PaxFlow: A Fast-Path Optimization for Read-Only Workloads in Deterministic Database Systems

Lisa Ozaki[1,a)] Haowen Li[2,b)] Hideyuki Kawashima[1,c)]

Abstract:
Deterministic databases such as Calvin achieve high throughput by pre-ordering transactions, but still impose unnecessary locking overhead on read-only (RO) workloads. This causes head-of-line blocking and poor scalability. We present PaxFlow, a fast-path optimization that bypasses the lock manager for RO transactions. PaxFlow improves throughput by up to 4.3× on read-only YCSB workloads, from 3.20K to 13.7K tps.

1. Motivation

Transaction processing is fundamental to modern distributed database systems, enabling large-scale applications such as financial services, e-commerce, and social media platforms. To ensure correctness under concurrent access, these systems must preserve ACID properties (atomicity, consistency, isolation, and durability).

Traditional concurrency-control protocols, such as Two-Phase Locking (2PL) [1] and Optimistic Concurrency Control (OCC) [2, 3], are non-deterministic and resolve conflicts at run time. In 2PL, avoiding deadlock requires aborting one or more transactions once a wait-for cycle is detected, while in OCC conflicting transactions are aborted during the validation phase. As contention increases, these aborts and rollbacks become more frequent, leading to degraded performance.

To address this problem, deterministic transaction processing has emerged as an attractive alternative [4]. Systems such as Calvin [5] predefine a global transaction order before execution, thereby eliminating deadlocks and achieving stable, high throughput even under high contention.

2. Problem

Deterministic transaction databases such as Calvin [5] execute transactions in a global serial order predetermined by a sequencer. To ensure correctness, this order is enforced through a lock-based protocol that coordinates concurrent execution.

However, a critical inefficiency arises because this uniform, lock-based enforcement is applied even to read-only (RO) transactions. Specifically, an RO transaction must still wait for locks held by preceding read-write transactions, even though it does not modify any data. This unnecessary locking introduces avoidable stalls and limits system scalability.

In deterministic systems, all transactions within an epoch are logically ordered but can be viewed as executing in parallel, with their commit order determined only by epoch completion. Thus, RO transactions can safely read from a previous epoch without violating serializability. By leveraging Multi-Version Concurrency Control (MVCC) [6], RO transactions can be executed fully without locks.

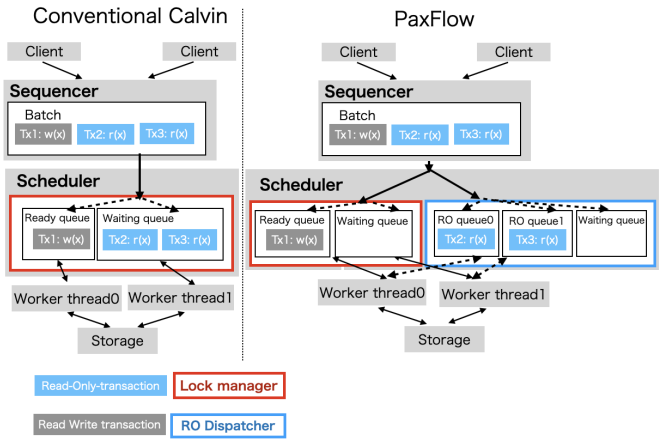

Fig. 1: Comparison of execution paths in conventional Calvin and PaxFlow.

3. Proposal

To address the scalability bottlenecks of deterministic databases, we propose PaxFlow, a fast-path optimization that enables read-only (RO) transactions to bypass the deterministic lock manager entirely.

As illustrated in Fig. 1, PaxFlow introduces four coordinated architectural components:

[1] Faculty of Environment and Information Studies, Keio University, Japan
[2] Graduate School of Media and Governance, Keio University, Japan
[a)] lisaozaki0402@keio.jp
[b)] tony_li.haowen@keio.jp
[c)] river@sfc.keio.ac.jp

1) Sequencer classification and batching

When a transaction request arrives from a client, the sequencer classifies it as either read-only (RO) or read-write (RW) according to its accessed data items. It then marks each batch with a flag indicating the presence of any RW transactions. Finally, the sequencer assigns a unique transaction ID (txid) and epoch number to every transaction and aggregates them into deterministic batches.

2) Scheduler Dispatch and Lock Handling

The scheduler classifies transactions into two execution paths: **Read-Write (RW)** and **Read-Only (RO)**. RW transactions are routed through the deterministic lock manager to preserve serializability via ordered lock acquisition. RO transactions, in contrast, are evaluated on the latest committed snapshot identified by the current `publish_epoch` and their designated `ro_read_version`.

Immediate Dispatch: When the required snapshot is available (i.e., `publish_epoch` ≥ `ro_read_version`), RO transactions are dispatched immediately and executed concurrently without locks.

Waiting Queue: If the required version has not yet been published, the RO transaction is temporarily placed in a waiting queue. The `ro_read_version` is advanced when the next batch arrives, provided that the previous batch contained RW updates, and then the queued transactions are released. This mechanism maintains strict snapshot consistency while enabling high concurrency for read-only workloads.

3) Worker execution

Worker threads give priority to **Read-Write (RW)** transactions to ensure that updates are applied in a deterministic order. **Read-Only (RO)** transactions, on the other hand, execute concurrently on the most recent committed snapshot, reading only data that was committed in the previous epoch.

4) Storage architecture

The storage subsystem maintains two layers: a write buffer that stores uncommitted modifications made by RW transactions and a committed snapshot that serves RO reads.

When all RW transactions in the current epoch finish, the system atomically merges the buffered updates into the committed layer, producing a new consistent snapshot which is the latest committed state.

Consequently, PaxFlow removes lock acquisition and release overhead.

4. Evaluation

Experiments were conducted on a dual-socket NUMA server with two Intel Xeon Gold 6240L CPUs (72 logical cores, 2.60 GHz) and 1.5 TB of memory, running Ubuntu 22.04.4 LTS. The source code is available at [7]. We implement and evaluate PaxFlow in the open-source Calvin repository using high-contention, read-only YCSB workloads [8], specifically varying the number of sequencers and dedicated RO dispatchers to empirically demonstrate its scalability.

As shown in Fig. 2, PaxFlow achieved up to **13.7K tps** (**4.3×** faster than Calvin's 3.20K tps) on read-only YCSB workloads [8]. Experimental parameters are summarized in Table 1.

Table 1: Experimental Settings

Parameter	Value
Transactions per batch	2,000
RO transactions	100% → 2,000
Operations per transaction	100
Hot records	10
Skew	0.99
Total records in database	1,000,000

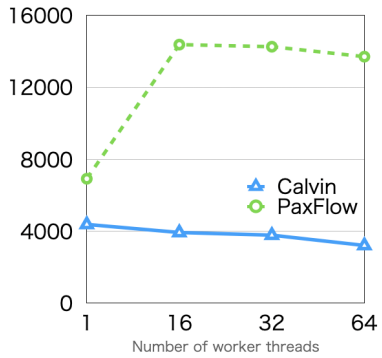

Fig. 2: Throughput comparison between Calvin and PaxFlow.

5. Conclusion

We proposed **PaxFlow**, a fast-path optimization for deterministic databases that eliminates RO locking bottlenecks to achieve up to 4.3× higher throughput on read-only YCSB workloads while preserving serializability.

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 JPNP16007 commissioned 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 JST COI-NEXT SQAI (JPMJPF2221), JST Moonshot R&D Grant Number JPMJMS2215.

References

[1] K. P. Eswaran, J. N. Gray, R. A. Lorie, and I. L. Traiger, "The notions of consistency and predicate locks in a database system," *Commun. ACM*, 1976.
[2] H. T. Kung and J. T. Robinson, "On optimistic methods for concurrency control," *ACM Trans. Database Syst.*, 1981.
[3] S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden, "Speedy transactions in multicore in-memory databases," in *SOSP*, 2013.
[4] A. Thomson and D. J. Abadi, "An overview of deterministic database systems," *IEEE Data Eng. Bull.*, 2018.
[5] A. Thomson, T. Diamond, S.-C. Weng, K. Ren, P. Shao, and D. J. Abadi, "Calvin: Fast distributed transactions for partitioned database systems," in *SIGMOD*, 2012.
[6] P. A. Bernstein and N. Goodman, "Concurrency control in distributed database systems," *ACM Comput. Surv.*, 1981.
[7] L. Ozaki and H. Li, "Paxflow source code," https://github.com/lisa0401/calvin/tree/ycsb-support.
[8] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with ycsb," in *SoCC*, 2010.