Scale Characteristics of an Append-Only Decision Log

kgai maintainers

kgai.dev · team@kgai.dev · August 2026

ABSTRACT

kgai records a software team's engineering decisions as an append-only, content-addressed log, split into one shard per writer and projected into a small live graph of domain elements. This report examines how that design behaves as the log grows. We summarize the published benchmark, in which a store holding 1,000,000 decisions across thirty writers' shards answers a decision lookup in about 100 ms, and we state its scope precisely, because the figure describes the decision lookup specifically and recall and free-text search are slower. We describe the benchmark method as documented in the repository, analyze the storage growth of a log that never deletes, examine the cost of rebuilding the projection after synchronization, and discuss the current free-text search path, which examines stored decision text rather than consulting a prebuilt index. We close with the measurements we do not have and the conclusions the published numbers cannot support.

1. Introduction

Any system that records history by accumulation eventually faces the same question. What happens when the log gets long. kgai records a software team's engineering decisions, including the rejected alternatives, as an append-only log that is never rewritten, and derives from it a small graph of domain elements that coding agents consult before changing code. Because nothing is ever deleted, growth is the intended steady state rather than an edge case, and the scale behavior of the design is a fair thing to ask about.

This report examines that behavior. Its centerpiece is the published benchmark, in which a single store holding 1,000,000 decisions across thirty writers' shards answers a decision lookup in about 100 ms. We state the scope of that figure precisely, because scope is the part most easily lost in summary. The figure describes the decision lookup specifically. Recall and free-text search are slower, and we discuss why in Section 6 without attaching a number to them, since the published record states those paths in words.

We are the maintainers of the system under discussion. This is an industry research report on our own design, not an independent evaluation, and Section 7 lists what our numbers cannot support.

2. System Model

kgai separates one log from two views of it. The source of truth is a decision log, append-only, content-addressed, and stamped with logical clocks in the manner of Lamport [1], stored as newline-delimited JSON in which every writing installation keeps its own shard. A decision is an immutable event recording author, rationale, and date alongside a list of structural mutations. The log replays into a derived read model split across two planes. The live element graph holds the current shape of the domain, a deliberately small and stable set of elements and links. The decision plane holds every decision ever recorded, linked to the elements it shaped and to the decisions it superseded.

Replay is deterministic. Events are totally ordered by logical clock value with the content hash as tiebreaker, every projection write is idempotent, and two stores that replayed the same events produce identical graphs, verified by comparing digests of the canonical export. The design follows a familiar lineage. Treating an immutable log as the authoritative record and everything queryable as a derived structure is the event-sourcing posture described by Kleppmann [2], the storage layout echoes log-structured systems [3, 4], and the refusal to update anything in place follows the argument that immutability simplifies coordination at a distance [5].

The consequence that matters for scale is architectural. The thing that grows without bound, the decision plane, is separated from the thing most queries traverse, the live graph, which grows only with the domain vocabulary of the team. The event model, the ordering, and the projection are specified by the opening reports of this series (TR-2026-01, TR-2026-02, and TR-2026-03).

3. Method

The benchmark method is documented in the repository, and the raw runs are archived there, so the numbers below can be inspected rather than taken on trust. The largest archived run populates one store with 1,000,000 decisions distributed across thirty writers' shards, which is the configuration behind all published figures. The synchronization transport was exercised separately against both MinIO and production S3, covering concurrent writers, write-once arbitration races, and detection of copied stores that would otherwise fork history silently.

The benchmarking literature is blunt about how results mislead, through unnamed workloads and summary statistics detached from their setup [6, 7]. We therefore attach the store shape to every figure we quote and name the query each figure belongs to. All published figures come from this one store shape. We have not published a scaling curve across store sizes or shard counts, and we do not extrapolate beyond the measured point.

4. What Stays Fast

The published figure is a decision lookup answering in about 100 ms against the 1,000,000 decision store, described in the published material as flat to one million. The published comparison table reports element

history in the same band. Both operations share the property that explains the flatness. They address their targets directly. A decision is content-addressed, and an element's history is the ordered chain of decisions linked to that element in the projection, so answering means retrieving a bounded neighborhood of the graph rather than scanning the log. Replay is never on the read path for these queries, since history exists in the projection as precomputed structure.

The second published result at this scale is not a latency but an equality. Independent clones of the store replay to byte-identical graphs, with sha256 digests of the canonical export matching regardless of the order in which events arrived. At one million events this determinism is more than an aesthetic property. It is what allows a latency measured on one machine to describe a graph that is provably the same graph on every other machine that holds the log.

5. Storage Growth and Rebuild Cost

An append-only log never deletes, so storage grows monotonically with the number of decisions recorded. The shards are plain-text newline-delimited JSON, readable and greppable without the engine. We publish no bytes-per-decision figure and therefore offer no disk-usage projection, only the qualitative statement that growth is proportional to events recorded and that superseded decisions occupy storage indefinitely. That cost is the price of the system's central property, that the history, dead ends included, remains queryable.

The projection is disposable by design. Synchronization is followed by a complete rebuild of the graph in canonical order, never an incremental patch, because a teammate's newly pulled events can precede already projected ones in the canonical order, and last-writer-wins property mutations would otherwise diverge between installations. The documented rationale for accepting a full rebuild is that the live element graph is small by design. We publish no rebuild latency figure. The same gap applies to as-of queries, which answer questions about past structure by replaying everything recorded up to the queried date into an ephemeral graph. Both are honest holes in the published record and appear again in Section 7.

6. What Does Not Stay Fast

Recall and free-text search are slower than decision lookups. That sentence is part of the published record and belongs in any faithful summary of the benchmark, so we repeat it here without qualification and without a number. Publishing a latency figure for a path we consider unfinished would invite exactly the context-free quotation the benchmarking literature warns against [7].

The reason is structural. Search compares the words of a query with element names and decision texts by overlap, deterministically and with no embeddings anywhere in the pipeline, so one store answers identically on every machine. The current implementation, however, maintains no index over that text. A free-text query examines stored decision text rather than consulting a precomputed structure, and its work

grows with the store in exactly the way the decision lookup's work does not. The standard remedy for this class of problem is well understood in the retrieval literature, an inverted index over the text [8]. The project roadmap carries a contextual-search index for stores that grow beyond roughly 100,000 decisions. A roadmap entry is a statement of intent, not a schedule, and we promise nothing about it here.

Until such an index exists, the accurate two-sided summary of kgai at scale is this. The addressed queries, decision lookup and element history, hold flat to a million decisions. The scanning queries, recall and free-text search, do not.

7. What We Do Not Measure

The published record is deliberately narrow, and the following gaps bound what it can support. We publish no latency figures for recall or free-text search, only the qualitative statement that they are slower. We measure one store shape, 1,000,000 decisions across thirty shards, and publish no scaling curve across store sizes and no sensitivity analysis across shard counts. We publish no rebuild time, no as-of replay time, no storage footprint, and no memory figures. The archived runs record their own conditions, but we have published no cross-hardware study, so the roughly 100 ms figure should be read as evidence of the shape of the curve rather than as a guarantee for any particular machine.

Finally, the benchmark was designed and run by the maintainers of the system it measures. The raw runs are archived in the repository precisely so that this does not have to be taken on faith, but archived self-measurement is not independent replication, and we know of no third-party reproduction to date.

8. Conclusion

An append-only decision log grows forever by definition, and the design question is what that growth is allowed to touch. kgai's answer is separation. The log grows without bound, while the live graph that queries traverse stays proportional to the team's domain vocabulary, and the published benchmark reflects the split. A store of 1,000,000 decisions across thirty writers' shards answers a decision lookup in about 100 ms, replays to byte-identical graphs on independent machines, and pays for that with storage that only grows and with recall and free-text search paths that are slower and unindexed today. The measured claims are narrow, one store shape and two fast query classes, and this report has tried to keep them exactly that narrow.

References

[1] L. Lamport. 1978. Time, Clocks, and the Ordering of Events in a Distributed System. Communications of the ACM 21(7), 558-565.

[2] M. Kleppmann. 2017. Designing Data-Intensive Applications. O'Reilly Media.

[3] P. O'Neil, E. Cheng, D. Gawlick, and E. O'Neil. 1996. The Log-Structured Merge-Tree (LSM-Tree). Acta Informatica 33(4).

[4] M. Rosenblum and J. K. Ousterhout. 1992. The Design and Implementation of a Log-Structured File System. ACM Transactions on Computer Systems 10(1).

[5] P. Helland. 2015. Immutability Changes Everything. ACM Queue 13(9). https://queue.acm.org/detail.cfm?id=2884038

[6] J. Gray (ed.). 1993. The Benchmark Handbook for Database and Transaction Processing Systems, second edition. Morgan Kaufmann.

[7] P. J. Fleming and J. J. Wallace. 1986. How Not to Lie with Statistics: The Correct Way to Summarize Benchmark Results. Communications of the ACM 29(3).

[8] J. Zobel and A. Moffat. 2006. Inverted Files for Text Search Engines. ACM Computing Surveys 38(2).