TECHNICAL WHITE PAPER

Designing a Memory-Mapped Cache Engine

Predictable recovery, bounded resource use, and production-safe indexing

Chandrakanth Anne

Review Draft • August 2026

Abstract

High-throughput cache engines are often evaluated only by steady-state latency, yet their hardest engineering problems appear at restart, during mutation bursts, under compaction, and at the boundary between corrupted and rebuildable state. This paper presents the architecture of an independently engineered, memory-mapped cache prototype designed for typed documents, exact and range queries, bounded heap use, predictable recovery, and fail-closed publication. The central idea is to keep immutable data in read-only mapped generations, constrain mutable state to a recoverable suffix, and make a checksummed manifest the sole authority for what is visible.

> **CORE THESIS** Performance is not a special fast path added after correctness. The durable layout, recovery protocol, index representation, and operating limits must all be designed as one system.

1. The Design Problem

The target workload combines a high proportion of point reads with a smaller stream of puts and deletes, while the canonical data set and indexes may be larger than the process heap. A practical design must ingest at least one million typed records, validate large collection catalogs at startup without rebuilding heap-wide maps, and preserve a direct single-server data path. Because the system is a cache rather than the source of truth, it may acknowledge a bounded asynchronous suffix that can be lost after machine failure; it must never expose a partial mutation, the wrong record, a mixed file generation, or an unchecked offset.

1.1 Engineering goals

- Bound heap usage independently of the total number of records and indexed terms.
- Keep immutable record and index files directly readable through memory maps.
- Recover by validating complete frames and replaying only a bounded mutable suffix.
- Separate authoritative data from secondary structures that can be quarantined and rebuilt.
- Make backpressure, disk headroom, query ceilings, and maintenance admission explicit.
- Collect reproducible p50, p95, p99, p99.9, RSS, mapped-byte, restart, and error evidence.

2. Architecture Overview

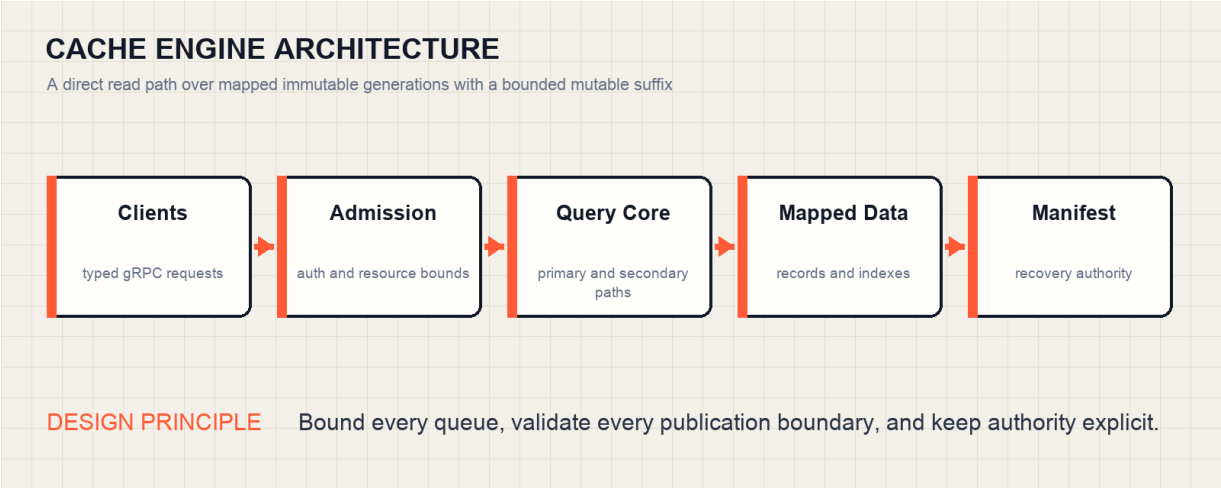

Figure 1. The request path stays direct while storage visibility is controlled by validated manifests.

Each collection owns a short commit sequencer, a bounded mapped recovery generation, and one or more immutable mapped generations. Requests are validated and encoded outside the sequencer. The sequenced section appends one complete recovery frame, updates the mapped primary delta, and then advances the visible sequence. Background flush advances a separate durable sequence. This split makes the asynchronous durability contract observable while keeping CPU-heavy parsing and encoding away from the serialization point.

3. Portable, Self-Validating Storage

Every file begins with a pointer-free, little-endian header and uses relative 64-bit offsets. Bounded blocks carry CRC32C checksums; authoritative manifests bind referenced complete files with SHA-256 digests. Published files are immutable and mapped read-only, and the storage layer owns the file handle for the full mapping lifetime. That ownership rule is essential on platforms where mapped-file retirement and deletion semantics differ.

3.1 Canonical records

JSON or another interchange representation is decoded against the collection schema and rewritten into a canonical record layout before commit. The layout stores stable field identifiers, fixed- and variable-width regions, document version, commit sequence, tombstone state, and absolute expiration. The record checksum is verified before a typed view is returned. Canonical bytes then become the common identity used by primary and secondary indexes.

3.2 Publication protocol

1. Write a complete new generation to a restrictive, same-directory temporary path.
2. Flush the content, finalize its checksums and digest, and rename to a never-reused generation name.
3. Create and validate the next manifest that references the complete authoritative bundle.
4. Publish the manifest and swap the in-process generation view only after validation succeeds.
5. Treat unreferenced output as an orphan that can be reclaimed; never infer authority from directory contents.

4. Mutation and Recovery

The mutable write path combines an append-only recovery log with a mapped open-addressed primary delta. A frame contains the complete key and canonical record or tombstone, which makes the delta derived state. Recovery scans to the last complete frame, ignores a torn final frame, rejects interior structural or checksum corruption, and rebuilds the primary delta idempotently.

> **VISIBILITY RULE** A mutation is not visible until the record append and primary reference update have completed. A process crash may recover a response-lost mutation, but readers never observe a half-published operation.

4.1 Sealing and write backpressure

Before a mapped log, table capacity, configured load factor, or probe bound is exceeded, the active generation is sealed into immutable record, primary-index, and document-table files. When the system cannot reserve mapping capacity or disk headroom, it rejects additional writes with a stable resource error while preserving safe reads. This is preferable to silent heap growth or an unbounded maintenance backlog.

5. Primary and Secondary Query Paths

5.1 Exact-key lookup

The primary index uses seeded 64-bit hashing, linear open addressing, full canonical key verification, a published load ceiling, and a bounded probe count. The current delta is checked first, then immutable generations from newest to oldest. A tombstone terminates lookup. Stable document identifiers are never reused, and physical offsets remain local implementation details rather than distributed identities.

5.2 Equality and membership

Immutable equality indexes separate a sorted term dictionary from postings. The builder selects an encoding based on local density: inline document identifiers for very small sets, delta-varint lists for sparse sets, containerized bitmaps for clustered sets, and base-relative dense bitmaps when span density justifies them. Boolean AND evaluates smaller candidates first; OR unions; NOT subtracts from a bounded live universe. Every candidate is revalidated against the newest authoritative record, so stale secondary entries can increase work but cannot change correctness.

5.3 Range, TTL, and pagination

Numeric, decimal, date, and timestamp values use sortable byte encodings. Sealed generations store fixed-width sorted range runs, while the active generation appends bounded add/remove markers. TTL is an absolute timestamp enforced immediately on every read; cleanup later emits ordinary tombstones. Pagination uses stable document ordering and a stateless cursor bound to the collection, schema, query, projection, and last document identifier. Integrity checks reject tampered or cross-query cursors.

5.4 Correlated query batches

A server-streaming batch API accepts bounded, independently identified queries and emits chunks in completion order. Per-batch and process-wide semaphores bound parallel work. A bounded response channel propagates receiver backpressure, item deadlines include scheduling and execution, and disconnects abort or discard remaining work. Correlation never depends on arrival order.

6. Compaction Without Invalidating Readers

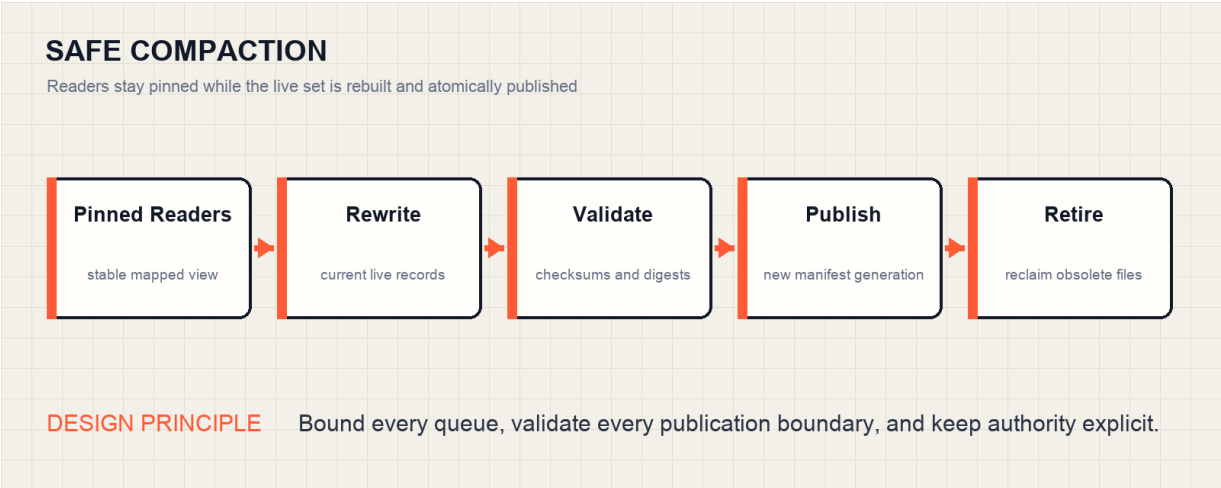

Figure 2. Manifest publication is the visibility point; reclamation occurs only after pinned readers release old mappings.

Append-only generations make foreground mutation inexpensive, but obsolete records, tombstones, expired values, and stale postings must eventually be removed. Full-tier compaction rewrites only the current live set into one replacement generation. Readers retain shared, pinned views during the long rewrite. Mutations pause only for the collection gate and the final view swap. A crash before manifest publication keeps the old view authoritative; a crash after publication selects the new view and later reclaims orphaned output.

6.1 Corruption domains

Authoritative record, primary, recovery, catalog, and manifest corruption fails the collection closed. Equality, posting, or range corruption can instead quarantine secondary queries while primary reads continue. An online rebuild derives new secondary structures from authoritative records and clears quarantine only after a new validated manifest is published.

7. Operations and Security

Production readiness requires more than a fast data path. Separate data and administration listeners, TLS or mTLS, bearer or certificate identities, role authorization, per-principal quotas, bounded streams, non-root container execution, checksums, audit events, and safe error messages are part of the architecture. Operational status must expose visible and durable sequence lag, disk-pressure state, secondary quarantine, pending mapping reclamation, maintenance progress, and bounded query activity without logging documents, tokens, private keys, or internal paths.

Failure domain	Detection	System response
Torn recovery tail	Frame length and checksum	Ignore incomplete final frame; recover valid prefix
Authoritative file corruption	CRC32C, SHA-256, structural bounds	Fail collection closed
Secondary-index corruption	Checksum or manifest digest	Quarantine secondary queries; preserve primary reads
Disk headroom exhausted	Admission thresholds and reserve	Read-only/backpressure state; keep safe reads
Long-lived readers	Pinned mapping references	Delay file deletion and retry reclamation

8. Qualification Evidence

The prototype was qualified through real-process tests, deterministic datasets, public gRPC drivers, crash-boundary injection, malformed fixtures, and cross-platform builds. The following retained results describe specific recorded hosts and are engineering evidence, not portable service-level guarantees.

Qualification	Scale	Recorded evidence
Primary load	1,000,000 documents	100,000 reads with 16 workers; warm primary p99 target under 2 ms
Equality queries	1,000,000 documents; 5 indexes	10,001 server queries; p99 2.341 ms; restart 1.000 s
Range queries	1,000,000 documents; 7 indexes	10,001 server queries; p99 6.488 ms; restart 2.547 s
Correlated batches	1,000 batches × 32 queries	712.85 queries/s; p99 batch 60.039 ms; zero item/query errors
Maintenance	10,000 records; 1,000 mutations	1,056,984 bytes reclaimed; 109 ms restart; zero storage/oracle errors

Source note: retained qualification reports and verification documents from the independent implementation. Results are host-specific and include the configured integrity and security controls.

9. Tradeoffs and Lessons

- Memory mapping removes heap copies; it does not remove lifecycle complexity. Mapping ownership, pinned readers, and deletion retries must be first-class concerns.
- Checksums identify damaged blocks, while manifests define authority. Both are required to avoid accepting mixed generations.
- Stale secondary entries are tolerable when every candidate is checked against authoritative current state and work remains bounded.
- Asynchronous cache durability must be explicit. Separate visible and durable sequences turn a vague promise into an observable contract.
- Compaction is a resource-governance problem as much as a storage algorithm. Disk reserves, I/O pacing, heap admission, and query fairness determine production behavior.
- Benchmark reports need reproducible datasets, exact configuration, errors, percentiles, memory evidence, and restart behavior; a single throughput number is insufficient.

10. Conclusion

A production-oriented cache engine earns trust by making its failure boundaries as deliberate as its fast path. Immutable mapped generations provide a compact read representation; the bounded recovery suffix makes mutation replay predictable; adaptive indexes align representation with data density; and manifest publication turns multi-file state into one validated decision. The result is not a general-purpose database. It is a refillable cache architecture whose resource limits, loss window, corruption behavior, and maintenance contract are explicit and testable.

References

1. RocksDB Overview and Write-Ahead Log Format. https://github.com/facebook/rocksdb/wiki/RocksDB-Overview

2. SQLite Write-Ahead Logging. https://www.sqlite.org/wal.html

3. SQLite Atomic Commit. https://www.sqlite.org/atomiccommit.html

4. MDB: A Memory-Mapped Database and Backend for OpenLDAP. https://www.openldap.org/pub/hyc/mdb-paper.pdf

5. Lucene Postings Format. https://lucene.apache.org/core/4_10_3/core/org/apache/lucene/codecs/lucene41/Lucene41PostingsFormat.html

6. Roaring Bitmap Format Specification. https://github.com/RoaringBitmap/RoaringFormatSpec

7. gRPC Flow Control. https://grpc.io/docs/guides/flow-control/

8. RocksDB Compaction. https://github.com/facebook/rocksdb/wiki/Compaction