

R6 service admission batching

The service collects complete, validated write requests already owned by its connection pool. After processing peer input, `admit_writes` submits at most sixteen through the existing durable `propose_batch` wrapper. It adds neither a coalescing timer nor an additional dynamically growing queue. Connections hold one outstanding request each; the existing 24-client admission bound remains. Round-robin selection advances after admission or an unsubmitted Busy response. A fast low-index connection cannot permanently take precedence over waiting higher-index connections.

The pinned ownership implementation checks every target slot before changing the batch. On window pressure the service halves the attempted prefix until it fits or even one request cannot be proposed. A full window is transient: the unsubmitted request stays pending, and the existing timeout answers Busy if it is still unproposed. History-space pressure, which needs maintenance, answers Busy at once (`durable.backpressure_transient`). Other bounded pending requests remain eligible on subsequent turns. A timeout before any slot assignment is also Busy; after slot assignment it remains Unknown_Outcome. A displaced submitted request retains the existing retry path and identity. Storage failure stops service.

Ordering and durability argument

Requests from different outstanding client calls overlap and may be ordered by batch admission. Each connection accepts one request at a time. A write response still comes only from `poll_write` after the durable outcome and application prefix exist. Consequently a completed earlier write precedes a later invocation; admission does not change the real-time order premise in the transaction/read argument. Read barriers retain their existing fresh, single-use semantics.

The durable wrapper validates the complete batch, obtains distinct slots through the whole pinned library and calls `finish` before releasing dependent packets. It returns proposal slots, not acknowledgement authority. Inside a service turn the proposal joins the turn's single journal barrier (SOD 0005 M1); nothing is released before it. Per-request savepoints, reference fallback and retry identity handling remain unchanged. A crash before the response is resolved by replay and the same session identity, including a crash after durable batching but before the service stores the returned slot numbers in its volatile connections.

The window-pressure regression fills the owner window, checks shrinking and that unproposed requests stay pending without premature responses, then drains the quorum and checks every admitted outcome. The fairness regression makes an early connection return while later connections wait and verifies the later connections receive earlier slots. Native transaction histories, minority/retry/restart checks and the before/after calibration qualify the integration. Benchmark results must identify the source and binary; the existence of batching alone is not a throughput claim.

Bounded peer draining

Authenticated peer connections receive complete consensus frames until the turn's 128-packet buffer is full; the turn steps them in sixteen-packet durable groups inside one journal transaction. A partial frame, unavailable input or snapshot control stops the burst. Clients retain one-frame dispatch. Peer output drains until TLS would block, bounded by the 256-frame and 2 MiB per-connection limits. Responses are flushed before the turn's barrier. The service still persists each turn before releasing its dependent packets. SOD 0005 measured that the earlier eight-frame bursts, not bytes, bounded the per-turn packet rate. This amortizes durability boundaries; it does not defer or remove them.

Measured calibration and remaining target gap

The measurements in this section describe the release candidate before SOD 0005. The post-SOD-0005 measurements and their same-run SQLite ratios are recorded in `docs/sod/records/0005-durable-turn-and-fast-skip-learning.typ` and `benchmarks/results/sod-0005/`.

The retained `workload-matrix-analysis-linux.json` binds 32 case reports to `workload-matrix-linux.json`. All 91,840 native operations complete without an unknown outcome or operation error and agree with the serial SQLite reference. This establishes the finite tested histories, not the provisional throughput goals. Both engines use pinned SQLite with FULL durability and at most sixteen requests already waiting per commit. The single-copy reference has no replication or TLS.

At 32 clients, 70 percent reads, 256-byte values and one statement/changed row, SQLodin completes 152.3 transactions/s, including 46.4 writes/s; matched SQLite completes 2,971.8 transactions/s. The ratio is 5.13 percent. Native read/write p99 is 397.1/471.8 ms. The 32-client pure-write case completes 73.6 writes/s against 2,845.0 for SQLite, with 1,230.8 ms p99. These finite samples miss the original 3,000 mixed/s, 900 mixed writes/s, 1,000 pure writes/s, 20/50 ms p99 and 25 percent relative-throughput goals. The scheduled-arrival cases also miss their targets; queueing time is included. Earlier tiny-run peaks are not sustained-rate claims.

Across the matrix, measured aggregate CPU is 0.087–0.572 cores and peak voter RSS is 21,434,368 bytes. Mean forwarded sync latency ranges from 5.5 to 21.2 ms. The extended 25,600-operation profiles attribute roughly 78–83 percent of each voter's workload elapsed time to accumulated sync time. Those forwarding counters include process setup, validation and shutdown, so they are attribution evidence, not an exact per-request critical-path decomposition. Low CPU usage together with sync cost supports investigating durable scheduling and commit amplification; it does not establish that a faster language, additional cores or a new consensus algorithm would meet the goals.

These measurements are from three processes sharing the benchmark instance's ZFS-backed storage. Native multi-host correctness runs are separate. Available observations suggest shared resources between benchmark instance .18 and voter instance .21; physical failure-domain independence and pool device mapping are unverified. The TCP connection calibration is not a packet RTT measurement. On 25 September the user approved a first release after correctness qualification with these performance shortfalls disclosed. The original throughput, p99 and relative-throughput numbers remain future improvement goals; none is relabelled as achieved. A release decision must distinguish correctness qualification from an achieved performance target. Resource, capacity and recovery observations remain separately reported under R6.3.

A small local pinned-SQLite cache check (`page-cache-hotset-local.json`) compares 2 MiB and 64 MiB page-cache settings over an 8 MiB hot payload and four copies. The larger cache eliminates reported cache misses after the first copy, but total elapsed time is essentially unchanged (0.271 versus 0.276 seconds). This local fixture does not establish a distributed throughput improvement or justify a new cache setting. Runtime cache policy remains unchanged; a counter reduction alone is not an accepted performance result.