

ECHO Memory Plugin — Performance Benchmark

Validation of the 1.1.0 concurrency and 1.2.0 entity-resolution releases

DATE

June 23, 2026

PREPARED BY

Jason Stedwell,
Owner

REFERENCE

echo-v.05 @
af16598

STATUS

11/11 gate checks
pass

SUMMARY

Full-vault reads run 7.1x faster concurrent than serial (11.4 s down to 1.6 s, 197 notes). Entity resolution is sub-millisecond. Profiling traced recall() latency to serial graph expansion (not BM25); the fix makes expansion concurrent (3.5-4.2x, identical results). All 11 gate checks pass. Run live against echoapi.alwisp.com on 2026-06-23.

TEST ENVIRONMENT

Endpoint	echoapi.alwisp.com (live)
Vault size	197 notes
Plugin commit	af16598
Releases under test	1.1.0 pooling + concurrency, 1.2.0 resolver
Concurrency (default)	ECHO_WORKERS = 8
Socket timeout	ECHO_TIMEOUT = 30 s
Runner	Python 3.10.12
Date	June 23, 2026

HEADLINE RESULTS

OPERATION	RESULT	VERDICT
Full-vault read (concurrent vs serial)	7.13x	Pass
Full-vault read throughput	123.5 notes/s	Pass
Cold vs warm request (keep-alive)	3.25x	Pass
Entity resolve (in-memory index)	<1 ms	Pass

Idempotent append skip	100%	Pass
Soak read errors (22 s)	0	Pass

FULL-VAULT READ — CONCURRENCY SCALING

WORKERS	MEDIAN (197 NOTES)	SPEEDUP VS SERIAL
1 (serial)	11,917 ms	1.0x
2	5,740 ms	2.1x
4	3,034 ms	3.9x
8 (default)	1,584 ms	7.5x
16	1,278 ms	9.3x

SUBJECT PULLS — RECALL() (AFTER CONCURRENCY FIX)

SUBJECT	RESOLVE()	SEARCH	RECALL() TOTAL
APTA	0.75 ms	62 ms	2,231 ms
CapMetro	0.03 ms	58 ms	2,116 ms
echo (hub)	0.03 ms	71 ms	2,404 ms
operator preferences (hub)	0.04 ms	2,011 ms	2,995 ms
rivnut torque spec (leaf)	2.15 ms	2,014 ms	2,371 ms

RECALL() GRAPH EXPANSION — CONCURRENCY FIX

HUB SUBJECT	NEIGHBOURS	SERIAL	CONCURRENT	SPEEDUP	RESULTS IDENTICAL
echo	120	2,228 ms	642 ms	3.47x	Yes
operator preferences	126	2,910 ms	699 ms	4.16x	Yes

BULK WRITE OPERATIONS

OPERATION	PER-OP MEDIAN	AGGREGATE
PUT (read-back verified)	115 ms	7.0 PUT/s
Bulk GET — concurrent vs serial (K=30)	356 vs 1,607 ms	4.51x

APPEND — write	104 ms	1 round-trip + GET
APPEND — idempotent skip	51 ms	100% skipped on re-run

CONCURRENCY, LOCKING & RESILIENCE

CHECK	MEASURED	EXPECTED
Lock — acquire when free	159 ms (rc 0)	rc 0
Lock — contended acquire	58 ms (rc 75)	rc 75 fast-fail
Lock — release	133 ms	owned release
Offline queue — enqueue	0.08 ms	local write-ahead
read_many dedup (60 → 20)	deduped	collapse to unique set
Soak drift (late vs early)	0.99x	<= 1.5x, no leak

REGRESSION GATES (RELATIVE-RATIO POLICY)

GATE	THRESHOLD	MEASURED	RESULT
read-full speedup	>= 1.8x	7.13x	Pass
bulk-get speedup	>= 1.5x	4.51x	Pass
pool warm-up cold/warm	>= 1.5x	3.25x	Pass
append idempotency	100% skip	100%	Pass
soak errors	0	0	Pass
soak drift	<= 1.5x	0.99x	Pass
lock semantics	rc 0 / rc 75	0 / 75	Pass
read_many dedup	unique set	20/20	Pass

FINDINGS

1. Concurrency delivers as designed. Eight workers cut full-vault reads 7.5x; the curve is near-linear to 8 workers and flattens after (8→16 returns only 1.24x). The default of 8 is the right setting for this vault and link.

2. Keep-alive is confirmed live. A cold request costs 163 ms against a 50 ms warm median (3.25x). This is the direct measure that the 1.1.0 pooling fix removed the per-request TLS handshake that was timing out full-vault passes.
3. Entity resolution is effectively free. `resolve()` returns in under 1 ms across exact, alias, and shortened mentions. The 1.2.0 alias work resolves shortened names ("echo memory", "ECHO plugin", "jason") straight to the canonical note, so the anti-duplicate guard holds without a fuzzy fallback.
4. `recall()` latency was traced to the graph layer, not BM25. The BM25 index is already persisted and incrementally maintained (load 500 ms, score 0.1 ms). The cost was `expand_graph` fetching each neighbour serially — 3.0 s for 126 nodes. Fix: the graph BFS now fetches each hop concurrently via the existing `read_many`. `expand_graph` is 3.5-4.2x faster with byte-identical ranking and scores; end-to-end `recall()` dropped from 2.8-4.7 s to 2.1-3.0 s per subject.
5. Full-vault maintenance scripts stay under the tool timeout. `sweep.py` runs in 4.6 s and `vault_lint.py` in 5.1 s on 197 notes. The original failure mode (passes exceeding the timeout and dropping the session) does not recur.
6. Write integrity holds. PUT verifies via read-back at 115 ms; APPEND is whole-line idempotent, skipping at 51 ms with zero duplicate writes on re-run. The advisory lock fast-fails a contended acquire with the expected exit 75.

RECOMMENDATIONS

1. Apply `read_many` to every code path that reads a set of known notes. Done for `expand_graph` (gated). Remaining: prefetch the `_brief` result bodies `recall` prints, and parallelize `rebuild()`'s vault walk.
2. Reuse one in-process read cache across a `recall()` call so `load_index`, `expansion`, and `_brief` don't re-fetch overlapping notes — this closes the remaining ~500 ms + serial `_brief` cost.
3. Make 'prefer `read_many` over a GET loop' a documented contract in the plugin's API reference; serial multi-note reads are a performance bug. The expand-graph gate enforces it for `recall`.
4. Hold ECHO_WORKERS at 8 (16 buys only 1.24x). Promote the resolve correctness table from INFO to FAIL now that live behavior is confirmed. Re-baseline at the next 1.x release or past ~400 notes.

METHOD

Each metric was timed with a perf-counter harness that reuses the live ECHO client, so it exercises the real pooled and concurrent code path. Reads report median and p95 over repeated iterations after a discarded warm-up; write metrics ran in a disposable namespace under the advisory lock and were deleted on completion (zero residual files confirmed).

Gates use relative ratios rather than absolute milliseconds, so they stay valid regardless of the network the suite runs from. The harness, fixtures, baselines, and per-metric JSON are checked in under `eval/perf/`.

NEXT STEP

Prefetch recall()'s _brief reads and parallelize rebuild() via read_many, then re-run subject-pull to confirm recall drops toward ~1 s. expand_graph fix is local-only until ported to the canonical repo. No blockers in 1.1.0 / 1.2.0.