Lecture #04: Optimistic Concurrency Control

15-721 Advanced Database Systems (Spring 2018)
http://15721.courses.cs.cmu.edu/spring2018/
Carnegie Mellon University
Prof. Andy Pavlo

1 Stored Procedures

Disk stalls are (almost) gone when executing transactions in an in-memory DBMS. But there are still other stalls when an application uses **conversational** API to execute queries on DBMS (e.g., JDBC/ODBC, wire protocols). Solutions to this problem:

1. Prepared Statements
2. Query Batches
3. Stored Procedures

Prepared Statements: Provide the DBMS the SQL statement ahead of time and assign it to a name/handle. Can invoke that query just by using that name. This removes SQL parsing, binding, and planning (sometimes).

Query Batches: Invoke multiple queries per network message. This reduces the number of network round-trips

Stored Procedures: A group of queries that form a logical unit and perform a particular task on behalf of an application directly inside of the DBMS. The application can then invoke the transaction as if it was an RPC. This removes both preparation and network stalls.

Advantages of Stored Procedures:

- Reduce the number of round trips between application and database servers.
- Increased performance because queries are pre-compiled and stored in DBMS.
- Procedure reuse across applications.
- Transparent sever-side transaction restarts on conflicts.

Disadvantages of Stored Procedures:

- Not as many developers know how to write stored procedure code.
- Outside the scope of the application so it is difficult to manage versions and hard to debug.
- Probably not to be portable to other DBMSs.
- DBAs usually do not give permissions out freely, so it makes it difficult for developers to constantly update their stored procedures.

2 Concurrency Control

A DBMS' concurrency control protocol to allow transactions to access a database in a multi-programmed fashion while preserving the illusion that each of them is executing alone on a dedicated system. The goal is to have the effect of a group of transactions on the database's state is equivalent to any serial execution of all transactions.

Concurrency Control Schemes

1. **Two-Phase Locking (Pessimistic):** Assume transactions will conflict so they must acquire locks on database objects before they are allowed to access them.
2. **Timestamp Ordering (Optimistic):** Assume that conflicts are rare so transactions do not need to first acquire locks on database objects and instead check for conflicts at commit time.

3 Timestamp Ordering Concurrency Control

Use timestamps to determine the order of transactions.

Basic T/O Protocol

- Every transaction is assigned a unique timestamp when they arrive in the system.
- The DBMS maintains separate timestamps in each tuple's header of the last transaction that read that tuple or wrote to it.
- Each transaction check for conflicts on each read/write by comparing their timestamp with the timestamp of the tuple they are accessing.
- The DBMS needs copy a tuple into the transaction's private workspace when reading a tuple to ensure repeatable reads.

Optimistic Concurrency Control (OCC)

Store all changes in private workspace. Check for conflicts at commit time and then merge. First proposed in 1981 at CMU by H. T. Kung [2].

Three Phases:

- *Read Phase:* Transaction's copy tuples accessed to private work space to ensure repeatable reads, and keep track of read/write sets.
- *Validation Phase:* When the transaction invokes **COMMIT**, the DBMS checks if it conflicts with other transactions. Parallel validation means that each transaction must check the read/write set of other transactions that are trying to validate at the same time. Each transaction has to acquire locks for its write set records in some global order. Original OCC uses serial validation.
  - Backward Validation: Check whether the committing transaction intersects its read/write sets with those of any transactions that have **already** committed.
  - Forward Validation: Check whether the committing transaction intersects its read/write sets with any active transactions that have **not** yet committed.
- *Write Phase:* The DBMS propagates the changes in the transactions write set to the database and makes them visible to other transactions' items. As each record is updated, the transaction releases the lock acquired during the Validation Phase

Timestamp Allocation: [4]

- *Mutex:* Worst option. Mutexes are terrible.
- *Atomic Addition:* Use compare-and-swap to increment a single global counter. Requires cache invalidation on write.
- *Batched Atomic Addition:* Needs a back-off mechanism to prevent fast burn.
- *Hardware Clock:* The CPU maintains an internal clock (not wall clock) that is synchronized across all cores. Intel only. Not sure if it will exist in future CPUs.
- *Hardware Counter:* Single global counter maintained in hardware. Not implemented in any existing CPUs.

4 Silo OCC

The **Silo** DBMS [1, 3] is an influential in-memory DBMS developed by Harvard and MIT. It is a single-node OLTP system that uses Serializable OCC with parallel backward validation. All transactions execute as stored procedures.

The twp key ideas of Silo is that it seeks to avoid all writes to shared-memory for read-only transactions and that it uses batched timestamp allocation using epochs.

Implementation Details:

- **Epochs**
  - Time is sliced into fixed-length epochs (40 ms).
  - All transactions that start within the same epoch will be committed together at the end of the epoch.
  - Transactions that span an epoch have to refresh themselves to be carried over into the next epoch.
  - Worker threads only need to synchronize at the beginning and end of each epoch.
- **Transaction IDs**
  - Each worker thread generates a unique transaction id based on the current epoch number and the next value in its assigned batch.
- **Garbage Collection**
  - Cooperative threads GC.
  - Each worker thread marks a deleted object with a **reclamation epoch**.
- **Range Queries**
  - The DBMSs handles phantoms by tracking the transactions scan set on indexes.
  - Re-execute scans in the validation phase to see whether the index has changed.
  - Have to include virtual entries for keys that do not exist in the index to prevent two threads from trying to insert the same key.

References

[1] Silo - Multicore In-Memory Storage Engine. `https://github.com/stephentu/silo`.

[2] H. T. Kung and J. T. Robinson. On optimistic methods for concurrency control. *ACM Trans. Datab. Syst*, 6(2), June 1981. URL `https://dl.acm.org/citation.cfm?id=319567`.

[3] S. Tu, W. Zheng, E. Kohler, B. Liskov, and S. Madden. Speedy transactions in multicore in-memory databases. In *SOSP '13: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles*, November 2013. doi: https://dl.acm.org/citation.cfm?id=2522713.

[4] X. Yu, G. Bezerra, A. Pavlo, S. Devadas, and M. Stonebraker. Staring into the abyss: an evaluation of concurrency control with one thousand cores. In *VLDB '14: Proceedings of the VLDB Endowment*, volume 8, pages 209–220, November 2014. URL `https://dl.acm.org/citation.cfm?id=2735511`.