Helios: A Disaggregated MySQL Database That Scales Without Replication

Masahide Fukuyama[1], Yusuke Miyazaki[1], Sho Nakazono[2], Hideyuki Kawashima[1]

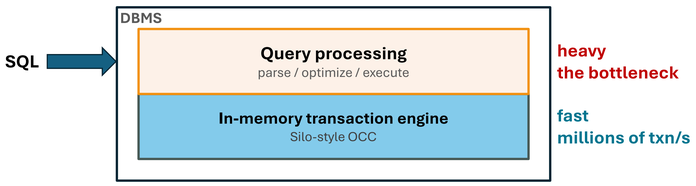
[1] Keio University [2] LY Corporation

Available on GitHub

1. Background

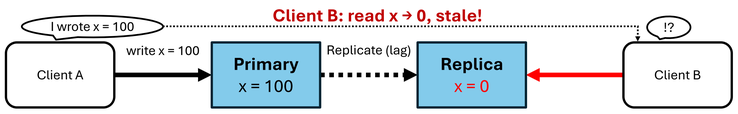
Query Processing limits End-to-End Throughput

- Modern in-memory transaction engines process **millions of txn/s** on a single many-core server [1].
- A DBMS must also parse and optimize SQL and execute the resulting query plans.
  - As engines get faster, **query processing becomes the bottleneck.**



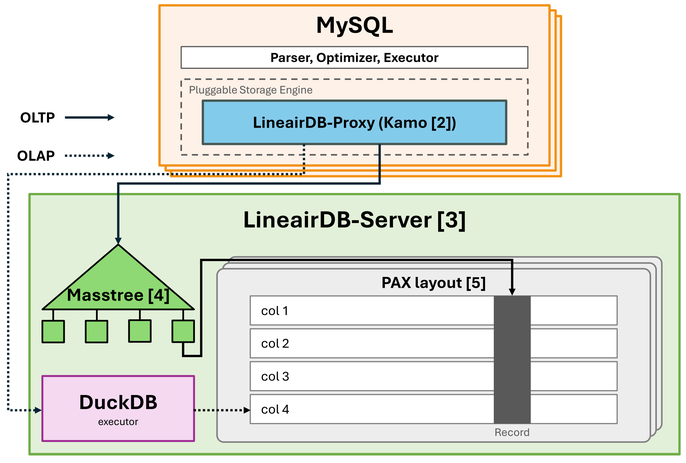
Scale-Out Pays for Capacity You May Not Need

- **Read-serving replicas** add read capacity and availability, but require update propagation and **can serve stale data**.
- **Shards** add storage and transaction capacity, but need **cross-node coordination** for multi-shard transactions.



2. Approach

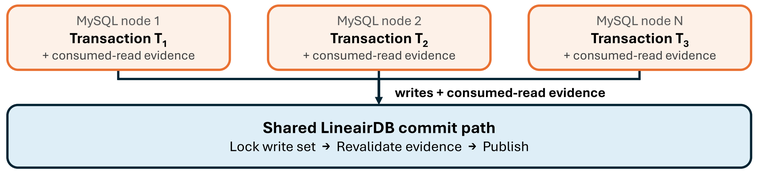
If the shared transaction engine is fast enough,
scale the query layer. Not the data.
Multiple read/write-capable MySQL nodes share ***one*** in-memory transaction engine.



3. Design

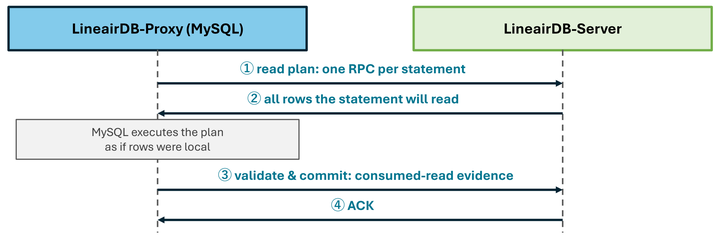
Shared Transaction Engine

- Each query node retains **validation evidence** for the point and range reads its OLTP transactions consume.
- At commit, the storage server **revalidates** this evidence using Silo's commit protocol [6] and orders commits.
- The serial order is consistent with real time, **external consistency across query nodes.**



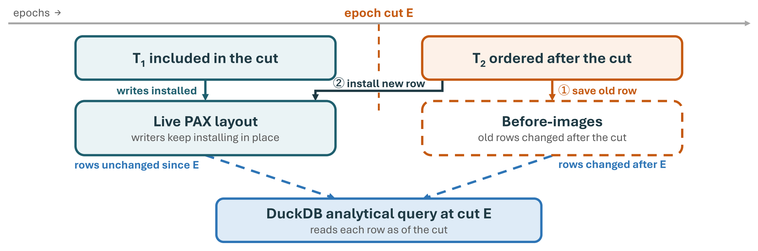
Plan-Derived Prefetch

- MySQL's row-at-a-time handler calls would create **many synchronous RPCs**.
- Helios compiles the storage accesses in **MySQL's chosen QEP** into a **dependent read plan**.
- The storage server **resolves dependent accesses in one RPC** by deriving later lookup keys from earlier results.



Consistent Analytical Reads

- Analytical queries run on the **same PAX layout [5] updated by OLTP transactions**.
  - **No analytical replica or ETL** is required.
- Before each query, Helios activates **before-image capture**, takes an **epoch cut**, and waits for commits included in the cut commits to install their writes.
- Transactions ordered after the cut save **each row's before-image** before overwriting it.
- DuckDB [7] reads each row **as of the cut** from the live PAX layout or its before-image.
  - **One commit epoch covers all writes of a transaction**, so the query **sees all or none**.



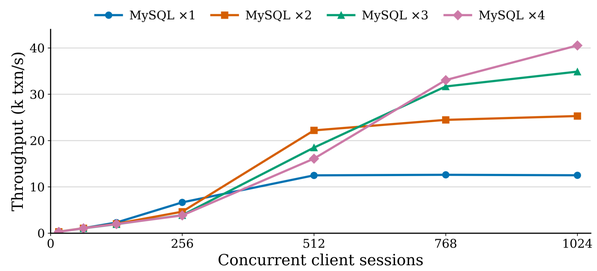
4. Evaluation

Setup (Amazon EC2)

Workload	Layer	Component & Count	Instance
TPC-C	Client / Routing	2 × BenchBase [8] + 1 × HAProxy	c6i.16xlarge
	Query	1–4 × MySQL	c6i.16xlarge
	Storage	1 × LineairDB (512 GiB)	r6i.16xlarge
TPC-H	Helios · TiDB (TiFlash, MPP) · MySQL (InnoDB), 1 each		c6i.16xlarge

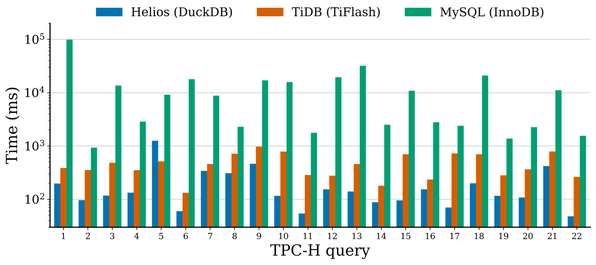
64 vCPUs each, Ubuntu 24.04.
TPC-C: 1,024 warehouses, five transaction types, serializable isolation, 16–1024 JDBC sessions, 60 s runs. TPC-H: SF=10

TPC-C: Query-node Scaling



3.24 × At 1,024 sessions, MySQL ×4 reaches 40.5k txn/s, a 3.24 × speedup. MySQL ×1 plateaus at 12.5k txn/s. The four-node curve is still rising.

TPC-H: Query Performance



2.20 × Helios totals 4.73 s, 2.20 × faster than TiDB/TiFlash at 10.4 s. MySQL/InnoDB takes 295 s, 62.4 × as long as Helios.

5. Conclusions

- Helios scales MySQL query processing **without read-serving replicas or storage shards** and preserves **external consistency**.
- Plan-derived read plans cut network round trips; epoch cuts + before-images give **consistent analytical reads** on one shared PAX layout.
- **Future work**: mixed OLTP/OLAP workloads; replicating the storage server for fault tolerance.

References

[1] T. Tanabe et al., "An Analysis of Concurrency Control Protocols for In-Memory Databases with CCBench," *PVLDB* 13(13), 2020.
[2] mitou-Kamo, "LineairDB MySQL Custom Storage Engine," GitHub repository, 2026.
[3] S. Nakazono, Y. Bessho, and T. Nakamori, "LineairDB: Transactional Key-Value Storage with Epoch-Based Synchronization," *Computer Software* 41(1), 2024.
[4] Y. Mao, E. Kohler, and R. T. Morris, "Cache Craftiness for Fast Multicore Key-Value Storage," *EuroSys*, 2012.
[5] A. Ailamaki, D. J. DeWitt, M. D. Hill, and M. Skounakis, "Weaving Relations for Cache Performance," *VLDB*, 2001.
[6] S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden, "Speedy Transactions in Multicore In-Memory Databases," *SOSP*, 2013.
[7] M. Raasveldt and H. Mühleisen, "DuckDB: An Embeddable Analytical Database," *SIGMOD*, 2019.
[8] D. E. Difallah, A. Pavlo, C. Curino, and P. Cudré-Mauroux, "OLTP-Bench: An Extensible Testbed for Benchmarking Relational Databases," *PVLDB* 7(4), 2013.

Acknowledgments

This work is based on results obtained from the NEDO projects JPNP20017 and JPNP25014, JSPS KAKENHI Grant 25H00446, JST CREST Grant JPMJCR24R4, the SECOM Science and Technology Foundation, and JST COI-NEXT SQAI (JPMJPF2221).
OpenAI Codex and Anthropic Claude assisted in drafting parts of the text; the authors reviewed, verified, and edited all content.