

SQLodin Working Group

Request for Discussion and Implementation Record

SOD 0005

Engineering Discussion

Durable Turns, Fast Learning, Quorum Reads and a Journal-Backed Application

STATE

DISCUSSION

CREATED

2026-09-25

LAST UPDATED

2026-09-25

INTENDED STATUS

In Discussion

AUTHORS

Vikrant Rathore, with assistance from Ronak Rathore

DISCUSSION FORUM

Remove serialized sync barriers from the durable write and fresh-read path

TOPICS & LABELS

performanceconsensusstoragetheory

Status of This Record

This document is a **SQLodin Discussion (SOD)** record authored in Typst and tracked in git. It follows an RFC-style specification structure so that architectural scope, rationale, trade-offs, proof obligations, and operational boundaries remain explicit. Documents using the placeholder number XXXXX are provisional drafts. Numbered SOD documents are permanent engineering records for discussion on improvement, architecture, and enhancements of SQLodin.

Contents

1	Abstract	2
2	Status and Implementation Boundary	2
3	Introduction & Background	2
4	Terminology and Scope	3
5	Problem Statement	3
6	Goals and Non-Goals	3
6.1	Goals	3
6.2	Non-Goals	3
7	High-Level Architecture & Workflow	4
8	Detailed Subsystem Design	4
8.1	M1: one durable barrier per service turn	4
8.2	M2: one-message learning of an owner's no-op	5
8.3	M3: skip in the turn that observes a higher slot	5
8.4	M4: the application database as a cache of the journal	5
8.5	M5: learning an owner's value from this voter's own vote	6
8.6	M6: quorum-frontier reads	6
8.7	Service I/O findings	7
9	Invariants & Correctness Guarantees	7
10	Security & Operational Considerations	7
11	Resource Budgets & Performance Targets	7

12	Validation and Acceptance Gates	8
12.1	Recorded measurements and evidence	8
12.1.1	Staged experiment (smallest test first)	8
12.1.2	Same-host matrix against same-run SQLite (.18)	9
12.1.3	Three hosts (.19/.20/.21), client on .18	9
12.1.4	Corrected-harness review measurement	9
12.1.5	Remaining gaps	9
12.1.6	Regression evidence	10
13	Alternatives Considered	10
14	Open Questions & Future Milestones	10
15	Discussion and Revision Notes	10
16	References	11

1 Abstract

The pre-SOD durable service performed up to seven **sequential** synchronization barriers per write and per fresh read. Group commit amortizes them across concurrent requests, but it cannot remove barriers that depend on one another. This record measures where the barriers come from, then adopts six changes. Each has a stated proof obligation, a model or proof, and a negative control:

- one journal barrier per service turn;
- one-message learning of an owner's no-op (Mencius simple consensus);
- skipping idle slots in the turn that observes a higher slot;
- the application database as a replayable cache of the FULL-synchronized journal;
- learning an owner's value from this voter's own vote when the write quorum is two;
- fresh reads through a quorum frontier (Paxos Quorum Reads) instead of a proposed marker.

The pinned Paxos library and its quorum rules are unchanged; the adapter uses only its public API. Measured on the same host against the same-run SQLite reference, fresh-read latency fell from about 21 ms to 0.27 ms and every workload ratio improved; the results section records the gaps that remain. **SOD** stands for **SQLODIN Discussions** — versioned engineering records for discussion on improvement, architecture, durable storage design, and throughput optimization for SQLodin.

2 Status and Implementation Boundary

In discussion; opened 25 September 2026. The record follows SOD 0004's open question: measure the saturation point first, then choose mechanisms. It adds no alternate consensus algorithm, dynamic membership or weaker acknowledgement, and it changes no paxos-odin source. The journal keeps synchronous=FULL. Only the application database of the separated format-5 store changes to WAL synchronous=NORMAL. The legacy format-4 store and the embedded Engine API keep FULL. The embedded durable host keeps its marker read barrier; only the network service uses quorum reads. This work post-dates the qualified 25 September release candidate. It needs its own qualification before any release record may cite it.

The implementation is available for review, but Discussion is not an acceptance decision. The review corrections below have separate evidence; historical measurements keep their original binary hashes. Quorum-frontier reads require homogeneous service capability, and the peer fingerprint now explicitly identifies that capability.

3 Introduction & Background

SOD 0004 identified synchronization and host scheduling as measurable costs. This discussion keeps the fixed-membership SQL contract and examines where durable work can be shared or replayed.

4 Terminology and Scope

A turn is one serialized service iteration, possibly split by capacity limits into multiple journal groups. A learned entry has either quorum decision evidence or the specific M2 no-op determinacy argument. A quorum frontier observes distinct configured voters after the read cohort closes. The application cache remains a transactional SQLite database; its FULL journal is the recovery source.

5 Problem Statement

On .18, with three local voters, ZFS storage, pinned SQLite 3.51.3 and the unchanged `calibrate_native_mixed.py` harness, the current candidate measured the following. Each case used 100 operations per client, 256-byte values and the FULL SQLite reference with groups of at most 16.

Clients	Reads %	SQLite tx/s	SQLodin tx/s	Ratio
1	70	985.3	19.8	2.0%
1	0	209.6	20.5	9.8%
8	70	1,153.2	70.1	6.1%
8	0	850.9	70.7	8.3%
32	70	2,974.7	160.1	5.4%
32	0	2,860.2	119.3	4.2%

One sequential client needs about 49 ms per write, while one FULL SQLite commit needs about 4.8 ms. A system-call trace (`strace -f -T`, run under the tracer rather than attached) of 20 sequential writes and 20 fresh reads at voter 1 recorded 5.3 sync calls per operation at the owner and 7.3 at each follower. Tracing inflates latency; the counts, not the times, are the finding. The trace shows this chain for one write in slot *s* owned by voter 1, with voters 2 and 3 idle:

1. Owner: vote for *s*, FULL journal commit, then Accept is released.
2. Followers: vote for *s*, journal commit, Accepted is released.
3. Followers, **next loop turn**: `node_progress` skips their own slots below *s*. Each skip is a separate owner proposal with its own journal commit.
4. Owner and the other follower vote on each skip (journal commit) and reply.
5. Skip owners count the quorum, record the decision (journal commit) and broadcast Commit.
6. Owner records each decision (journal commit). Only now is its prefix through *s* contiguous.
7. Owner applies the SQL in its application database with another FULL commit, then acknowledges. A fresh read proposes a marker, so it follows the same chain. The service loop also executes `step_batch`, `propose_batch`, `begin_read`, `tick`, `progress` and local delivery as separate durability transitions. A busy turn can therefore sync four or more times. A further CPU cost is that each journal record re-measures the history budget, opening the catalog database and running `lstat/statvfs`.

6 Goals and Non-Goals

6.1 Goals

Reduce sequential barriers and fresh-read log work without losing acknowledged SQL effects, retry fences, agreement or read freshness. Retain bounded service work and attributable measurements.

6.2 Non-Goals

No dynamic membership, alternate Paxos library, unrestricted SQL or weaker durable acknowledgement. No throughput guarantee follows from the latency model. The prior release qualification is not automatically qualification of this changed host.

7 High-Level Architecture & Workflow

Let d be one FULL sync barrier, δ one one-way message delay, and a the application CPU time of a request. Write L for the acknowledgement latency of an uncontended write. Then:

$$L_{\text{now}} \approx 7d + 6\delta + a \quad \text{and} \quad L_{\text{new}} \approx 3d + 2\delta + a.$$

The new chain is owner vote, follower vote with skips, and owner decision followed by unsynchronized application. With $d \approx 9$ ms, this predicts roughly a $2.3\times$ single-client improvement.

Throughput has two limits. First, a single-threaded voter executes at most $\frac{1}{d+c}$ turns per second, where c is the turn's CPU time. Each turn can admit $B \leq 16$ proposals, so

$$X \leq N \cdot \frac{B}{d+c}.$$

Second, by Little's law, at most $W = 64$ undecided or unreleased slots exist in the window, including skips. With skip fraction f , useful throughput therefore satisfies

$$X \leq (1-f) \cdot \frac{W}{L}.$$

Today the dominant term is the sync count per turn, not the window. With one barrier per turn and L_{new} near 30 ms, the window bound is about 2,100 slots/s. That remains well above the current throughput. Raising W is a later decision, to be made only if measurement shows the window saturating.

8 Detailed Subsystem Design

8.1 M1: one durable barrier per service turn

A **turn** opens one journal transaction. It stages every protocol transition of the loop iteration into that transaction: authenticated peer packets, new client proposals, the timer tick when due, and ownership progress. Network read cohorts use M6 and do not propose a marker. It commits once and applies the released contiguous prefix. Only then does it release messages and host requests. The existing `Transition_Group` already does this for up to sixteen packets. M1 extends it to every transition kind. A group that would exceed its message, decision or request capacity commits early, and the next transition opens a new group. A capacity limit can therefore add a barrier but never drop work.

INVARIANT – Durable-before-release (unchanged, SOD 0003 `DurableBeforeSend`)

For every released message or committed entry e produced by transition t_i of a turn, $\text{writes}(t_1) + \dots + \text{writes}(t_i)$ is durable before e leaves the host.

PROOF – M1 refines per-transition persistence

Let $T = t_1 \dots t_k$ be a turn. The single SQLite transaction contains the concatenation of the transitions' write records, in the transitions' order, hash-chained and sequenced exactly as k separate commits would be. SQLite transactions are atomic, so after a crash the durable journal is either the pre-turn journal or the post-turn journal. In the first case no effect of T was released: releases occur only after COMMIT returns. The world therefore cannot distinguish this case from a crash before t_1 , which is a behavior of the reference host. In the second case the durable state equals the reference host's state after t_k . Every release is covered by the invariant, because a prefix of the concatenation is durable whenever the whole of it is. Any in-memory state advanced before a failed commit is discarded because the host is poisoned and restarts from the journal, as `step_batch` already does. $\square$

The per-record history check becomes one check per group. `HISTORY_TRANSITION_RESERVE` is sized for $\text{PACKED_CAPACITY} \times (2 \cdot \text{CHUNK} + 1) \times \text{MAX_STEP_BATCH} \times 4$ records, which is 2,112 packed records. A group is flushed before it reaches 1,024 records. The reserve therefore still bounds every group, and admission control still runs before each proposal.

8.2 M2: one-message learning of an owner's no-op

Rotating ownership partitions each slot's ballots. Round 0 belongs to the slot owner $o(s)$ alone, and rounds $r \geq 1$ belong to revokers. A revoker proposes the highest-ballot vote found in its read quorum, or the host no-op $\perp$ if it finds none (B3 in `start_revocation/maybe_resolve_chunk`).

INVARIANT – Owner-skip determinacy

If $o(s)$ has issued a round-0 Accept for $(s, \perp)$, then every Accept ever issued for s , at any ballot, carries $\perp$. Consequently $\perp$ is the only value that can be chosen for s .

PROOF – Induction on ballots

Premises: (P1) acceptors refuse a round-0 Accept whose ballot node is not $o(s)$ (`on_accept`). (P2) $o(s)$ issues at most one round-0 value per slot across crashes: its vote is durable before the Accept is released, and after restart `own_next` resumes above the highest used slot. (P3) A revoker's offer at round r equals the value of the highest-ballot vote in a read quorum of promises at r , or $\perp$ if there is none.

Base: by P1 and P2, the only round-0 value for s is $\perp$. Step: assume every offer at a round below r carries $\perp$. Every vote at a round below r is for some offer, so it carries $\perp$. By P3, the round- r offer is either a reported vote, which carries $\perp$, or $\perp$ itself. A value is chosen only if it has a quorum of votes for one offer, so only $\perp$ can be chosen. $\square$

A learner may therefore record $(s, \perp)$ as decided as soon as it receives the owner's round-0 Accept for $\perp$. The owner itself still learns through its quorum of replies; self-learning was not needed. This is exactly Mencius's "learn no-op from the coordinator's SKIP" rule [Mao et al., OSDI 2008]. SQLodin states it more conservatively than Mencius: the owner's no-op vote must still be durable before the Accept leaves, because P2 relies on it. The rule applies only to the exact host no-op mutation `make_skip(0, 0)`. Read markers, snapshot barriers and seals carry nonzero keys and still require a quorum.

Implementation (`owner_fast_commit` in `src/paxos.odin`): when a turn steps an Accept at round 0 whose ballot node is both the sender and `owner_of(slot)`, and whose value equals the host no-op, it also steps `Commit(slot, bot)` from that sender in the same group. The Commit uses the upstream's public `node_step` and existing `on_commit`, so no upstream code changes. If a peer misses the Accept, retransmission (`resend_to`) still sends Commit for decided cells, so the change does not remove any liveness path.

Read freshness (`ReadFence/ReadCohort`) is unaffected. That proof needs the marker's slot to be undecided when the read begins: if another voter had already applied past it, the slot's value was fixed before the marker existed. Fast learning changes **when** a voter learns the value of a no-op slot. It does not change **which** value that slot can have.

8.3 M3: skip in the turn that observes a higher slot

Mencius Rule 2 skips a server's own unused instances as soon as it receives a suggestion for a higher one, and piggybacks the skip on the accept reply. In SQLodin, `highest_seen` rises when a turn steps an Accept. M1 runs ownership progress after the packets of the same turn, so the skips are voted in the same barrier and released together with the Accepted reply. This is ordinary owner proposal traffic and needs no new safety argument. It removes one barrier and one network hop from the chain, and with M2 it removes the skip's own quorum round.

8.4 M4: the application database as a cache of the journal

A separated store already recovers by replaying the chosen suffix above the application watermark (`recover_application`). The FULL barrier on every application commit is therefore redundant **for durability**, provided that three conditions hold.

INVARIANT – Journal-backed application (J1–J3)

J1: an entry is applied only after the journal transaction containing its decision record has committed.
J2: for every generation with base b , the journal retains every decision record in $(b, A_{\text{mem}}]$, and the base image is FULL-synchronized. **J3:** every application commit is deterministic in (prior state, slot, value) under the SQL policy and build identity already required by SOD 0003.

PROOF – Acknowledged outcomes survive power loss

WAL with synchronous=NORMAL guarantees atomicity and consistency. After any crash, the database equals the result of some prefix of its committed transactions, and never less than the last completed checkpoint. For a separated store, that prefix ends at some $A_{\text{disk}} \geq b$. By J1 and J2, every decision in $(A_{\text{disk}}, A_{\text{pre-crash}}]$ is in the durable journal. Recovery replays it, and by J3 it recreates the same rows, outcome ledger, session state and watermark as before the crash. An acknowledgement was sent only after local application, which by J1 follows a durable decision. The acknowledged outcome is therefore reproduced. A process crash (SIGKILL) loses nothing, since WAL frames are already in the kernel's page cache. $\square$

J1 holds by construction: application follows the group's journal commit. J2 holds today: generations copy every record with slot $>$ base and sync the certified image; retirement never removes the active journal; within a generation the journal is append-only. J3 is the determinism assumption used for replica agreement. The embedded Engine API, where the database is the only record, keeps FULL. `check_storage` checks both databases at FULL during open and recovery. Only after recovery does `application_cache_mode` switch the separated application connection to NORMAL. The journal stays FULL.

This mechanism follows the same structure as `rqlite`, which runs SQLite unsynchronized and rebuilds it from the Raft log and snapshot. It is deliberately more conservative: `rqlite` uses `synchronous=OFF`, while SQLodin keeps NORMAL so that the file stays consistent after an OS crash and never needs a full rebuild.

8.5 M5: learning an owner's value from this voter's own vote

Under concurrency, a voter learns another owner's value only from that owner's Commit. That takes owner vote, this vote, the owner's decision barrier, a hop, and this voter's barrier. Every write also waits for the latest concurrent value below it. With a write quorum of two ($N \leq 3$), the owner's durable round-0 vote and this voter's durable round-0 vote for the same value **are** a write quorum. Once the group containing this voter's vote is durable, the value is chosen by definition. `owner_fast_commit` therefore steps the Commit after stepping the Accept, when the ledger shows this voter's vote at the same ballot and value. The Commit enters the same journal group, and application still follows that group's barrier. With larger quorums the rule is disabled.

`OwnedSkip.tla` adds `VoteLearn`. The `OwnedSkipLearnUnvoted` negative control learns from the Accept alone, without this voter's vote, and violates `LearnedAgreement`. On the three-process loopback cluster M5 made no measurable throughput difference; it is kept because it is proven, cheap, and shortens follower application for reads served there.

8.6 M6: quorum-frontier reads

A fresh read previously proposed a marker, so it paid the full write chain (about 21 ms here). Following Paxos Quorum Reads, a service cohort now closes its membership and records its own `highest_seen`. It sends a peer-only frontier request, and when `read_quorum - 1` peers have answered for that cohort it sets H to the maximum reported value. Members are answered once the applied prefix reaches H .

PROOF – Freshness and real-time order

A write acknowledged before invocation is chosen at some slot s , so a write quorum holds durable votes for s . The queried read quorum intersects it. `highest_seen` is at least every slot a voter has durably voted for or decided, never decreases in a process, and resumes above the durable ledger after restart. Every observation follows invocation, so $H \geq s$. For an earlier read, every state-changing slot it observed likewise has a voting quorum. An M2-only learned no-op need not have a voting quorum, but skipping its observation cannot change a SQL result. Freshness is a claim about application history, not every no-op frontier. $\square$

`QuorumRead.tla` checks this argument exhaustively for three voters and two slots. Its negative controls, peers reporting applied prefixes and answering without a peer, violate `RealTimeOrder`. Replies for another cohort or duplicate replies are ignored. Requests repeat every 200 ms, and a minority cannot complete a frontier. Optimistic transactions use the same barrier, since their proof needs only freshness (`specs/transaction-order.typ`).

8.7 Service I/O findings

Instrumented runs, with instrumentation not retained in the source, showed two further limits. First, peers exchanged at most eight frames per turn in each direction; with one frame per consensus message, this capped packets per turn. Links now drain until TLS would block. They receive until a 128-packet turn buffer is full, stepped in groups of sixteen within the turn, and the frame ring holds 256 entries, with the 2 MiB byte bound unchanged. Second, a full consensus window answered `Busy`, which made the client reconnect and fail over. A full window now leaves the unproposed request pending until its own timeout, while history-space pressure still answers `Busy`. Responses produced before a turn are flushed before its barrier rather than after it. None of these changes a durability or acknowledgement rule.

9 Invariants & Correctness Guarantees

Keep the fixed, authenticated, non-Byzantine membership and deterministic SQL assumptions from SOD 0003. M2 expands the learner's justification: an owner no-op is uniquely determined even before a voting quorum is known. It does not turn arbitrary `Accept` messages into decisions. M5 still requires the owner's durable vote and this voter's matching vote. Read-frontier quorums must count voter identities across reconnects, not connection objects. The old `DurableHistoryProof` quorum premise is not a literal invariant for every M2-only no-op; `OwnedSkip` determinacy supplies that additional seam. A complete mechanized composition remains unclaimed.

NORMAL application commits rely on the retained FULL journal and a durable generation base. A process kill alone does not simulate loss of the OS cache. A targeted regression restores an older checkpointed application prefix after acknowledgement while retaining the journal, then checks data recovery and retry deduplication. This tests the replay seam, not physical hardware.

10 Security & Operational Considerations

Use homogeneous peer capabilities. The hello fingerprint includes `frontier=1`, so a marker-only peer is rejected before service rather than accepted and disconnected when a frontier arrives. `Public next_id` returns an externally usable reservation: inside a turn it closes pending journal work and syncs the reservation before returning. This may add a barrier to an embedding caller; network frontier reads do not allocate these IDs. Keep maintenance outside an open turn.

11 Resource Budgets & Performance Targets

Keep the 64-slot protocol window, 16-request proposal/application groups and 2 MiB output-byte cap. SOD 0005 raises each output ring to 256 frames and introduces a shared 128-packet input buffer. Journal

groups flush before 1,024 records. Capacity pressure may add a sync barrier; it must not release effects early. The current limits are specified in `specs/resource-contract.typ`.

The original SOD 0004 throughput and p99 goals remain unchanged improvement objectives under the owner's accepted release disposition. This record neither lowers them nor turns the short paired run into a capacity guarantee. The 256 MiB dirty-tail threshold triggers maintenance; it is not a hard maximum on retained history or database size.

12 Validation and Acceptance Gates

- **Formal:** `OwnedSkip.tla` (TLC) checks agreement between fast-learned (M2 and M5) and chosen values, including three failed-voter cases under fairness. Three negative controls must fail: a revoker that offers any value when it finds no vote, an owner that forgets its round-0 vote, and learning a value without this voter's own vote. `JournalCache.tla` checks J1–J3; its negative controls apply before the journal commit and trim without a durable image. `QuorumRead.tla` checks M6, with the two controls above. `OwnedSkipProof.tla` (TLAPS) proves no-op determinacy inductively for unbounded ballots and values.
- **Implementation:** all Odin tests in debug, optimized and individual-commit configurations. New tests cover turn grouping (one barrier, nothing released early, empty turns write nothing), M2 and M5 fast learning, rejection of a relayed Accept, the synchronous mode of each database, quorum-frontier cohorts and a read at a voter that missed a write acknowledged by the other two. The existing after-journal-commit crash boundaries exercise replay of an application tail behind the journal. The full `tools/check.py` gate, network-service, transaction-history, CLI and Python checks, and the three-host checks, all run on the changed binary.
- **Performance:** the same `calibrate_native_mixed.py` matrix on `.18`, with the SQLite reference measured in the same run, with every run's report retained.

12.1 Recorded measurements and evidence

Review qualification: the original three-host worker did not propagate thread exceptions and verified a sum computed from its surviving writes. A worker failure could therefore inflate reported throughput and still pass verification. These historical figures remain observations from that harness; they are not independently verified completed-workload rates. The calibration harness is separate. No stored sample is discarded or silently relabelled as a corrected run.

All reports are in `benchmarks/results/sod-0005/`. Every sample carries its binary hash. The baseline binary is built from the unmodified parent revision, and baseline and candidate always alternate. Nothing was discarded; failed attempts are listed below.

12.1.1 Staged experiment (smallest test first)

Before the full implementation, each mechanism was added to a prototype build, one at a time. The tiny workload was 40 sequential writes, 40 sequential fresh reads and one 24-client write burst on the `.18` loopback cluster:

Build	Write ms	Read ms	Burst w/s
Baseline	46.4	50.1	97
1. M4 application cache	42.6	42.3	140
1. M2 no-op learning	34.0	33.6	116
1. M1/M3 one barrier per turn	21.0	20.9	226
1. M6 quorum reads	21.2	0.27	300

The single-client latency matched the three-barrier prediction. A negative result followed: enlarging peer I/O bursts (stage E) and adding M5 did **not** measurably raise loopback throughput. Instrumentation then showed no dropped packets, but each write spanning seven or more barrier times under load. The measured cause is sync latency, 8–14 ms at 512 B–256 KiB, independent of size, combined with three

voters sharing one ZFS intent log on .18. On that host, fsync latency also alternates between about 1 ms and 9 ms regimes.

12.1.2 Same-host matrix against same-run SQLite (.18)

calibrate_native_mixed.py, 100 operations per client, 256-byte values, SQLite FULL with groups of at most 16, measured in the same run:

Clients / reads	Baseline tx/s (ratio)	SOD 0005 tx/s (ratio)	Speed-up
1 / 70%	15.8 (1.7%)	165.5 (18.2%)	10.5×
1 / 0%	12.2 (10.5%)	46.3 (20.7%)	3.8×
8 / 70%	58.1 (9.4%)	314.1 (40.6%)	5.4×
8 / 0%	71.1 (8.4%)	181.0 (21.3%)	2.5×
32 / 70%	161.8 (5.5%)	534.3 (18.0%)	3.3×
32 / 0%	103.1 (3.7%)	194.9 (6.8%)	1.9×

12.1.3 Three hosts (.19/.20/.21), client on .18

tools/compare_three_hosts.py, 3 alternating repetitions, all replicas verified; medians:

Measure	Baseline	SOD 0005	Change
Sequential write latency	54.6 ms	19.3 ms	2.8×
Sequential fresh-read latency	55.5 ms	0.58 ms	95×
24 clients, pure writes	133 w/s	651 w/s	4.9×
24 clients, 70/30 mix	190 tx/s	519 tx/s	2.7×

With 32 clients (three-host-comparison-32.json, --clients 32), again 3 alternating repetitions with all replicas verified:

Measure (median)	Baseline (samples)	SOD 0005 (samples)	Change
Sequential write latency	44.3 ms	19.4 ms	2.3×
Sequential fresh-read latency	44.6 ms	0.51 ms	87×
32 clients, pure writes	94.8 w/s (78.5, 94.8, 157.4)	784 w/s (832, 784, 300)	8.3×
32 clients, 70/30 mix	269 tx/s (197, 327, 269)	755 tx/s (828, 755, 741)	2.8×

One candidate pure-write sample (300 w/s) is well below the other two. It is retained and not explained; the run did not isolate its cause.

12.1.4 Corrected-harness review measurement

One fresh alternating pair used 32 clients, voters .19/.20/.21 and the client on .18. Each binary completed 800 pure-write and 960 mixed operations. The mixed case contained 286 writes. Every replica's account rows, balances and payload lengths matched the independently planned work.

Measure	Baseline	Reviewed binary
Pure writes/s	164.8	791.5
70/30 transactions/s	284.6	628.3
Sequential write median, ms	52.4	19.3
Sequential read median, ms	54.2	0.6

This short paired run supports the direction of the improvement; it is not a sustained-capacity claim or a replacement for repeated distributions. Its JSON retains both binary hashes, the corrected harness hash, completed counts and durations. Historical reports above remain unchanged.

12.1.5 Remaining gaps

The original targets remain unmet in the pure-write and high-concurrency cases: 32-client pure writes reach 6.8% of SQLite on the shared-disk host, against the 25% goal. Separate hosts help (4.9×

clients and $8.3\times$ at 32 clients there), which is consistent with shared intent-log contention on .18. The irreducible chain is the owner's vote barrier, the voter's barrier and the owner's decision barrier, plus waiting for turn boundaries. Under mixed load a fresh read also waits for writes in flight below its frontier, the "rinse" of quorum reads, so mixed reads cost about one write latency. Candidates, each needing its own proof: overlapping the sync with network I/O on a sync thread; report frontiers based on decisions where M5 guarantees knowledge; and a larger window if Little's-law saturation appears.

12.1.6 Regression evidence

- `tools/check.py` passes in full on .18. It covers style and vet; 169 tests in each of the debug, optimized and individual-commit configurations; the upstream suite; durability crash boundaries; mixed-SQL replicas; process-cluster SIGKILL; contracts; 30 fault simulations of 150,000 steps; examples; benchmark smoke; and the CLI.
- These service checks pass on the final binary: network service (formats 4 and 5), transaction histories (two seeds, 16 rounds each), ORM transactions, majority and minority, session crashes, snapshot service and catch-up, admission, Python features and the CLI. A first attempt at the history, ORM, Python and CLI checks failed only because .18's system Python lacked SQLAlchemy, or because the shell path was wrong. Both attempts are listed in `check-summary.log`.
- The three-host qualification (`three-host.json`) passes all 19 checks, including each absent voter, SIGKILL convergence and certified generations surviving restart.
- The formal runners pass 72 TLC cases (59 existing, 13 new) and 89 TLAPS obligations (12 + 36 existing, 41 new).

13 Alternatives Considered

Alternative	Disposition
Owner sends Accept in parallel with its own sync	Saves one d , but violates the upstream durability gate and P2 above. Would need an upstream proof change. Deferred.
Apply before the owner's decision barrier	Saves one barrier for the owner's own writes, but lets application state become durable ahead of its decision evidence, which recovery rejects. Rejected.
Owner self-learning of its own no-ops	Not needed for the measured chain; the owner learns through replies. Not implemented.
synchronous=OFF for the application database	Allows corruption after OS crash, which would force a rebuild. Rejected in favor of NORMAL.
Dedicated sync thread with pipelined turns	Overlaps the sync with network I/O. Larger change to a single-owner design. Revisit if the turn rate saturates after M1.
Larger window W	Only if Little's-law saturation is measured.

14 Open Questions & Future Milestones

Performance work still needs to distinguish device synchronization, owner-turn scheduling and window saturation. The unexplained low sample remains unexplained; tenant contention is a hypothesis, not a measured attribution. Qualification of this post-release host remains separate from the earlier candidate. This review introduces targeted regressions, not new soak or capacity gates.

15 Discussion and Revision Notes

25 September, implementation review: retained M1–M6 and corrected two host boundaries: a reconnected peer must not count twice in one read quorum, and a returned ID must survive an aborted enclosing turn. Added a five-voter reconnect model with a connection-counting negative control and a replay test that deliberately loses an acknowledged application suffix. The five-voter seam test is not a five-voter deployment qualification.

The comparison worker now propagates every thread failure, records planned and completed counts, and checks every account row and payload length on each voter. Fault-injected worker tests reject failed reads, failed writes and equal-total but incorrect row data. Existing measurements remain untouched. The missing SOD 0005 bundle inclusion and stale storage-mode explanation were corrected. The record stays In Discussion; these fixes do not invent a publication or release decision.

Fresh review evidence is in `benchmarks/results/sod-0005/review/review.json`. Three focused regressions pass locally and on Linux; the two bug regressions fail against the submitted code. Eight targeted TLC cases pass, including required negative controls. Four benchmark-worker tests pass. The reviewed Linux binary passes native mTLS checks and bounded serial transaction histories with each voter absent and after full restart. The corrected three-host comparison is recorded above. The first two history attempts lacked SQLAlchemy; their failed reports remain beside the passing run using cached test dependencies. The expensive unoptimized full-suite compilation was stopped before execution; this review does not claim a fresh full-gate pass. The earlier 169-test results remain attributed to the submitted revision, not silently reused for the review fixes.

16 References

- Y. Mao, F. Junqueira, K. Marzullo. Mencius: Building Efficient Replicated State Machines for WANs. OSDI 2008.
- A. Charapko, A. Ailijiang, M. Demirbas. Linearizable Quorum Reads in Paxos. HotStorage 2019.
- P. O'Toole. rqlite design notes on SQLite WAL, synchronous and Raft-log recovery.
- SQLite documentation: PRAGMA synchronous and WAL durability.