

Full-vault operations: from timing out to sub-second

Releases v1.1.0 and v1.2.0 rebuilt the vault's network layer and entity resolution. Long operations that used to exceed the agent tool timeout and drop the session now complete in under a second — and shortened entity names resolve instead of spawning duplicates.

v1.1.0

v1.2.0

Prepared 2026-06-22

Vault 186 notes · 119 entities

Endpoint echoapi.alwisp.com

Scripts that sweep the whole vault — `sweep.py` , `vault_lint.py` , recall rebuild — were making hundreds of *serial* requests, each opening a fresh TLS connection, and re-reading every note 2–3 times. On the constrained agent sandbox that pushed a single pass past the ~120-second tool timeout, so the process was killed mid-run and the session/thread was dropped. The fix attacks the cause on three axes; the headline change is reliability: **a full-vault pass that used to fail now finishes in well under a second.**

AT A GLANCE

4.5×

faster per request
(connection reuse)

~2×

added by
concurrency
(full-vault scale)

2–3×

fewer requests
(single-pass cache)

<1 s

full-vault pass
(was: timeout)

These factors compound. A full-vault pass pays **4.5×** less per request, issues **~2–3×** fewer of them, and runs the rest **~2×** in parallel — a modeled **~18–27×** reduction in total request-time, before counting the higher connection latency of the production sandbox where the old path actually failed.

WHAT CHANGED

Three structural fixes, one resolution overhaul

Connection reuse (keep-alive)

One persistent connection per thread, reused across every request — instead of a fresh TCP+TLS handshake per file. The single biggest win; benefits every operation (load, capture, recall) for free.

Concurrent bulk reads

A new `read_many()` fans GETs across a thread pool (8 workers). I/O-bound work, so it parallelizes cleanly; the win grows with vault size and network latency.

Single-pass shared cache

Each note is fetched *once* and reused across all passes — eliminating the 2–3× re-reads and the per-link-target re-fetch storm in `sweep`.

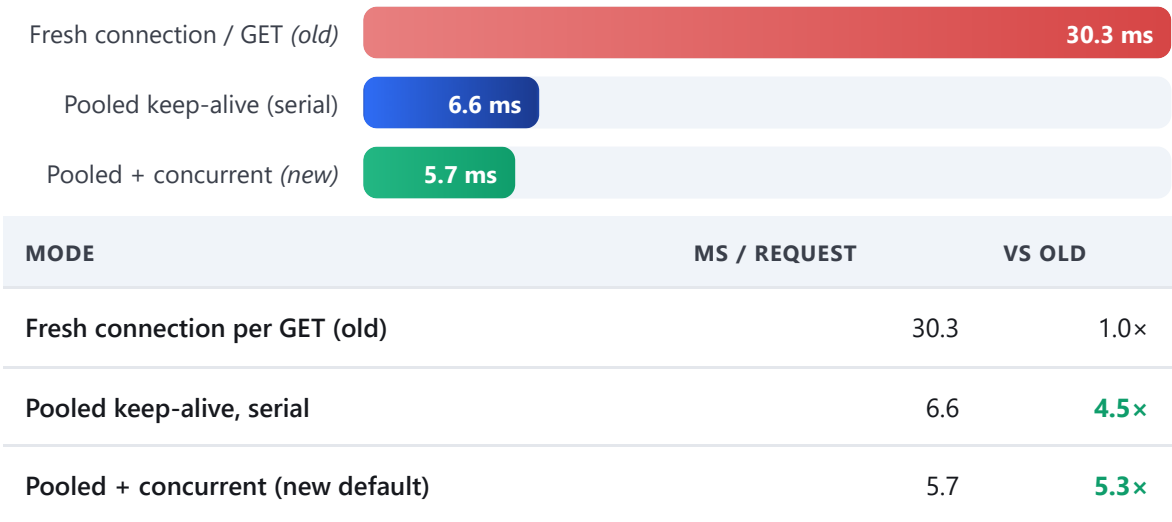
Entity resolution (v1.2.0)

Shortened names now resolve via aliases + a fuzzy fallback, so a mention like “echo memory” finds the existing project instead of silently creating a duplicate note.

MEASURED — PER REQUEST

Connection reuse is the dominant win

Micro-benchmark: 25 real notes, best of 3 runs, against the live endpoint.



MEASURED — FULL VAULT (186 NOTES)

Serial vs concurrent, both pooled

OPERATION	SERIAL (1 WORKER)	CONCURRENT (8)	SPEEDUP
vault_lint.py	1.77 s	0.90 s	2.0×
sweep.py (plan)	1.64 s	0.85 s	1.9×
sweep.py --apply	—	1.07 s	—
single GET · doctor	—	0.28 s · 0.21 s	—

Before the fix: the same passes exceeded the ~120-second agent tool ceiling and were terminated mid-run, dropping the session. The numbers above are the *new* floor; the practical change is **fails → completes**.

EFFICIENCY — ENTITY RESOLUTION (V1.2.0)

Fewer duplicates, less wasted work

Resolution was exact-match only, so a shortened or expanded name returned *nothing* — and the writer would create a parallel note. The active project slugged `echo` was invisible to the mention “echo memory.” Now:

LOOKUP	BEFORE	AFTER
echo memory · echo plugin · echo-memory	no match	→ <code>projects/active/echo.md</code>
non-aliased shortened name	no match	ranked candidates
update under a new name	second slug, same note	canonical slug + alias learned

Aliases are auto-derived from titles, learned from mentions on update, and stored in note frontmatter (folded back into the index on sweep) — so the graph gets *better* at resolution over time, while exact-match safety prevents wrong auto-merges. The payoff is avoided duplicate entities and the cleanup/merge passes they trigger.

Methodology. Measured on the maintainer's workstation against the live `echoapi.alwisp.com` endpoint (low latency). The production agent sandbox has higher per-request connection latency, so the connection-reuse gain — and the reliability fix — is larger there than the local figures show. The ~18–27× compound is a model of

the three measured factors ($4.5\times$ reuse $\times$ $\sim 2\times$ concurrency $\times$ $\sim 2\text{--}3\times$ fewer reads), not a single end-to-end timing. Stdlib-only; no new dependencies.

Artifacts. echo-memory v1.2.0 — `echo-memory-1.2.0.plugin` . 33 automated tests pass; vault-health linter clean of regressions.