HIGH-PERFORMANCE CONCURRENCY CONTROL MECHANISMS FOR MAIN-MEMORY DATABASES

Boren Zang,
David R. Cheriton School of Computer Science
Overview

- Background of MVCC
- Optimistic MVCC
- Pessimistic MVCC
- Evaluation
- Personal reflection
Main Memory Database

- Data reside in memory.
- Support high transaction rates.
- Current concurrency control methods (e.g., Single-version locking) do not always scale.
Multiversion Concurrency Control (MVCC)

- Serialization of transactions
  - Read stability
    - The readability should not change when a transaction tries to read a version of the record.
  - Phantom avoidance
    - Scans do not return new transactions
Lower isolation level than serialization

- Repeatable read: No phantom avoidance.
- Read committed: Only guarantee reads are committed. No validation is required.
- Snapshot isolation: Read as beginning of versions. No validation is required.
Example

Figure 1: Example account table with one hash index. Transaction 75 has transferred $20 from Larry’s account to John’s account but has not yet committed.
TRANSACTION PHASES

Normal processing phase
Preparation phase
Postprocessing phase.
OPTIMISTIC TRANSACTIONS

Validation-based
Normal Processing Phase

- Start scan
  - Record information about indexes and predicates
- Check predicate
- Check visibility
  - May need commit dependency check
- Read version
  - Store versions into a ReadSet for further validation
- Check updatability
  - Updatable: End field equals infinity or it contains a transaction ID and the referenced transaction has aborted
Normal Processing Phase (cont.)

- Update version
  - The transaction creates a new version
  - Set Transaction ID to End Field

- Delete version
  - Update without creating new version
Preparing Phase

- Read validation
  - Read visibility check
  - Check for phantoms

![Diagram showing validation outcomes for V1, V2, V3, and V4 with read and phantom statuses.]

**Figure 3: Possible validation outcomes.**
PESSIMISTIC TRANSACTIONS

Lock-based
Lock Types

- Record Locks
  - Locks on versions
  - Ensure version readability

- Bucket Locks
  - Locks on Buckets
  - Check for phantoms
Wait-for dependencies

- Eagerly update
- Incoming dependency: Wait on other transactions
- Outcoming dependency: Waited by other transactions
- Wait-for graph: Directed graph for deadlock detection
Normal Processing Phase

- Start Scan: A bucketlock is taken out to prevent phantom
- Check predicate
- Check Visibility: Record lock checking
- Read Version: Acquire locks
- Check updatability
- Update Version: Take out wait-for dependencies if the current version is locked
- Delete Version: Same as updating version
- Release locks.
Experiment Setup

- two-socket Intel Xeon X5650 @ 2.67 GHz (Nehalem) that has six cores per socket. Hyper-Threading was enabled. The system has 48 GB of memory, 12 MB L3 cache per socket, 256 KB L2 cache per core, and two separate 32 KB L1-I and L1-D caches per core.
Experimental Results

R=10 and W=2 in each transaction
Table with 10 million rows

Figure 4: Scalability under low contention

R=10 and W=2 in each transaction
Table with 1000 rows

Figure 5: Scalability under high contention
## Different isolation level

<table>
<thead>
<tr>
<th>Tolerance Level</th>
<th>Read Committed tx/sec</th>
<th>Repeatable Read tx/sec</th>
<th>% drop vs RC</th>
<th>Serializable tx/sec</th>
<th>% drop vs RC</th>
</tr>
</thead>
<tbody>
<tr>
<td>1V</td>
<td>2,080,492</td>
<td>2,042,540</td>
<td>1.8%</td>
<td>2,042,571</td>
<td>1.8%</td>
</tr>
<tr>
<td>MV/L</td>
<td>974,512</td>
<td>963,042</td>
<td>1.2%</td>
<td>877,338</td>
<td>10.0%</td>
</tr>
<tr>
<td>MV/O</td>
<td>1,387,140</td>
<td>1,272,289</td>
<td>8.3%</td>
<td>1,120,722</td>
<td>19.2%</td>
</tr>
</tbody>
</table>
Impact of Short Read Transactions

Fixed threads #: 24

Figure 6: Impact of read-only transactions (low contention)

Figure 7: Impact of read-only transactions (high contention)
Impact of Long Read Transactions

**Figure 8:** Update throughput with long read transactions

**Figure 9:** Read throughput with long read transactions
## TATP Results

<table>
<thead>
<tr>
<th>Transactions per second</th>
<th>1V</th>
<th>MV/L</th>
<th>MV/O</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4,220,119</td>
<td>3,129,816</td>
<td>3,121,494</td>
</tr>
</tbody>
</table>
Personal reflection

- No real-world evaluation of serialized level.
- Garbage collection can be a future direction