The Relevance of Key-Value Stores (KVS) in Document Management, CRM, Chat, and Telecom Systems: An Erlang-Native Approach Aligned with Industrial SSD-Optimized Interfaces

Maxim
Synrc Research

May 22, 2026

Abstract

Key-Value Stores (KVS) provide a lightweight, high-performance abstraction for data management, particularly suited to write-intensive and stream-oriented workloads. This paper introduces the Synrc KVS library for the Erlang/OTP ecosystem and evaluates its applicability to document management systems (workflow), Customer Relationship Management (CRM), messaging/chat platforms, and telecommunications. We position KVS as a practical, embeddable solution that leverages backends such as RocksDB, drawing conceptual parallels to the SNIA Key Value Storage API 1.1 — the industry-standard interface pioneered in Samsung's KV-SSD prototypes for direct key-value access to SSDs.

1 Introduction to KVS

KVS (Key-Value Store) is a lightweight client-side interface built on B-tree-like database abstractions, designed for straightforward data storage and processing within the Erlang/OTP ecosystem. Developed by Synrc, KVS offers an abstract, compact, and battle-tested API (approximately 500 lines of code) that supports core operations for ledgers, messengers, storage systems, and banking applications.

KVS supports multiple backends, including Mnesia, RocksDB, filesystem (FS), Spandb, and others. This flexibility allows seamless adaptation from embedded systems to high-load servers. The primary modules are:

- **KVS** — for basic data operations (`put`, `get`, `delete`, `index`, `seq`, `count`, `dir`).
- **KVS Stream** — for working with data chains (feeds), cursors (writers/readers), and asynchronous processing.

Thanks to polymorphic (extensible) records and support for chain/tree structures, KVS is ideal for systems where data is organized in sequences or hierarchies. It ensures atomicity, crash resilience, and efficient navigation.

RocksDB, a primary backend for KVS, is a direct descendant of LevelDB and continues the tradition of classic embedded key-value stores such as Berkeley DB. This lineage makes RocksDB significantly more performant than traditional SQL databases for document-oriented workloads requiring high write throughput and low latency. RocksDB employs LSM-trees optimized for SSD writes, reducing write amplification.

This software-level KVS approach aligns conceptually with hardware-accelerated key-value interfaces such as the **SNIA Key Value Storage API 1.1**, developed to enable direct KV access to SSDs.

2 API Overview

2.1 KVS Module

- `put/1`: Stores a record using its ID as the key.
- `get/2`: Retrieves a record by key from a table.
- `delete/2`: Deletes a record.
- `index/3`: Searches records by field and value (RDBMS backends).
- `seq/2`: Generates a new atomic ID.
- `count/1`: Counts records in a table.
- `dir/0`: Returns the list of tables.

The backend is configured via application environment (e.g. `{dba, store_rocks}`).

2.2 KVS Stream Module

- `writer/1`, `reader/1`: Create cursors.
- `save/1`, `load_reader/1`: Persist/load cursors.
- `add/1`, `append/2`: Add elements to a chain.
- `next/1`, `prev/1`, `top/1`, `bot/1`: Navigation.
- `take/1`, `drop/1`: Extract/skip elements.
- `remove/2`, `cut/2`: Remove or truncate chain.

3 Suitability for Specific Systems

3.1 Document Management (Workflow Systems)

KVS efficiently manages document chains: each document is a record in a feed. The `add/1` operation appends new stages, while readers track status asynchronously. Polymorphic records support extensible metadata. Atomic `seq/2` ensures unique IDs.

3.2 CRM Systems

Customer interaction histories are modeled as event feeds. `append/2` adds new events, saved cursors provide personalized views, and field indexing enables fast queries.

3.3 Chat and Messaging Systems

KVS Stream is ideal for messaging. Message chains act as feeds with efficient pagination and cursor-based unread tracking. Cursor persistence ensures seamless recovery after restarts.

3.4 Queues with Cursors for Asynchronous Processing

KVS implements reliable queues using chains and persistent cursors, supporting high-throughput background job processing in Erlang.

3.5 Additional Use Cases

- Arbitrary Feeds (news, event logs)
- Immutable Ledgers and transaction chains
- Logging and time-series data
- Banking applications
- Gaming real-time event feeds
- IoT sensor streams
- Content Management Systems (versioning)
- Social network personalized feeds

4 Conclusion

Synrc KVS stands out for its simplicity, flexibility, and deep integration with Erlang/OTP. By providing high-level stream and cursor abstractions on top of efficient KV backends like RocksDB, it embodies principles similar to the SNIA KV API 1.1 — optimized data paths with minimal overhead.

For further details, visit the official documentation: https://kvs.n2o.dev

References

[1] SNIA Key Value Storage API Specification 1.1.

[2] Samsung KV-SSD Technology.

[3] RocksDB Documentation.