Lecture #06: Multi-Version Concurrency Control (MVCC) – Part 2

15-721 Advanced Database Systems (Spring 2018)

http://15721.courses.cs.cmu.edu/spring2018/

Carnegie Mellon University
Prof. Andy Pavlo

Microsoft Hekaton

Incubator project started in 2008 to create new in-memory OLTP engine for **Microsoft SQL Server**. Led by top database experts Paul Larson and Mike Zwilling. It was important that the new engine integrated with **MSSQL** ecosystem. It was also necessary that the engine support all possible OLTP workloads with predictable performance.

Key lessons:

- Use only lock-free data structures. This means no latches, spin locks, or critical sections for indexes, transaction map, memory allocator, garbage collector.
- Only one single serialization point in the DBMS to get the transaction's begin and commit timestamp using an atomic addition (CAS).

The Hekaton concurrency control relies on compare-and-swap to install new versions [2].

- Each transaction is assigned a timestamp when they begin (`BeginTS`) and when they commit (`EndTS`).
- Each tuple contains two timestamps that represents their visibility and current state.
  - **`BEGIN`**: The `BeginTS` of the active transaction or the `EndTS` of the committed transaction that created it.
  - **`END`**: The `BeginTS` of the active transaction that created the next version or infinity or the `EndTS` of the committed transaction that created it.
- Speculative Reads: Hekaton allows transactions to read versions of transactions of uncommitted transactions. Then checks in validations if the transactions that it read uncommitted data committed. The DBMS does not allow speculative write; first transaction to write a new version succeeds, second gets aborted.

Transaction State Map

Global map of all transactions states in the system. Transactions have to consult this map to determine the state of a transaction.

- `ACTIVE`: The transaction is executing read/write operations.
- `VALIDATING`: The transaction has invoked commit and the DBMS is checking whether it is valid.
- `COMMITTED`: The transaction is finished, but may not have updated its version's `TS`.
- `TERMINATED`: The transaction has updated the `TS` for all of the versions that it created.

Transaction Meta-Data

- `Read Set:` Pointers to every version that the transaction read.
- `Write Set:` Pointers to the versions that the transaction updated (old and new), deleted (old), and inserted (new).
- `Scan Set:` Stores enough information needed to perform each scan operation.
- `Commit Dependencies:` List of transactions that are waiting for this transaction to finish.

Transaction Validation

- `Read Stability:` Check that each version read is still visible as of the end of the transaction.
- `Phantom Avoidance:` Repeat each scan to check whether new versions have become visible since the transaction began.
- Extend of validation depends on isolation level:
  - `SERIALIZABLE`: Read Stability + Phantom Avoidance.
  - `REPEATABLE READS`: Read Stability.
  - `SNAPSHOT ISOLATION`: None
  - `READ COMMITTED`: None

HyPer

Read/scan set validations are expensive if transactions access a lot of data. Appending new versions hurts the performance of OLAP scans due to pointer chasing and branching. Record level conflict checks may be too coarse-grained and incur false positives.

HyPer uses a MVCC implementation that is designed for HTAP workloads [4]. It is a column-store with delta record versioning. Avoids write-write conflicts by aborting transactions that try to update an uncommitted object.

- In-Place updates for non-indexed attributes
- Delete/Insert updates for indexed attributes
- N2O version chains
- No Predicate Locks and No Scan Checks

Transaction Validation

- **First-Writer Wins:** The version vector always points to the last committed version. Do not need to check whether write-sets overlap.
- Check the undo buffers (i.e., delta records) of transactions that committed after the validation transaction started.
- Compare the committed transaction's write set for phantoms using **Precision Locks** [1].
- Only need to store the transaction's read predicates and not its entire read set.

Version Synopsis

- Store a separate column that tracks the position of the first and last versioned tuple in a block of tuples.
- When scanning tuples, the DBMS can check for strides of tuples without older versions and execute more efficiently.

CMU Cicada

In-memory OLTP engine based on optimistic MVCC with append-only storage (N2O) [3]. Designed to be scalable for both low and high contention workloads.

- Best-effort in-lining
- Loosely Synchronized Clocks
- Contention Aware validation
- Index Nodes Stored in Tables

Best-Effort In-lining

- Record meta-data is stored in a fixed location.
- Threads will attempt to inline read-mostly version within this meta-data to reduce version chain traversals.

Fast Validation

- **Contention-aware Validation:** Validate access to recently modified (high contention) records first.
- **Early Consistency Check:** Pre-validated access set before making global writes.
- If the DBMS knows that most of the recently executed transactions committed successfully, then it can skip Contention-Aware Validation and Early Consistency check
- **Incremental Version Search:** Resume from last search location in version list.

References

[1] J. R. Jordan, J. Banerjee, and R. B. Batman. Precision locks. In *Proceedings of the 1981 ACM SIGMOD International Conference on Management of Data*, SIGMOD '81, pages 143–147, 1981. doi: 10.1145/582318.582340. URL http://doi.acm.org/10.1145/582318.582340.

[2] P.-A. Larson, S. Blanas, C. Diaconu, C. Freedman, J. M. Patel, and M. Zwilling. High-performance concurrency control mechanisms for main-memory databases. In *VLDB '11: Proceedings of the VLDB Endowment*, volume 5, pages 298–309, December 2011. URL https://dl.acm.org/citation.cfm?id=2095686.2095689.

[3] H. Lim, M. Kaminsky, and D. G. Andersen. Cicada: Dependably fast multi-core in-memory transactions. In *SIGMOD '17: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 21–35, 2017. doi: https://dl.acm.org/citation.cfm?id=2749436.

[4] T. Neumann, T. Mhlbauer, and A. Kemper. Fast serializable multi-version concurrency control for main-memory database systems. In *SIGMOD '15: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data*, pages 677–689, 2015. doi: https://dl.acm.org/citation.cfm?id=2749436.