Kamo: Bringing Many-Core Scalability to MySQL

Yusuke Miyazaki*, Sho Nakazono†, Tatsuhiro Nakamori*,
Haowen Li*, and Hideyuki Kawashima*
*Keio University
†LY Corporation

***Abstract*—Kamo targets high-concurrency transaction processing in MySQL without replacing its SQL layer. Building on LineairDB, an embedded transactional key-value library with strict serializability, and its initial MySQL integration, Kamo adds relational tables, secondary indexes, and relational update semantics while extending the handler paths required by the evaluated workloads. BenchBase experiments show that Kamo's end-to-end throughput increased with client concurrency under the evaluated configurations: at 72 client threads, it reaches 5,163 requests/s in a single-node TPC-C experiment, 14.9 times the throughput of MySQL/InnoDB under the evaluated configuration.**

***Index Terms*—MySQL, storage engine, transaction processing, secondary index, many-core database**

I. Motivation and Integration Gap

Modern servers expose extensive thread-level parallelism, yet exploiting it within an established RDBMS remains difficult: in our TPC-C evaluation, MySQL/InnoDB throughput nearly plateaus as client concurrency rises. A standalone many-core transaction engine does not provide MySQL's SQL parser, optimizer, or client protocol. Kamo reuses these MySQL server components while implementing the storage-engine handler paths required by the evaluated workloads. MySQL's pluggable storage-engine boundary makes this separation possible [1], [2].

LineairDB offers multi-key transactions with strict serializability and a SiloNWR concurrency-control option aimed at contended many-core workloads; prior work also presented an initial MySQL storage-engine integration [3], [4]. Supporting the evaluated relational workloads, however, requires two extensions. The storage layer needs named tables, insert/update/delete semantics, secondary indexes, and ordered scans. The adapter layer needs broader handler coverage to translate MySQL table records and index tuples into LineairDB keys and values.

II. Design and Contributions

Kamo's first contribution is a relational extension to LineairDB. A concurrent table dictionary maps each table name to a distinct primary index; `CreateTable` and transaction-local `SetTable` preserve table identity through reads, writes, scans, logging, and checkpoints. The transaction API adds insert, update, delete, forward and reverse scans, and read-your-writes behavior. Secondary indexes map encoded secondary keys to primary keys and support uniqueness, key-changing updates, point lookup, and bidirectional range scans. Precision-locking metadata and snapshots distinguish tables and indexes, preventing range validation from conflating identical byte keys in different namespaces.

The second contribution is the MySQL adapter, whose implementation is publicly available [5]. Handler callbacks for table creation, row insertion, update, deletion, primary and secondary index access, full scans, and commit or rollback map to these APIs. The adapter serializes MySQL rows into LineairDB values and constructs order-preserving encodings for the data types exercised by the tests and evaluated workloads. Search flags become point, prefix, bounded-range, or predecessor plans; iterative handler calls consume materialized primary keys. Lazy transaction creation and commit/rollback translation align LineairDB transactions with MySQL statement and explicit-transaction boundaries. Tests cover multiple tables, composite and unique indexes, modification, rollback, scans, joins, and all five TPC-C transaction classes.

III. Evaluation

We evaluated Kamo and MySQL 8.0.43 with InnoDB using BenchBase [6] on the same KVM server: an AMD EPYC 9654P (96 cores, 192 vCPUs), 1 TiB memory, and Ubuntu 22.04.5. Both engines were evaluated at the `SERIALIZABLE` isolation level with no request-rate limit. TPC-C uses scale factor 1, the 45/43/4/4/4 transaction mix, a 60-second measurement, and 1–72 client threads. YCSB-A uses scale factor 1 (1,000 initial records), a 50/50 read/update mix, a 15-second measurement, and 1–144 threads.

Kamo's advantage emerges as concurrency increases (Fig. 1). On TPC-C, it delivers 1,582 requests/s at 12 threads versus 298 for InnoDB (5.3×), then reaches 5,163 versus 346 requests/s at 72 threads (14.9×). On YCSB-A, Kamo reaches 43,189 requests/s at 144 threads versus 2,568 for InnoDB (16.8×). A separate end-to-end experiment with ProxySQL, one primary, two semi-synchronous replicas, and all queries routed to the primary yields 139.7 versus 7.45 requests/s at 144 threads on TPC-C (18.7×). The integrated Kamo prototype exhibited concurrency scaling across a complex OLTP workload and a simpler read/update workload, although this evaluation does not isolate the overhead of the relational and handler layers.

IV. Limitations and Conclusion

The evaluation targets scale-factor-1 workloads; larger and storage-resident datasets remain future work. Kamo demonstrates a two-layer approach to relationalizing a transactional

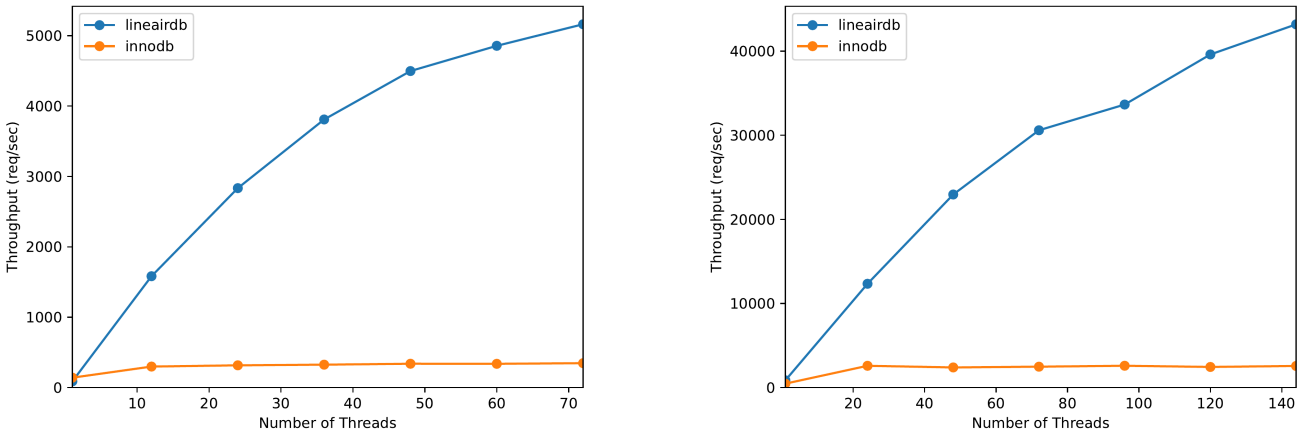


Fig. 1. Single-node throughput for Kamo (`lineairdb` in the source plots) and MySQL/InnoDB: TPC-C (left) and YCSB-A (right). Conditions are stated in the text.

key-value library and connecting it to an established RDBMS without modifying MySQL's SQL layer.

Acknowledgment

This paper is based on results obtained from the project "Research and Development Project of the Enhanced Infrastructures for Post-5G Information and Communication Systems (JPNP20017)" and JPNP25014 by the New Energy and Industrial Technology Development Organization (NEDO), and from JSPS KAKENHI Grant Number 25H00446, and from JST CREST Grant Number JPMJCR24R4 and from SECOM Science and Technology Foundation and from JST COI-NEXT SQAI (JPMJPF2221).

References

[1] Oracle Corporation, "Overview of MySQL storage engine architecture," MySQL 8.0 Reference Manual, https://dev.mysql.com/doc/refman/8.0/en/pluggable-storage-overview.html, 2026, accessed: 2026-08-07.

[2] ——, "MySQL 8.0.43: handler class reference," MySQL Server Developer Documentation, https://dev.mysql.com/doc/dev/mysql-server/8.0.43/classhandler.html, 2025, interface documentation for dynamically loadable storage engines.

[3] S. Nakazono, H. Uchiyama, Y. Fujiwara, Y. Nakamura, and H. Kawashima, "NWR: Rethinking Thomas Write Rule for omittable write operations," 2020. [Online]. Available: https://arxiv.org/abs/1904.08119

[4] S. Nakazono, Y. Bessho, and T. Nakamori, "LineairDB: Transactional key-value storage with epoch-based synchronization," *Computer Software*, vol. 41, no. 1, pp. 1_15–1_35, 2024.

[5] mitou-Kamo contributors, "LineairDB-storage-engine (Kamo)," GitHub repository, https://github.com/mitou-Kamo/LineairDB-storage-engine, 2026, commit 4e11f9d, accessed: 2026-08-07.

[6] BenchBase contributors, "BenchBase: A multi-DBMS SQL benchmarking framework," GitHub repository, https://github.com/cmu-db/benchbase, 2022, Kamo submodule commit 979b53b, accessed: 2026-08-07.