(Online Appendix) *Chasing Lightspeed Consensus*: Fast Wide-Area Byzantine Replication with Mercury

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

Online Appendix

In this companion online appendix, we present the full correctness proof of the Mercury and the experimental results about the validity of our simulation and emulation scenarios when compared with a real network. In particular, in this appendix, we prove the following theorems.

Theorem 1. *If an operation o is finalized in i-th position of the decision log, then no client observes an operation $o' \neq o$ in this position of the decision log.*

Theorem 2. *An operation issued by a correct client will eventually be finalized.*

Correctness of Mercury

In the following, we prove the correctness of Mercury transformation, as summarized in Figure 1. In particular, we prove it preserves the safety and liveness of its underlying protocol, AWARE [1], with up to t failures.

To abstract the exact service being implemented on top of Mercury, our proofs consider the replicated decision log abstraction as described in §2.1, in which every operation needs to be allocated consistently in a given position/slot of the log. We say an operation is *finalized* when the client that issued it knows it was durably executed and will never be reverted in the system. The exact number of replies required for finalizing an operation is defined in Rule C2, in Figure 1.

Safety

Instead of only proving all operations are executed in total order, we have to prove all clients observe operations in total order, which ensures linearizability [4]. To prove that, we start by showing that an operation finalized in some position of the decision log in a mode of operation is kept in that position after a mode switch.

Proposition 1. *Let o be an operation finalized in* conservative *mode in the i-th position of the decision log. After the system switches to subsequent* fast *mode, o will still be the i-th operation executed in the system.*

Proof. Assume, without loss of generality, that the system switches from *conservative* to *fast* mode right after executing its j-th operation, with $j \geq i$. In this case, all operations ordered after the switch will be finalized in positions after j (Rule S3 of Figure 1), and thus the effect of previously ordered operations (including o) will be maintained. □

Lemma 1. *Let o be an operation finalized in* fast *mode in the i-th position of the decision log. After a synchronization phase that switches the system to the* conservative *mode, o will still be the i-th operation executed in the system.*

Proof. Let's assume, for the sake of contradiction, that an operation o finalized in the i-th position of the decision log in fast mode does not appear in this position after a synchronization phase. This can happen for one of two reasons:

1. *Operation o was erased, making the i-th position empty.* This can only happen if there is no correct replica in the intersection between the quorum of $n - t_{fast} - 1$ replicas that informed the client about the finalization of o and the quorum of $n - t$ replicas that informed the new leader about the requests ordered in fast mode during the synchronization phase. This is not possible since these quorums intersect in $(n - t_{fast} - 1) + (n - t) - n = \frac{3t}{2}$ replicas[1], which is clearly bigger than t, for $t \geq 2$ (when the fast mode brings benefits).
2. *Operation o was replaced by another operation o' in the i-th position.* This happens only if the number of replicas reporting o' as prepared is higher than the number of replicas reporting o in the synchronization phase quorum of $n - t$ replicas (Rule S6 step 2 of the transformation in Figure 1). Since o was finalized, at least $C = n - t_{fast} - 1 - t$ *correct replicas* informed the client about the execution of o. Under an equivocation scenario, operation o' would be reported by at most $C' = n - C + t = t + t_{fast} + 1$ replicas, including the equivocators that participated in the decision of o and o'. Notice that the $t_{fast} + 1$ replicas that maliciously voted in the preparation of the two requests can be detected by the lightweight forensics protocol used in the synchronization phase (Rule S6 step 1 of the transformation). Therefore, these replicas will be ignored in the synchronization phase quorum (Rule S6 step 2) (and later expelled from the system—Rule S6

[1]This value is obtained by using $t_{fast} = \frac{t}{2}$ and $n = 3t + 1$.

Client

C1 **Invocation:** Send $\langle \textsc{request}, o \rangle$ to all replicas.

C2 **Finalization:** Accept a result *res* for o if received a set of matching replies $rep = \{\langle \textsc{reply}, h(o), \mathit{fast}, \mathit{res} \rangle\}$ such that either:

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

In case of a timeout, keep re-sending the request and inspecting the decision log of replicas until one of the conditions above is satisfied.

C3 **Panic:** Broadcast $\langle \textsc{panic}, o, rep \rangle$ to the replicas if replies $rep = \{\langle \textsc{reply}, h(o), \texttt{true}, * \rangle\}$ contains diverging results for operation o.

Replica

State

fast	mode of operation	boolean	`false`
chkp	last stable checkpoint	bytes	`null`

Building Blocks

AWARE	AWARE SMR protocol (conservative)
AWARE★	normal case operation of AWARE in fast mode
AUDIT	lightweight forensics procedure of Figure 2

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

S2 **Periodic Checkpoint:** When a snapshot of the service state $chkp'$ is created after processing consensus j, broadcast signed message $\langle \textsc{checkpoint}, h(chkp'), j \rangle$. 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, run AUDIT.

S3 **Client Panic:** If *fast* and received a message $\langle \textsc{panic}, o, rep \rangle$ with diverging signed replies for o in rep from a client, run AUDIT.

S4 **Abort:** If *fast*: Broadcast a VIEW-CHANGE message if one condition applies: (1) a request *timer* expires, (2) a message $\langle \textsc{poc}, poc \rangle$ with a valid PoC is received from some replica, or (3) upon *consensus latency disappointment*—see §4.1.3.

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

S6 **Synch. Phase:** Upon receiving $t + 1$ matching VIEW-CHANGE messages, use AWARE' synch. phase to replace the current leader and synchronize the decision log. If *fast*, set *fast* to `false` and run AUDIT if no $\langle \textsc{poc}, poc \rangle$ message has been received so far.

1. When waiting for NEW-VIEW messages, the next leader checks for PoCs and ignores messages from equivocating replicas. When consolidating operations for each position of the decision log, the new leader picks the most commonly reported prepared value.
2. Upon a PoC is produced or received during the synch. phase, 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 $\langle \textsc{reconfigure}, \mathit{culprits}, poc \rangle$ using AWARE.
3. Upon deciding $\langle \textsc{reconfigure}, \mathit{culprits}, poc \rangle$, a replica verifies the *poc* using AUDIT, and removes the *culprits* from the system.

Figure 1. A summary of Mercury.

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 at least $t_{fast} + 1$ malicious replicas that provided signed Accept messages for both decisions. These two conflicting proofs are the PoC (proof-of-culpability) for the malicious replicas.

F3 **Blame culprits:** If $poc \neq \emptyset$: Broadast a $\langle \textsc{poc}, poc \rangle$ message.

Figure 2. Lightweight forensics procedure.

steps 3 and 4), ensuring no more than t replicas report o' in this quorum. Consequently, even in this worst-case scenario, the new leader will always see $n - 2t > t$ replicas reporting o, being thus the most common value appearing in the reported values. This result will make o be kept in the i-th position of the log after a synchronization phase.

□

The following theorem proves Mercury' main safety property: finalized operations are observed in the same position of the decision log by every correct client.

Theorem 1. *If an operation o is finalized in i-th position of the decision log, then no client observes an operation $o' \neq o$ in this position of the decision log.*

Proof. We start by observing that, according to Rule C2 of the transformation (Figure 1), a client accepts an operation result as finalized only if the number of matching REPLY messages *in the same mode* satisfies certain quorum sizes. Let $mode(o)$ be the mode of operation in which o was finalized, and assume, for the sake of contradiction, that an operation $o' \neq o$ appears as finalized in the i-th position for some correct client.

To show correct clients cannot observe different operations o and o' in i-th position of the system history, we need to consider all possible combinations of modes for o and o'. We start by considering the cases in which both operations were finalized in the same mode without any switch in the system mode between their executions. There are two cases to consider:

1. $mode(o) = mode(o') = \mathit{conservative}$: if both operations were executed in the conservative mode, then the total order of operations ensured by AWARE's state machine replication algorithm [1, 6] makes it impossible for different clients to observe finalized operations o and o' in the same position in the system history.
2. $mode(o) = mode(o') = \mathit{fast}$: if both operations were ordered in fast mode, then $f > t_{fast}$ malicious replicas (including the leader) can lead to different correct replicas deciding different operations for the same position

i of the decision log. In this case, we have to show that clients waiting for $n - t_{fast} - 1$ matching replies is enough to ensure they cannot observe two finalized operations in the same position. This holds because the size of the intersection of any two reply quorums is $(n - t_{fast} - 1) + (n - t_{fast} - 1) - n = n - t - 2$, which is bigger than t for any $t \geq 2$ (when the fast mode can be used). This means correct clients will not observe two finalized operations in the same decision log position in our system model.

Now, we need to consider the cases in which there was exactly one mode switch between the finalization of o and o'. There are two cases to consider:

1. $mode(o) = conservative$ and $mode(o') = fast$: Proposition 1 states that finalized operation o position cannot be altered in a conservative-to-fast switch, i.e., it will still appear as the i-th operation executed, and thus operation o' cannot appear in position i.
2. $mode(o) = fast$ and $mode(o') = conservative$: Lemma 1 states that if o was finalized in fast mode, a subsequent synchronization phase that switches the system to conservative mode keeps the o in the i-th position of the decision log, making thus impossible for another operation o' to be finalized in this position.

Lastly, we need to consider all the cases above, but in which multiple mode switches happen between the finalization of o and o'. Proposition 1 and Lemma 1 ensure a finalized operation is preserved in the system state even after a mode switch. Consequently, by applying induction arguments, it is easy to see that the number of switches between the finalization of o and o' does not change the fact o will always remain the i-th operation in the decision log. □

Liveness

As defined in §3, SMR liveness comprises the guarantee that operations issued by correct clients are eventually finalized. Proving the liveness of BFT protocols is generally perceived as considerably more intricate than proving their safety [2]. However, this distinction does not apply to MERCURY because our transformation relies upon the underlying protocol (AWARE, a variant of the well-known BFT-SMaRt) for ensuring this property. The following theorem defines the liveness of MERCURY.

Theorem 2. *An operation issued by a correct client will eventually be finalized.*

Proof. To argue about that, suppose a correct client c sends operation o to the replicas. As stated before, an operation is finalized if the client observes it was executed in a certain position by sufficiently many replicas in the same mode (Rule C2 of Figure 1). Again, we have to consider the two modes of MERCURY:

1. *Conservative mode:* If the leader is correct and there is sufficient synchrony, then a batch containing o will be eventually decided through AWARE (Rule S1). In case of a faulty leader or asynchrony, the timers associated with o on correct replicas will expire, and the synchronization phase will be executed until a correct leader forces replicas to decide a batch containing o (potentially after GST). At this point, at least $n - t$ correct replicas will send matching replies, which are collected by c until it has a weighted response quorum [7] to finalize o.
2. *Fast mode:* In fast mode, before GST no liveness can be ensured, and the system will switch back to the conservative mode. After GST, we have to consider two cases:
   a. Case $f \leq t_{fast}$ and correct leader: This case is analogous to the conservative mode because AWARE★ solves consensus with up to t_{fast} faulty replicas.
   b. Case $t_{fast} < f \leq t$ or faulty leader: In this case, AWARE★ *might not* be finished, and the timers associated with o (Rule S1) will expire in correct replicas. This will cause replicas to initiate the synchronization phase (Rule S4), switching to the conservative mode, in which the request will be ordered and finalized, as explained above. 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 (Rule C2). This can be repeated until the next periodic checkpoint is formed to verify in which position on the decision log o appears (Rule S2). This will eventually happen because if o is not ordered successfully, a timeout will trigger at the replicas, causing them to initiate the synchronization phase and switch the protocol to the conservative mode. The liveness argument from the conservative mode then eventually holds for o.

□

Validation of Emulated/Simulated Networks

In this section, we compare the results of the experiment conducted in §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 [3] and Phantom [5]. The repeated experiments should give some insights into 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 period (i.e., a year). In

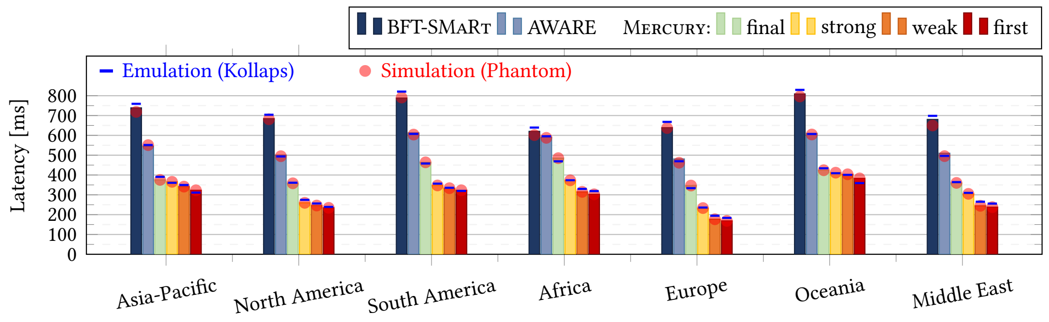

Figure 3. Comparison of clients' observed end-to-end latencies for protocol runs with BFT-SMaRt, AWARE and Mercury in different network environments: real, emulated, and simulated. The client results are averaged over all regions per continent.

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 improvements of the quorum-based protocols we are working with.

In Figure 3, we contrast protocol runs for BFT-SMaRt, AWARE, and Mercury 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.

References

[1] Christian Berger, Hans P. Reiser, João Sousa, and Alysson Neves Bessani. 2022. AWARE: Adaptive Wide-Area Replication for Fast and Resilient Byzantine Consensus. *IEEE Transactions on Dependable and Secure Computing (TDSC)* 19, 3 (2022), 1605–1620. https://doi.org/10.1109/TDSC.2020.3030605

[2] Manuel Bravo, Gregory Chockler, and Alexey Gotsman. 2022. Liveness and latency of Byzantine state-machine replication. In *Proc. of the 36th Int. Symp. on Distributed Computing (DISC)*. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 12:1–12:19.

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

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

[5] Rob Jansen, Jim Newsome, and Ryan Wails. 2022. Co-opting Linux Processes for High-Performance Network Simulation. In *Proc. of the USENIX Annual Technical Conference (USENIX ATC)*. USENIX Association, Berkeley, CA, 327–350.

[6] João Sousa and Alysson Bessani. 2012. From Byzantine Consensus to BFT State Machine Replication: A Latency-Optimal Transformation. In *Proc. of the 9th IEEE European Dependable Computing Conf. (EDCC)*. IEEE, 37–48.

[7] João Sousa and Alysson Bessani. 2015. 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)*. IEEE, 146–155. https://doi.org/10.1109/SRDS.2015.40