Kamo: Bringing Many-Core Scalability to MySQL

Yusuke Miyazaki [1] · **Sho Nakazono** [2] · **Tatsuhiro Nakamori** [1] · **Haowen Li** [1] · **Hideyuki Kawashima** [1]

[1] Keio University [2] LY Corporation

Source code
github.com/mitou-Kamo/LineairDB-storage-engine

Key Contributions

what was missing → what we built → what the evaluation shows (and does not show)

C1 **Relational extensions inside LineairDB —** named tables, insert/update/delete semantics, secondary indexes, and ordered scans (7 PRs merged upstream).

C2 **Substantially extended MySQL handler —** building on the initial LineairDB plugin [4], it adds the search-planning, cursor, composite-key, statistics, and transaction paths needed to run all five BenchBase TPC-C transaction classes.

C3 **End-to-end evaluation —** unmodified BenchBase over MySQL's SQL/JDBC interface vs MySQL 8.0.43/InnoDB, both SERIALIZABLE, scale factor 1: Kamo's requests/s keep rising with client threads while InnoDB plateaus — 14.9× (TPC-C) / 16.8× (YCSB-A) on one node, 18.7× (TPC-C) with 3-node semi-sync. Client-thread sweeps, not core-count scaling.

1 Problem & Motivation

- **MySQL/InnoDB** predates the many-core era: a 2009 multicore study found its threads blocked at InnoDB's thread-concurrency gate for **≈39% of execution time** [3], and in our runs its throughput plateaus at low client concurrency (Sec. 7).
- Many-core-scalable engines exist (Silo [1], LineairDB [4]) — but as embedded key-value libraries with no SQL and no standard client protocol, so existing applications cannot use them.
- **Goal:** many-core scalability **behind an unchanged MySQL interface** [10].
- **Approach:** extend LineairDB with relational operations (tables, secondary indexes, insert/update/delete, ordered scans) and extend the MySQL storage-engine handler; MySQL's server and clients stay unmodified (Sec. 3–5).

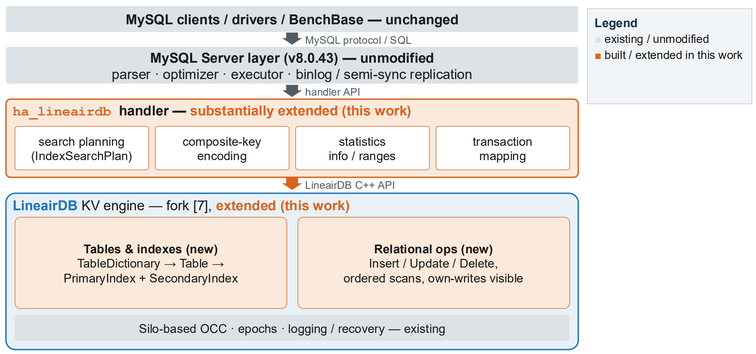
2 Technical Gap

- **LineairDB** [4]: embedded transactional key-value library (Silo-style optimistic CC + epochs [1]); strictly serializable, many-core-scalable — **not an SQL database.**
- **An initial MySQL storage-engine plugin for LineairDB existed [4, 9]**, covering single-key reads, upserts, and table scans — not the relational workloads targeted here.

Baseline at project start [9]	Needed for BenchBase TPC-C
· KV `Read/Write/Scan`, one flat key space · Primary index only · Plugin: single-key read, upsert, full scan · No reverse / cursor scans	· Tables + DDL; secondary / unique / composite indexes · Insert / update / delete semantics · Index cursors, both scan directions · Optimizer stats, txn mapping

- **Precedent:** MyRocks needed a substantial relational layer to put RocksDB behind MySQL [2] — the same gap Kamo must close for LineairDB.

3 System Architecture



The MySQL server layer is left unchanged: the MySQL client protocol, the BenchBase/JDBC path, and row-based binlog replication remain usable.

4 LineairDB Extensions

- **Named tables:** `Table` + `TableDictionary` give each table its own primary index, so MySQL DDL addresses tables by name; identity survives logging and checkpoints.
- **Relational DML:** `Insert` / `Update` / `Delete` abort on duplicate or missing keys (unlike upsert-only `Write`); scans run in ASC/DESC order and see the transaction's own writes.
- **Secondary indexes (new):** one secondary key maps to many primary keys; point and range layers are designed to keep lookups and range scans consistent — needed for TPC-C's non-primary-key access paths. Index changes are logged as (PK, Add/Remove) deltas.
- **Correctness:** removed self-conflict false aborts in precision locking; test-first, 7 PRs merged upstream.

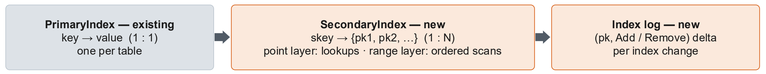
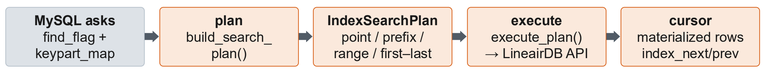


Fig. 1 Data model added to LineairDB: a per-table primary index (1 : 1) plus secondary indexes that map one secondary key to a set of primary keys (1 : N); index updates are logged as deltas.

5 MySQL Integration — extending ha_lineairdb



Read path (new): every MySQL search mode becomes an explicit plan; secondary lookups fetch primary keys, then re-read base rows.

- **Write path:** row writes extract the (composite) primary key from MySQL row buffers and update the base row plus every affected secondary index in one transaction.
- **Composite keys:** per-keypart encoding (NULL · type · payload · length) yields one binary-comparable key and enables prefix search, for the types and index shapes in the tests and workloads.
- **Optimizer statistics:** heuristic row counts via `info()/records_in_range()` — without them MySQL chose full scans and TPC-C throughput collapsed.
- **Transactions:** lazy start on first handler call; statement vs explicit commit; engine aborts propagate to MySQL rollback; SERIALIZABLE only.
- **Outcome:** MySQL's SQL/server layer stays unchanged; all five BenchBase TPC-C transaction classes run through the storage-engine plugin [6].

6 Experimental Setup

DB nodes	3 × KVM VMs (1 primary + 2 replicas), each AMD EPYC 9654P, 192 vCPUs, 1 TiB RAM, Ubuntu 22.04, Linux 5.15; single-node runs use one VM with BenchBase on it
Proxy/client	ProxySQL + BenchBase node (3-node runs): Xeon Gold 6212U, 20 vCPUs, 31 GiB
Baseline	MySQL 8.0.43 with InnoDB ("Existing MySQL" in plots); Kamo = extended LineairDB [7] + handler [6]; BenchBase [5, 8]
Isolation	SERIALIZABLE for both engines (Kamo supports SERIALIZABLE only); no request-rate limit
TPC-C	scale factor 1 = 1 warehouse; mix 45/43/4/4/4 (New-Order / Payment / Order-Status / Delivery / Stock-Level); 60 s per run†
YCSB-A	scale factor 1 = 1,000 records; 50 % read / 50 % update; 15 s per run†
Client threads	BenchBase terminals†: 3-node 1, 16, …, 144; single node TPC-C 1, 12, …, 72 and YCSB-A 1, 24, …, 144
Replication	MySQL semi-synchronous, row-based binlog; a commit waits for the ACK of ≥ 1 replica†, no async fallback† (Fig. 2)

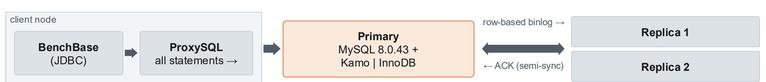
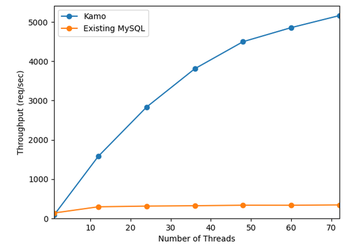


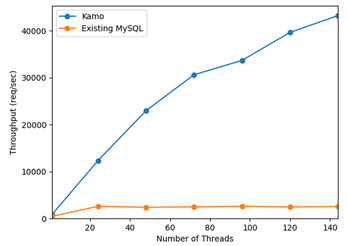
Fig. 2 3-node configuration: ProxySQL routes every statement to the primary (replica reads disabled); schema creation and loading ran directly on the primary, the execute phase through ProxySQL. An end-to-end replicated setup, not a scale-out experiment.

7 Results

Single node — BenchBase and MySQL on the same VM (x: client threads, y: requests/s)

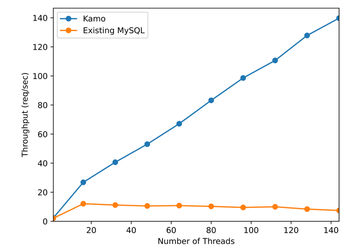


(a) TPC-C, single node, 1 warehouse — Kamo 5,163 vs InnoDB 346 req/s at 72 threads (14.9×). InnoDB is nearly flat from 12 threads; at 1 thread InnoDB is faster (140 vs 88).

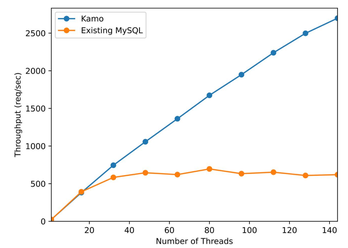


(b) YCSB-A, single node, 1,000 records — Kamo 43,189 vs InnoDB 2,568 req/s at 144 threads (16.8×). InnoDB stays near 2.5 k req/s from 24 threads on; 1 thread: 859 vs 464.

3-node semi-sync + ProxySQL — every statement executes on the primary (Fig. 2)



(c) TPC-C, 3-node via ProxySQL, 1 warehouse — Kamo 139.7 vs InnoDB 7.45 req/s at 144 threads (18.7×); InnoDB peaks at 12.1 (16 threads). Both far below single node (1 thread: 2.4 vs 2.1).



(d) YCSB-A, 3-node via ProxySQL, 1,000 records — Kamo 2,698 vs InnoDB 620 req/s at 144 threads (4.35×); comparable up to 16 threads (382 vs 392) — a narrower gap than in (b).

Requests/s as exported from BenchBase (not tpmC); one archived value per point. With one warehouse, TPC-C concentrates contention on a few district rows, so these runs do not separate the effect of data contention from that of the concurrency-control method; they show end-to-end behaviour under the stated configurations only.

8 Limitations

- SERIALIZABLE only; SQL coverage driven by the evaluated workloads; optimizer statistics are heuristic; ProxySQL is a single point of failure in the 3-node setup.
- Scale factor 1 (1 warehouse / 1,000 records), requests/s over BenchBase — not an official TPC-C result (no tpmC); the overhead of the relational and handler layers is not isolated.
- Concurrency was varied by client threads, not by the cores given to the server; per-core scaling, NUMA effects, and larger warehouse counts were not measured.
- Server/InnoDB parameters beyond those listed, warm-up, and repetitions are not in the archived exports; the low InnoDB throughput in the 3-node runs is reported as measured, without a diagnosed cause.

9 Conclusion & Future Work

- **Kamo makes an existing many-core transaction engine usable through MySQL's standard SQL/JDBC interface** by adding relational operations to LineairDB and extending the storage-engine handler; MySQL's server layer is untouched.
- Under the evaluated configurations its throughput kept rising with client threads where InnoDB plateaued (14.9× / 16.8× single node, 18.7× 3-node TPC-C). Diagnostic runs point to server-layer logging (single node) and relay-log synchronization (3-node) as the next bottlenecks — not isolated by the comparisons shown.
- **Next:** measure with more warehouses and controlled core counts, remove the ProxySQL single point of failure, broaden SQL coverage, and prepare a documented, reproducible open-source release.

References & Acknowledgments

[1] S. Tu et al. Speedy Transactions in Multicore In-Memory Databases. SOSP 2013.
[2] Y. Matsunobu et al. MyRocks: LSM-Tree Database Storage Engine Serving Facebook's Social Graph. PVLDB 13(12), 2020.
[3] R. Johnson et al. Shore-MT: A Scalable Storage Manager for the Multicore Era. EDBT 2009.
[4] S. Nakazono, Y. Bessho, T. Nakamori. LineairDB: Transactional Key-Value Storage with Epoch-based Synchronization. Computer Software 41(1):15–35, 2024.
[5] D. E. Difallah, A. Pavlo, C. Curino, P. Cudré-Mauroux. OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases. PVLDB 7(4), 2013.
[6] Kamo (evaluated): github.com/mitou-Kamo/LineairDB-storage-engine, commit `4e11f9dddb5c209e7e868de7b1e72e6eadfaebe3`
[7] Extended LineairDB (evaluated): github.com/wasanemon/LineairDB-ssh, branch secondary_index_test, commit `60566a1073fb61c9e72dd5108876ffcf953e094c`
[8] BenchBase (evaluated): github.com/cmu-db/benchbase, commit `979b53b043f934220f703b149f27a7ee0f992b63`
[9] Project-start baselines: LineairDB commit `5df6372e728c` (2025-02); initial MySQL plugin (github.com/Tatzhiro/LineairDB-storage-engine) commit `5df270e874ed` (2023-04).
[10] Oracle. MySQL 8.0 Reference Manual: Storage Engine Architecture. [11] Anthropic. Claude. Large language model, accessed Aug.–Sep. 2026.

Acknowledgments: 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).

AI disclosure: Claude (Anthropic) [11] was used for language editing and for revising the wording and layout of this poster; the authors verified and take full responsibility for the final text.