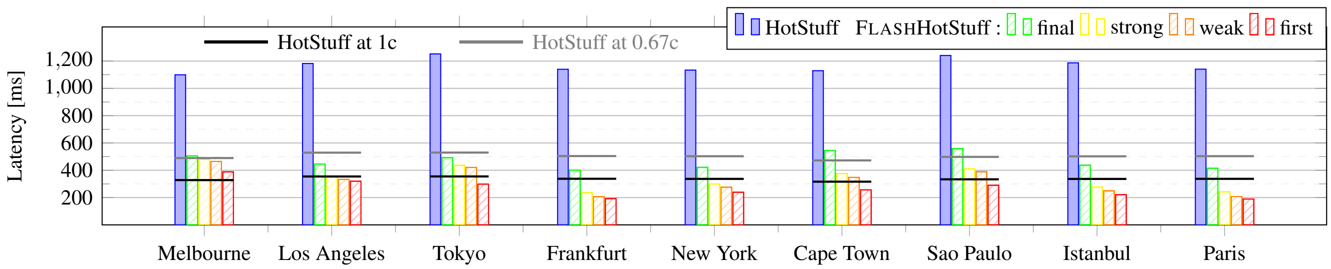


Figure 14: Latencies of HotStuff using FlashConsensus techniques for $n = 51$ replicas.

Further, the speedup increases when using the incremental consistency levels of the correctable. The average speedup across all client locations from BFT-SMaRt to FlashConsensus's *final* level is 1.83× and becomes incrementally higher for the speculative levels *strong* (2.70×), *weak* (2.99×) and *first* (3.23×). For comparison, the speedup that BFT-SMaRt would achieve if the speed of network links approximated the speed of light is roughly 2.5×.

These results show that using smaller weighted quorums is an important technique to achieve low consensus latency, achieving results that are competitive even with the expected latency of our baseline protocol with speed-of-light links.

6.3 FlashConsensus-flavored HotStuff

The principle of improving quorums introduced in FlashConsensus is general enough to be used in other BFT SMR protocols to decrease consensus latency. For example, our techniques can be directly applied to speedup agreement in multi-leader protocols [55], as long as the leaders are selected only among the best-connected replicas, or even in protocols providing additional guarantees such as fair ordering [61]. Here, we experimentally demonstrate this aspect by applying FlashConsensus ideas to HotStuff [60].

Compared to BFT-SMaRt and PBFT, HotStuff uses an agreement pattern with one additional phase and achieves a linear communication complexity by letting the leader collect and distribute quorum certificates in each phase. It results in 7 communication steps per consensus instance, instead of 3 as required by BFT-SMaRt/PBFT (see Figure 2). This design makes the overall system's latency even more sensitive to how fast agreement can be achieved, which depends on the speed a HotStuff leader can succeed in collecting quorum certificates. FlashHotStuff selects a configuration in which the leader can communicate fast with a set of well-connected replicas that are granted more voting power.

Setup. We use a prediction model of the HotStuff protocol[4] to simulate FlashHotStuff and normal HotStuff protocol runs on the same latency map of $n = 51$ replicas used before (Figure 12). These simulations compute the achievable latencies for different consistency levels of FlashHotStuff, the latency of the original HotStuff and the hypothetical latencies of HotStuff when network speeds are as fast as possible.

Results. Simulation results show that FlashHotStuff significantly improves HotStuff's latency (see Figure 14). The consensus latency is optimized from 853ms in HotStuff to only 177ms in FlashHotStuff, which corresponds to a consensus execution speedup of 4.82×, achieved by the incorporation of weights, optimal leader placement, and the use of smaller quorums. For client request latencies with *final* consistency, the highest speedup observed was in Frankfurt (2.84×) and lowest in Cape Town (2.07×).

Like before, the speedup increases with lower consistency levels. The average speedup across all client locations from HotStuff to FlashHotStuff's *final* level is 2.56× and becomes incrementally higher for the speculative levels *strong* (3.69×), *weak* (4.05×) and *first* (4.74×).

[4] This model is similar to the one AWARE uses to anticipate the effect of weight and leader changes during its self-optimization [9].

7 Related Work

In this section, we discuss various optimizations proposed for BFT SMR, with a focus on those that address changing conditions, geographical distribution of replicas, and low latency. Additionally, we cover related work on BFT forensics.

Adaptivity in BFT SMR. Making BFT protocols adaptive to their environment has been studied in multiple works [6, 7, 14, 17, 24, 38, 45, 51]. RBFT monitors system performance under redundant leaders to assert that a faulty leader cannot purposely degrade performance [6]. Optimizing the leader selection has been studied in several works (e.g., [24, 38]). Further works investigated adaptively switching the consensus algorithm [7, 14], strengthening the protocol by reacting to perceived threat level changes [51], being network-agnostic (tolerating a higher threshold in synchronous networks) [12] or adapting the state transfer strategy to the available network bandwidth [17]. In contrast, FLASHCONSENSUS can be applied to most BFT protocols and accelerates planetary-scale Byzantine consensus by making both system configuration (leader and replica weights) *and* threshold adaptive without impacting resilience through the integration of BFT forensics.

Geographically-distributed SMR. Various works studied the improvement of SMR for WANs [2, 20, 25, 26, 40, 41, 44, 46, 57–59]. Mencius, one of the first of these works, optimizes performance in WANs using a rotating leader scheme that allows clients to pick their geographically closest replica as its leader [40]. While Mencius tolerates only crash faults, EBAWA uses the same rotating leader technique together with trusted components on each replica to tolerate a minority of Byzantine replicas in a protocol that requires the same number of communication steps as Mencius [57]. Steward proposes a hierarchical, two-layered architecture for replication. Regional groups within a system site run Byzantine agreement, and these replication groups are finally connected over a CFT protocol [2]. Steward's idea of a hierarchical architecture was later employed in Fireplug [44] for efficient geo-replication of graph databases by composing multiple BFT-SMaRt groups. PBFT-CS refines PBFT with client-side speculation. Clients send subsequent requests after predicting a response to an earlier request without waiting for the earlier request to commit – however, clients need to track and propagate the dependencies between requests [58]. WHEAT optimizes BFT SMR latency by incorporating weighted replication and tentative executions [54], while AWARE enriches WHEAT through self-monitoring capabilities and dynamic optimization by adjusting weights and leader position [9]. By utilizing BFT forensics, FLASHCONSENSUS can safely use smaller consensus quorums and accelerate Byzantine consensus even further than AWARE, mastering the resilience-performance trade-off that previously limited AWARE's performance.

Fast BFT. It has been shown that having additional redundancy (and using a less-than-optimal resilience threshold) can be efficiently utilized to develop faster consensus variants, i.e., two-step Byzantine consensus [32, 37, 42], or even one-step asynchronous Byzantine consensus for scenarios that are contention-free [27, 52]. FLASHCONSENSUS extends a latency-optimal protocol similar to PBFT to significantly improve latency without losing resilience.

BFT forensics. BFT forensics is a technique for analyzing safety violations in BFT protocols after they happened [50], yielding results such as that at least $t+1$ culprits can be identified in case of an equivocation (with the accountability of up to $\frac{2n}{3}$ replicas that may actually be Byzantine). Polygraph is an accountable Byzantine consensus algorithm tailored for blockchain applications that allow the punishment of culprits (e.g., via stake slashing) in case of equivocations [18]. A simple transformation to obtain an accountable Byzantine consensus protocol from any Byzantine consensus protocol has been proposed in [19]. The Basilisc class of protocols solves consensus with $n \leq 3t+d+2q$ replicas tolerating t general Byzantine failures, d deceitful failures (that violate safety – the ones that can be identified using forensics), and q benign failures [47]. The Basilisc resilience is shown to be optimal. IA-CCF [49] shows that by logging all messages exchanged on PBFT in a blockchain, it is possible to identify any number of misbehaving replicas in case of equivocations. FLASHCONSENSUS does not differentiate failures or require a blockchain, employing instead a light version of the BFT forensics protocol of [50] to identify a limited number of equivocators in a fast regency (or view, in PBFT parlance), enabling the secure use of smaller quorums.

8 Conclusion

FLASHCONSENSUS accelerates planetary-scale Byzantine consensus by combining weighted replication with BFT forensics to safely underestimate the resilience threshold, using faster quorums to drive consensus decisions. We showed how to obtain FLASHCONSENSUS from AWARE, by incorporating the ideas of abortable state machine replication, BFT forensics, and incremental consistency guarantees. Notably, FLASHCONSENSUS always achieves linearizability and liveness under the optimal resilience threshold – even when agreement quorums are formed using the fast threshold.

Our evaluation results indicate that latency benefits are substantial, i.e., achieving a speedup of $1.87\times$ over BFT-SMART. Our methodology is essentially a transformation that can be applied to other BFT protocols improving their speed under geographical dispersion. We showed that, if a protocol's agreement patterns consist of more communication steps (like in HotStuff), it results in even stronger benefits ($2.56\times$ speedup). We also see potential in the client-side speculation, which allows the application to choose a relaxed consistency level from the client-replica contract on the granularity level of individual operations. It can be a good fit for operations that are time-sensitive and not security-critical as they can benefit from speedups of up to 6×.

Acknowledgements. This work was supported by FCT through the ThreatAdapt (FCT-FNR/0002/2018) and SMaRtChain (2022.08431.PTDC) projects, and the LASIGE Research Unit (UIDB/00408/2020 and UIDP/00408/2020). Further, this work has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) grant number 446811880 (BFT2Chain).

References

[1] Salem Alqahtani and Murat Demirbas. BigBFT: A multileader Byzantine fault tolerance protocol for high throughput. In *Proc. of the IEEE Int. Performance, Computing, and Communications Conference (IPCCC)*, pages 1–10, 2021.

[2] Yair Amir, Claudiu Danilov, Danny Dolev, Jonathan Kirsch, John Lane, Cristina Nita-Rotaru, Josh Olsen, and David Zage. Steward: Scaling Byzantine fault-tolerant replication to wide area networks. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 7(1):80–93, 2010.

[3] Elli Androulaki et al. Hyperledger Fabric: a distributed operating system for permissioned blockchains. In *Proc. of the 13th European Conference on Computer Systems (EuroSys)*, pages 30:1–30:15, 2018.

[4] Karolos Antoniadis, Antoine Desjardins, Vincent Gramoli, Rachid Guerraoui, and Igor Zablotchi. Leaderless consensus. In *Proc. of the 41st IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*, pages 392–402, 2021.

[5] Pierre-Louis Aublin, Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The next 700 BFT protocols. *ACM Transactions on Computer Systems (TOCS)*, 32(4):1–45, 2015.

[6] Pierre-Louis Aublin, Sonia Ben Mokhtar, and Vivien Quéma. RBFT: Redundant Byzantine fault tolerance. In *Proc. of the 33rd IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*, pages 297–306, 2013.

[7] Jean-Paul Bahsoun, Rachid Guerraoui, and Ali Shoker. Making BFT protocols really adaptive. In *Proc. of the 29th IEEE Int. Parallel and Distributed Processing Symp. (IPDPS)*, pages 904–913, 2015.

[8] Christian Berger, Hans P. Reiser, and Alysson Bessani. Making reads in BFT state machine replication fast, linearizable, and live. In *Proc. of the 40th IEEE Int. Symp. on Reliable Distributed Systems (SRDS)*, pages 1–12, 2021.

[9] Christian Berger, Hans P. Reiser, João Sousa, and Alysson Neves Bessani. AWARE: Adaptive wide-area replication for fast and resilient Byzantine consensus. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2020.

[10] Christian Berger, Sadok Ben Toumia, and Hans P. Reiser. Does my BFT protocol implementation scale? In *Proc. of the 3rd Int. Workshop on Distributed Infrastructure for Common Good (DICG)*, pages 1–6, 2022.

[11] Alysson Bessani, João Sousa, and Eduardo EP Alchieri. State machine replication for the masses with BFT-SMaRt. In *Proc. of the 44th Annu. IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, pages 355–362, 2014.

[12] Erica Blum, Jonathan Katz, and Julian Loss. Network-agnostic state machine replication. *preprint arXiv:2002.03437*, 2020.

[13] Frank Cangialosi, Dave Levin, and Neil Spring. Ting: Measuring and exploiting latencies between all Tor nodes. In *Proc. of the ACM Internet Measurement Conference (IMC)*, pages 289–302, 2015.

[14] Carlos Carvalho, Daniel Porto, Luís Rodrigues, Manuel Bravo, and Alysson Bessani. Dynamic adaptation of Byzantine consensus protocols. In *Proc. of the 33rd Annual ACM Symposium on Applied Computing (SAC)*, pages 411–418, 2018.

[15] Daniel Cason, Enrique Fynn, Nenad Milosevic, Zarko Milosevic, Ethan Buchman, and Fernando Pedone. The design, architecture and performance of the Tendermint blockchain network. In *Proc. of the 40th IEEE Int. Symp. on Reliable Distributed Systems (SRDS)*, pages 23–33, 2021.

[16] Miguel Castro and Barbara Liskov. Practical Byzantine fault tolerance. In *Proc. of the 3rd Symposium on Operating Systems Design and Implementation (OSDI)*, pages 173–186, 1999.

[17] Tairi Chiba, Ren Ohmura, and Junya Nakamura. Network bandwidth variation-adapted state transfer for geo-replicated state machines and its application to dynamic replica replacement. *preprint arXiv:2204.08656*, 2022.

[18] Pierre Civit, Seth Gilbert, and Vincent Gramoli. Polygraph: Accountable Byzantine agreement. In *Proc. of the 41st IEEE International Conference on Distributed Computing Systems (ICDCS)*, pages 403–413, 2021.

[19] Pierre Civit, Seth Gilbert, Vincent Gramoli, Rachid Guerraoui, and Jovan Komatovic. As easy as ABC: Optimal (A)ccountable (B)yzantine (C)onsensus is easy! In *Proc. of the 37th IEEE International Parallel and Distributed Processing Symposium (IPDPS)*, pages 560–570, 2022.

[20] Paulo Coelho and Fernando Pedone. Geographic state machine replication. In *Proc. of the 37th IEEE Int. Symp. on Reliable Distributed Systems (SRDS)*, pages 221–230, 2018.

[21] Tyler Crain, Christopher Natoli, and Vincent Gramoli. Red Belly: A secure, fair and scalable open blockchain. In *Proc. of the 42nd IEEE Symp. on Security and Privacy (S&P)*, pages 466–483, 2021.

[22] Tobias Distler, Christian Cachin, and Rüdiger Kapitza. Resource-efficient Byzantine fault tolerance. *IEEE Transactions on Computers (TC)*, 65(9):2807–2819, 2015.

[23] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.

[24] Michael Eischer and Tobias Distler. Latency-aware leader selection for geo-replicated Byzantine fault-tolerant systems. In *Proc. of the 1st Workshop on Byzantine Consensus and Resilient Blockchains (BCRB)*, pages 140–145, 2018.

[25] Michael Eischer and Tobias Distler. Resilient cloud-based replication with low latency. In *Proc. of the 21st International Middleware Conference*, pages 14–28, 2020.

[26] Michael Eischer, Benedikt Straßner, and Tobias Distler. Low-latency geo-replicated state machines with guaranteed writes. In *Proc. of the 7th Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC)*, pages 1–9. 2020.

[27] Roy Friedman, Achour Mostefaoui, and Michel Raynal. Simple and efficient oracle-based consensus protocols for asynchronous Byzantine systems. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 2(1):46–56, 2005.

[28] Paulo Gouveia, João Neves, Carlos Segarra, Luca Liechti, Shady Issa, Valerio Schiavoni, and Miguel Matos. Kollaps: decentralized and dynamic topology emulation. In *Proc. of the 15th European Conference on Computer Systems (EuroSys)*, pages 1–16, 2020.

[29] Rachid Guerraoui, Matej Pavlovic, and Dragos-Adrian Seredinschi. Incremental consistency guarantees for replicated objects. In *Proc. of the 12th USENIX Symp. on Operating Systems Design and Implementation (OSDI)*, pages 169–184, 2016.

[30] Guy Golan Gueta, Ittai Abraham, Shelly Grossman, Dahlia Malkhi, Benny Pinkas, Michael Reiter, Dragos-Adrian Seredinschi, Orr Tamir, and Alin Tomescu. SBFT: a scalable and decentralized trust infrastructure. In *Proc. of the 49th Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, pages 568–580, 2019.

[31] Maurice P. Herlihy and Jeannette M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(3):463–492, 1990.

[32] Heidi Howard, Aleksey Charapko, and Richard Mortier. Fast Flexible Paxos: Relaxing quorum intersection for Fast Paxos. In *Proc. of the 22nd Int. Conf. on Distributed Computing and Networking (ICDCN)*, pages 186–190, 2021.

[33] Rob Jansen, Jim Newsome, and Ryan Wails. Co-opting Linux processes for high-performance network simulation. In *Proc. of the USENIX Annual Technical Conference (USENIX ATC)*, pages 327–350, 2022.

[34] Flavio Junqueira, Yanhua Mao, and Keith Marzullo. Classic Paxos vs Fast Paxos: Caveat emptor. In *Proc. of the Workshop on Hot Topics in System Dependability*, 2007.

[35] Scott Kirkpatrick, Charles Daniel Gelatt, and Mario P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[36] Katharina Kohls and Claudia Diaz. Verloc: Verifiable localization in decentralized systems. In *Proc. of the 31st USENIX Security Symposium (USENIX Security)*, pages 2637–2654, 2022.

[37] Petr Kuznetsov, Andrei Tonkikh, and Yan X Zhang. Revisiting optimal resilience of fast Byzantine consensus. In *Proc. of the 40th ACM Symposium on Principles of Distributed Computing (PODC)*, pages 343–353, 2021.

[38] Shengyun Liu and Marko Vukolić. Leader set selection for low-latency geo-replicated state machine. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 28(7):1933–1946, 2017.

[39] Dahlia Malkhi and Michael Reiter. Byzantine quorum systems. *Distributed computing*, 11(4):203–213, 1998.

[40] Yanhua Mao, Flavio P. Junqueira, and Keith Marzullo. Mencius: Building efficient replicated state machines for WANs. In *Proc. of the 8th USENIX Conf. on Operating Systems Design and Implementation (OSDI)*, pages 369–384, 2008.

[41] Yanhua Mao, Flavio P. Junqueira, and Keith Marzullo. Towards low latency state machine replication for uncivil wide-area networks. In *Proc. of the 5th Workshop on Hot Topics in System Dependability (HotDep)*, 2009.

[42] Jean-Philippe Martin and Lorenzo Alvisi. Fast Byzantine consensus. *IEEE Transactions on Dependable and Secure Computing (TDSC)*, 3(3):202–215, 2006.

[43] Ray Neiheiser, Miguel Matos, and Luís Rodrigues. Kauri: Scalable BFT consensus with pipelined tree-based dissemination and aggregation. In *Proc. of the 28th ACM SIGOPS Symp. on Operating Systems Principles (SOSP)*, pages 35–48, 2021.

[44] Ray Neiheiser, Luciana Rech, Manuel Bravo, Luís Rodrigues, and Miguel Correia. Fireplug: Efficient and robust geo-replication of graph databases. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 31(8):1942–1953, 2020.

[45] Martin Nischwitz, Marko Esche, and Florian Tschorsch. Raising the AWAREness of BFT protocols for soaring network delays. In *Proc. of the 47th IEEE Conf. on Local Computer Networks (LCN)*, pages 387–390, 2022.

[46] Shota Numakura, Junya Nakamura, and Ren Ohmura. Evaluation and ranking of replica deployments in geographic state machine replication. In *Proc. of the 38th IEEE Int. Symp. on Reliable Distributed Systems Workshops (SRDSW)*, pages 37–42, 2019.

[47] Alejandro Ranchal-Pedrosa and Vincent Gramoli. Basilic: Resilient optimal consensus protocols with benign and deceitful faults. *preprint arXiv:2204.08670*, 2022.

[48] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.

[49] Alex Shamis, Peter Pietzuch, Burcu Canakci, Miguel Castro, Cédric Fournet, Edward Ashton, Amaury Chamayou, Sylvan Clebsch, Antoine Delignat-Lavaud, Matthew Kerner, Julien Maffre, Olga Vrousgou, Christoph M. Wintersteiger, Manuel Costa, and Mark Russinovich. IA-CCF: Individual accountability for permissioned ledgers. In *Proc. of the 19th USENIX Symp. on Networked Systems Design and Implementation*, NSDI'22, 2022.

[50] Peiyao Sheng, Gerui Wang, Kartik Nayak, Sreeram Kannan, and Pramod Viswanath. BFT protocol forensics. In *Proc. of the 28th ACM Conference on Computer and Communications Security (CCS)*, 2021.

[51] Douglas Simoes Silva, Rafal Graczyk, Jérémie Decouchant, Marcus Völp, and Paulo Esteves-Verissimo. Threat adaptive Byzantine fault tolerant state-machine replication. In *Proc. of the 40th IEEE Int. Symp. on Reliable Distributed Systems (SRDS)*, pages 78–87, 2021.

[52] Yee Jiun Song and Robbert van Renesse. Bosco: One-step Byzantine asynchronous consensus. In *Proc. of the 22nd International Symposium on Distributed Computing (DISC)*, pages 438–450, 2008.

[53] João Sousa and Alysson Bessani. From Byzantine consensus to BFT state machine replication: A latency-optimal transformation. In *Proc. of the 9th IEEE European Dependable Computing Conf. (EDCC)*, pages 37–48, 2012.

[54] João Sousa and Alysson Bessani. Separating the wheat from the chaff: An empirical design for geo-replicated state machines. In *Proc. of the 34th IEEE Int. Symp. on Reliable Distributed Systems (SRDS)*, pages 146–155, 2015.

[55] Chrysoula Stathakopoulou, Tudor David, and Marko Vukolić. Mir-BFT: High-throughput BFT for blockchains. *preprint arXiv:1906.05552*, 2019.

[56] Xiao Sui, Sisi Duan, and Haibin Zhang. Marlin: Two-phase bft with linearity. In *Proc. of the 52nd Annual IEEE/IFIP Int. Conf. on Dependable Systems and Networks (DSN)*, pages 54–66, 2022.

[57] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, and Lau Cheuk Lung. EBAWA: Efficient Byzantine agreement for wide-area networks. In *Proc. of the 12th IEEE Int. Symp. on High Assurance Syst. Eng. (HASE)*, pages 10–19, 2010.

[58] Benjamin Wester, James Cowling, Edmund B. Nightingale, Peter M. Chen, Jason Flinn, and Barbara Liskov. Tolerating latency in replicated state machines through client speculation. In *Proc. of the 6th USENIX Symp. on Networked Systems Design and Implementation (NSDI)*, pages 245–260, 2009.

[59] Lei Yang, Seo Jin Park, Mohammad Alizadeh, Sreeram Kannan, and David Tse. DispersedLedger: High-throughput Byzantine consensus on variable bandwidth networks. In *Proc. of the 19th USENIX Symp. on Networked Systems Design and Implementation*, NSDI'22, 2022.

[60] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: BFT consensus with linearity and responsiveness. In *Proc. of the 38th ACM Symp. on Principles of Distributed Computing (PODC)*, pages 347–356, 2019.

[61] Yunhao Zhang, Srinath Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. Byzantine ordered consensus without Byzantine oligarchy. In *Proc. of the 14th USENIX Symp. on Operating Systems Design and Implementation*, OSDI'20, 2020.

Appendix

A FlashConsensus Correctness Argument

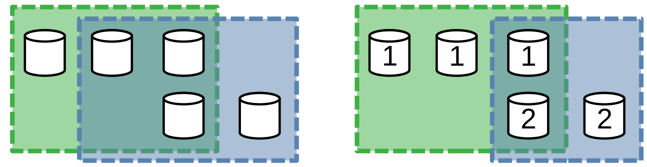
In the following, we argue why FlashConsensus, as summarized in Figure 7, preserves the safety and liveness of its underlying protocol, AWARE [9], with up to t failures.

Safety. Intuitively, we need to ensure the following property: if an operation o is finalized at a client, then it was the i-th executed operation in all correct replicas, and no client observes the effects of some $o' \neq o$ executed at position i.

A client c accepts an operation o as *finalized* if it receives enough matching replies, depending on FlashConsensus's mode:

- *Conservative mode:* A correct replica generates a response for an operation o after executing it as its i-th operation. This happens after a batch containing o was decided in an AWARE-NC consensus instance j. Since AWARE implements consensus tolerating t failures, all correct replicas will decide the same batch in consensus j, execute o at position i, and generate the same reply. This means at least $n-t$ replicas will produce the same reply for o, enabling c to collect a weighted response quorum [54] to finalize it.
- *Fast mode:* In this mode, client c finalizes o only if it collected $n - t_{fast} - 1$ matching replies for it (see §4.3). We have to consider two cases, depending on the actual number of faulty replicas f in the system.
    1. Case $f \leq t_{fast}$: This case is analogous to the conservative mode since AWARE-NCF satisfies SMR safety under t_{fast}.
    2. Case $t_{fast} < f \leq t$: In this case, an equivocation can cause two correct replicas to execute conflicting operations $o' \neq o$ at position i. However, the fact c waits for $n - t_{fast} - 1$ matching replies for finalizing an operation o ensures this operation will never be reverted, and replicas executing o' can not get their responses accepted as shown in §4.3.

Liveness. As defined in §3, SMR liveness comprises the guarantee that any operation o issued by a correct client eventually completes. To argue about that, suppose a correct client c sends operation o to the replicas. Again, we have to consider the two modes of FlashConsensus:



(a) *Egalitarian*: All quorums have the same size of exactly $\lceil \frac{n+t+1}{2} \rceil$ replicas.

(b) *Weighted*: A quorum contains at most $n-t$ and at least $2t+1$ replicas.

Figure 15: BFT quorums for $n = 5$, $t = 1$, and $\Delta = 1$ (on b).

- *Conservative mode:* If the leader is correct and there is sufficient synchrony, then a batch containing o will be eventually decided through AWARE-NC. In case of a faulty leader or asynchrony, the timers associated with o on correct replicas will expire, and the AWARE-SYNC will be executed until a correct leader can use AWARE-NC to force replicas to decide a batch containing o (potentially after GST). At this point, $n-t$ correct replicas send matching replies, which are collected by c until it has a weighted response quorum [54] to finalize o.
- *Fast mode:* In fast mode, we have to consider two cases:
    1. Case $f \leq t_{fast}$, correct leader, and synchrony: This case is analog to the conservative mode because AWARE-NCF solves consensus with up to t_{fast} faulty replicas.
    2. Case $t_{fast} < f \leq t$ or faulty leader or asynchrony: In this case, AWARE-NCF *might not* be finished, and the timers associated with o will expire in correct replicas. This will cause replicas to initiate the synch. phase and changes to the conservative mode. Alternatively, f Byzantine replicas might participate in consensus but not reply to c, which will never be able to collect $n - t_{fast} - 1$ matching replies. In this case, the client periodically reattempts to confirm the result of o by checking the log of decisions until the next checkpoint is formed to verify in which position on the finalized request log o appears. This will eventually happen, because if o is not ordered successfully, then a timeout at the replicas triggers, which causes replicas to initiate the synch. phase and switches the protocol to the conservative mode. The liveness argument then continues from the conservative mode.

B Weighted BFT Replication

WHEAT. WHeight-Enabled Active replicaTion (WHEAT) [54] is an empirical design for optimizing BFT SMR for geographically dispersed deployments. A core insight of WHEAT is that client latency can be reduced by utilizing Δ additional replicas that do not contribute to increasing the resilience threshold, i.e., $n = 3t + 1 + \Delta$.

Quorums in WHEAT are not formed using an egalitarian Byzantine majority of replicas like in other BFT works (e.g., [16,53,56,60]), but are formed instead by weighted replication, which enables proportionally smaller quorum sizes. The knack is that a clique of well-connected replicas (replicas with low latency to each other) can be nominated to be the ones forming these smaller quorums (see Figure 1a). The approach still ensures the availability of the replicated system since other replicas with lower voting weight can be accessed to form fallback quorums.

BFT systems typically probe a Byzantine dissemination quorum, which contains $\lceil \frac{n+t+1}{2} \rceil$ replicas [39] as shown in Figure 15a. With $n = 5$ replicas, the quorum size needs to be 4 to ensure these quorums intersect in at least $t + 1$ replicas.

Albeit, it is also possible to satisfy the *same intersection property* by relying on specific replicas more than on others, which is captured by assigning *weights* as we can see in Figure 15b. A fast quorum comprises three replicas with a total voting weight of five. This quorum is smaller than an egalitarian quorum, yet it guarantees the intersection since it contains all the replicas with high voting power.

Now, suppose that the fastest, geographically nearest replicas are the ones in this particular quorum. They can progress the voting phases of consensus more swiftly, as they need to wait less time to collect the necessary votes. By accelerating consensus, WHEAT can also decrease the overall latency of the system [54].

Unlike traditional systems like PBFT, WHEAT does not wait for a predefined number of replicas but waits for a predefined sum of votes of Q_v.

To distribute weights (or voting power) among the replicas, WHEAT uses a bimodal scheme that divides the voting power in such a way that the $2t$ best-connected replicas have a voting power of $V_{max} = 1 + \frac{\Delta}{t}$ while the remaining replicas have a voting power of 1. As a result, the number of votes required for a quorum is $Q_v = 2t \cdot V_{max} + 1$. This distribution ensures that even if t best-connected replicas fail, it is possible to form quorums using the other replicas.

Notice that the size of the *smallest* quorum only depends on the chosen threshold t, not on the actual size of the system n. When using this weighting scheme, all quorums contain between $2t + 1$ and $n - t$ replicas.

AWARE. Distributing weights is difficult in practice, as the decision of what is the optimal weight configuration for a given set of network characteristics is non-trivial. To tackle this issue, AWARE (Adaptive Wide-Area REplication) [9] allows geo-replicated state machines to self-optimize at runtime. By using an automated dynamic weight tuning and leader placement scheme, AWARE supports the emergence of fast quorums in the system.

At its core, AWARE monitors latencies between all replicas and creates a corresponding latency matrix. This matrix is used to predict the expected consensus latency for several configurations of weight distributions and leader locations. Given a fixed threshold t, AWARE automatically chooses the fastest configuration found using the latency matrix, and a meta-heuristic called simulated annealing [35]. By running this method periodically, AWARE can adapt to changes in the network: when the latency matrix is recalculated and changes are detected, the system automatically reconfigures the weight distribution and/or leader location to better suit the current network conditions.

C Validation of Emulated/Simulated Networks

In this section, we compare the results of the experiment conducted in Section 6.1.1 for which we used the real AWS cloud infrastructure with supplementary experiments that use an emulated and simulated network environment that mimic the AWS infrastructure by using latency statistics from cloudping. For these network environments, we use state-of-the-art network emulation and simulation tools, namely Kollaps [28] and Phantom [33]. The repeated experiments should give some insights of how close real network characteristics can be modeled using emulation and simulation tools. A threat to validity is that the network statistics average latency observations over a larger time span (i.e., a year). In contrast, when conducting experimental runs, short-time fluctuations might make individual network links appear faster or slower than usual, which can impact the speed at which certain quorums are formed. Nevertheless, we are convinced that using the network tools and latency statistics creates reasonably realistic networks that can be used to validate the latency gains of the quorum-based protocols we are working with.

In Figure 16, we contrast protocol runs for BFT-SMaRt, AWARE, and FlashConsensus on a real network, as well as in the Kollaps-based and Phantom-based networks. On average, we observed latencies that were 1.5% higher in the emulated network than on the real AWS network. Respectively, the simulated network yielded latency results that were, on average, 0.8% lower than the real network

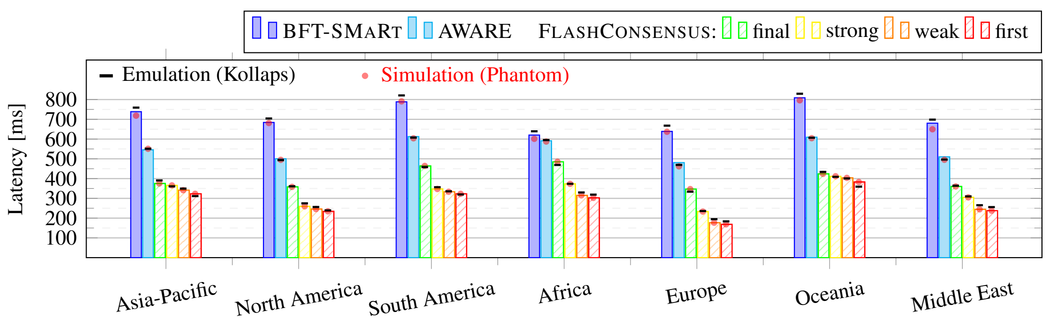


Figure 16: Comparison of clients' observed end-to-end latencies for protocol runs with BFT-SMaRt, AWARE and FlashConsensus in different network environments: real, emulated, and simulated. The client results are averaged over all regions per continent.