Staring into the Abyss: An Evaluation of Concurrency Control with One Thousand Cores

Xiangyao Yu\textsuperscript{1} George Bezerra\textsuperscript{1} Andrew Pavlo\textsuperscript{2}
Srinivas Devadas\textsuperscript{1} Michael Stonebraker\textsuperscript{1}

\textsuperscript{1}CSAIL, Massachusetts Institute of Technology
\textsuperscript{2}Dept. of Computer Science Carnegie Mellon University

Published in VLDB 2014

Presenter: Vaibhav Jain
Motivation(1)

- The era of single-core CPU speed-up is over.
- Number of cores on a chip is increasing exponentially
  - Increase computation power by thread level parallelism
  - 1000-core chips are near...

Xeon Phi (up to 61 cores)  

Tilera (up to 100 cores)
Motivation(2)

- Is the DBMS ready to be scaled?
  - Most DBMSs still focus on single-threaded performance
  - Existing works on multi-cores focus on small core count
Objective

• To evaluate transaction processing at 1000 cores.
• Focus on one scalability challenge: Concurrency control.
• Discuss the bottlenecks and improvements needed.
Implementation

• Concurrency Control Schemes
• DBMS TestBed
## Concurrency Control Schemes

<table>
<thead>
<tr>
<th>CC Scheme</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DL_DETECT</td>
<td>2PL with deadlock detection</td>
</tr>
<tr>
<td>NO_WAIT</td>
<td>2PL with non-waiting deadlock prevention</td>
</tr>
<tr>
<td>WAIT_DIE</td>
<td>2PL with wait-and-die deadlock prevention</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>Basic T/O algorithm</td>
</tr>
<tr>
<td>MVCC</td>
<td>Multi-version T/O</td>
</tr>
<tr>
<td>OCC</td>
<td>Optimistic concurrency control</td>
</tr>
<tr>
<td>HSTORE</td>
<td>T/O with partition-level locking</td>
</tr>
</tbody>
</table>

- **Two-Phase Locking (2PL)**
- **Timestamp Ordering (T/O)**
- **Partitioning**
Two-Phase Locking (1)
Two-Phase Locking (2)

- Lock conflict
  - DL_DETECT: always wait.
  - NO_WAIT: always abort.
  - WAIT_DIE: wait if older, otherwise abort

- Example systems
  - Ingres, Informix, IBM DB2, MS SQL Server, MySQL (InnoDB)
## Concurrency Control Schemes

<table>
<thead>
<tr>
<th>CC Scheme</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DL_DETECT</td>
<td>2PL with deadlock detection</td>
</tr>
<tr>
<td>NO_WAIT</td>
<td>2PL with non-waiting deadlock prevention</td>
</tr>
<tr>
<td>WAIT_DIE</td>
<td>2PL with wait-and-die deadlock prevention</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>Basic T/O algorithm</td>
</tr>
<tr>
<td>MVCC</td>
<td>Multi-version T/O</td>
</tr>
<tr>
<td>OCC</td>
<td>Optimistic concurrency control</td>
</tr>
<tr>
<td>HSTORE</td>
<td>T/O with partition-level locking</td>
</tr>
</tbody>
</table>
Timestamp Ordering (T/O) (1)

Each transaction has a unique timestamp indicating the serial order.

1. **TIMESTAMP** *(Basic Timestamp Ordering)*  
   - R/W request rejected if tx timestamp < timestamp of last write.

2. **MVCC** *(Multi-Version Concurrency Control)*  
   - Every write op creates a new timestamped version
   - For read op, DBMS decides which version it accesses.
Timestamp Ordering (T/O) (2)

3. **OCC (Optimistic Concurrency Control)**
   - Private workspace of each transaction.
   - At commit time, if any overlap, tx is aborted and restarted.
   - Advantage: short contention period.

Example systems

Oracle, Postgres, MySQL (InnoDB), SAP HANA, MemSQL, MS Hekaton
# Concurrency Control Schemes

<table>
<thead>
<tr>
<th>CC Scheme</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DL_DETECT</td>
<td>2PL with deadlock detection</td>
</tr>
<tr>
<td>NO_WAIT</td>
<td>2PL with non-waiting deadlock prevention</td>
</tr>
<tr>
<td>WAIT_DIE</td>
<td>2PL with wait-and-die deadlock prevention</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>Basic T/O algorithm</td>
</tr>
<tr>
<td>MVCC</td>
<td>Multi-version T/O</td>
</tr>
<tr>
<td>OCC</td>
<td>Optimistic concurrency control</td>
</tr>
<tr>
<td>HSTORE</td>
<td>T/O with partition-level locking</td>
</tr>
</tbody>
</table>
H-Store

• Database divided into disjoint memory subsets called partitions.
• Each partition protected by locks.
• Tx acquires locks to all partitions it needs to access.
• DBMS assigns it a timestamp and adds it to lock queues.
DBMS Test Bed (1)

Graphite: **CPU simulator**, scales up to 1024 cores.

- Application threads mapped to simulated core threads.
- Simulated threads mapped to multiple processes on host machines.
DBMS Test Bed (2)

• Implemented light-weight pthread based DBMS.
• Allows to swap different concurrency schemes.
• Ensures no other bottlenecks than concurrency control.
• Reports transaction statistics.
General Optimizations

1. **Memory Allocation:**
   Custom malloc, resizable memory pool for each thread.
2. **Lock Table:**
   Instead of centralized lock table, per-tuple locks
3. **Mutexes:**
   Avoid mutex on critical path.
   - For 2PL, centralized deadlock detector
   - For t/o: allocating unique timestamps.
Scalable 2PL

1. **Deadlock Detection**
   - Making deadlock detector lock free by keeping local wait-for graph.
   - Thread searches for cycles in partial wait-for graph.

2. **Lock Thrashing**
   - Holding locks until commit => bottleneck in concurrent Txs.
   - Timeout threshold : abort Tx if wait time exceeds timeout.
Scalable T/O

1. **Timestamp Allocation**
   a) Batched atomic addition
      - Manager returns multiple timestamps for a request.
   b) CPU clocks
      - Read logical clock of core, concatenate with thread id.
      - requires synchronized clocks.
   c) Hardware counters
      - Physically located at center of CPU.
Evaluation
Read-Only Workload

The diagram shows the throughput (in Million transactions per second) of different benchmarks with varying numbers of cores. The benchmarks include:

- DL_DETECT
- TIMESTAMP
- NO_WAIT
- MVCC
- WAIT_DIE
- OCC

The throughput increases linearly with the number of cores for all benchmarks.
Read Only Workload

- 2PL schemes are scalable for read only benchmarks
Read Only Workload

- 2PL schemes are scalable for read only benchmarks
- Timestamp allocation limits scalability
Read Only Workload

- 2PL schemes are scalable for read only benchmarks
- Timestamp allocation limits scalability
- Memory copy hurts performance
Write Intensive (medium contention)

No_Wait, Wait_Die scales better than others.

DL_Detect inhibited by lock thrashing.
Write Intensive (High contention)

- Scaling stops at small core count (64)
Write Intensive (High contention)

- Scaling stops at small core count (64)
- NO_WAIT has good performance but falls due to thrashing.
Write Intensive (High contention)

- Scaling stops at small core count (64)
- NO_WAIT has good performance but falls due to thrashing.
- OCC wins at 1000 cores as one Tx always commits.
More Analysis

1. Short Transactions => Low Lock contention
   Longer Transactions => Timestamp allocation not a bottleneck.


   Partitioned workloads => H-Store best algorithm
# Bottlenecks Summary

<table>
<thead>
<tr>
<th>Concurrency Control</th>
<th>Waiting (Thrashing)</th>
<th>High Abort Rate</th>
<th>Timestamp Allocation</th>
<th>Multi-partition</th>
</tr>
</thead>
<tbody>
<tr>
<td>DL_DETECT</td>
<td>✅</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NO_WAIT</td>
<td></td>
<td>✅</td>
<td></td>
<td></td>
</tr>
<tr>
<td>WAIT_DIE</td>
<td>✅</td>
<td></td>
<td>✅</td>
<td></td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td></td>
<td></td>
<td>✅</td>
<td></td>
</tr>
<tr>
<td>MULTIVERSION</td>
<td></td>
<td></td>
<td>✅</td>
<td></td>
</tr>
<tr>
<td>OCC</td>
<td></td>
<td>✅</td>
<td>✅</td>
<td></td>
</tr>
<tr>
<td>HSTORE</td>
<td>✅</td>
<td></td>
<td>✅</td>
<td>✅</td>
</tr>
</tbody>
</table>
Summary

All algorithms fail to scale as core increases.

- **Thrashing** limits the scalability of 2PL algorithms
- **Timestamp allocation** limits the scalability of T/O algorithms
Project Ideas

• New concurrency control approaches to tackle scalability problem.
• Hardware solutions to DBMS bottlenecks unsolvable in software side.
• Hybrid approach: Switch b/w schemes depending on workload.
Questions
Thrashing

transactions

A

B

C

D

tuples

x

y

z

u

v

Locking

Waiting