CS 839: Design the Next-Generation Database
Lecture 4: Multicore (Part I)

Xiangyao Yu
1/30/2020
Announcements

Email me if you are not in HotCRP
https://wisc-cs839-ngdb20.hotcrp.com

New deadline for submitting paper review:
Before lecture starts

This course is on PhD breadth requirement list

Please talk to me to discuss project ideas
Transactions on column-store
  • Pros: Compression, good for read workload, good for sequential writes
  • Cons: More I/O for row selection/update/insert

Data format for HTAP?
  • Hot data in row format, convert cold data to column format in background
  • Different formats in replicas

Small processor near disk
  • Compression/decompression, encryption, filtering, sorting, hashing, hot data
  • Coalesce random accesses
  • Fast indexing
Staring into the Abyss: An Evaluation of Concurrency Control with One Thousand Cores

Xiangyao Yu
MIT CSAIL
yxy@csail.mit.edu

George Bezerra
MIT CSAIL
gbezerra@csail.mit.edu

Andrew Pavlo
Carnegie Mellon University
pavlo@cs.cmu.edu

Srinivas Devadas
MIT CSAIL
devadas@csail.mit.edu

Michael Stonebraker
MIT CSAIL
stonebraker@csail.mit.edu
Lesson learned: Talk to people about your research
Many-core systems have arrived

- The era of single-core CPU speed-up is over
- Number of cores on a chip is increasing exponentially
  - 1000-core chips are a near…
- DBMSs are not ready
  - Most DBMSs still focus on single-threaded performance
  - Existing works on multi-cores focus on small core count

Xeon Phi (up to 61 cores)

Tilera (up to 100 cores)
Many-core systems have arrived
Databases on 1000-core systems

- DBMS on future computer architectures
- Will DBMSs scale to this level of parallelism?

All classic concurrency control algorithms fail to scale to 1000 cores.

- What are the main bottlenecks to scalability?
- What improvements will be needed from the software and hardware perspectives?
1000-Core DBMS

- On Line Transaction Processing (OLTP)
- Concurrency control is a key limiting factor to the scalability
- new database: **DBx1000**
  - Support all seven classic concurrency control algorithms
  - Study the fundamental bottlenecks
    - https://github.com/yxymit/DBx1000
- Graphite Multi-core Simulator
Simulated Hardware

- **CPU:** 1024 in-order core
- **Cache:** 32KB L1, 512KB L2
- **Network:** 2D-mesh
Graphite Simulator[1]
# 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**
Wait-for Graph:

T1 <---- T2 when T2 waits for a lock held by T1

Periodically, detect cycles in the graph and abort the transaction that holds the fewest locks
2PL – NO_WAIT, WAIT_DIE

**NO_WAIT:** A transaction cannot wait for another transaction. Whenever two transactions conflict, the requesting transaction aborts.

**WAIT_DIE:** A transaction T1 waits for another transaction T2 only if T1 has higher priority than T2 (e.g., T1 starts execution before T2).

Pros over NO_WAIT
- Guaranteed forward progress (i.e., no starvation)
- Fewer aborts

Cons over NO_WAIT
- Locking logic is more complex
Each transaction is assigned a unique timestamp indicating the serial order.
Each transaction is assigned a unique timestamp indicating the serial order.

Read from T (T.ts. = 5)
Timestamp Ordering – Basic

Each transaction is assigned a unique timestamp indicating the serial order.

Read from T (T.ts. = 25)

wts=10  rts=20

Timestamp Order
Each transaction is assigned a unique timestamp indicating the serial order.

Read from T (T.ts. = 25)
Timestamp Ordering – Basic

Each transaction is assigned a unique timestamp indicating the serial order.

Write from T (T.ts. = 15)
Each transaction is assigned a unique timestamp indicating the serial order.

Write from T (T.ts. = 5)
Each transaction is assigned a unique timestamp indicating the serial order.

Write from T (T.ts. = 25)
Each transaction is assigned a unique timestamp indicating the serial order.

Write from T (T.ts. = 25)
Timestamp Ordering – MVCC

MVCC: Multi-Version Concurrency Control

Read from T (T.ts. = 5)

wts=10   rts=20

Timestamp Order
Timestamp Ordering – MVCC

MVCC: Multi-Version Concurrency Control

A transaction can read previous versions
Pros:

• Timestamp order is the serialization order
• Logic for locking is simplified
• In MVCC, read-only and read-write transactions do not conflict

Cons:

• Timestamp allocation is a bottleneck
Pessimistic/Optimistic vs. 2PL/TO
Partition-Level Locking – H-store

**Pro:** Only one lock per partition

**Con:** Performance degrades for multi-partition transactions
Partition-Level Locking – H-store

![Graph showing throughput in million transactions per second against percentage of multi-partition transactions]

- **Single Partition Transaction**
- **Multi Partition Transaction**
Evaluation – Experimental Setup

Yahoo! Cloud Serving Benchmark (YCSB)

- 20 million tuples
- Each tuple is 1KB (total database is ~20GB)

Each transaction reads/modifies 16 random tuples following a skewed pattern

Serializable isolation level
Evaluation – Readonly

2PL schemes are scalable for read-only benchmarks
2PL schemes are scalable for read-only benchmarks

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

Timestamp allocation limits scalability

Memory copy hurts performance
Evaluation – Medium Contention

Write : Read = 50% : 50%

DL_DETECT does not scale due to deadlocks and thrashing
Evaluation – High Contention

Scaling stops at small core count
Scaling stops at small core count

NO_WAIT has good performance until 1000 cores
Scaling stops at small core count

NO_WAIT has good performance until 1000 cores

OCC wins at 1000 cores
# Scalability Bottlenecks

<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>
Solutions to Timestamp Allocation

Mutex based allocation
Mutex based allocation
Atomic instruction
Solutions to Timestamp Allocation

Mutex based allocation
Atomic instruction
Batch allocation
Solutions to Timestamp Allocation

- Mutex based allocation
- Atomic instruction
- Batch allocation

Hardware Counter (~1000 million ts/s)
Solutions to Timestamp Allocation

- Mutex based allocation
- Atomic instruction
- Batch allocation

**Hardware Counter (~1000 million ts/s)**

**Distributed Clock (perfect scalability)**

- All clocks must be synchronized
1000-core – Q/A

Why 1000?

Workload realistic?

Simulator (Graphite) realistic?

Distributed transactions?
  • Similar conclusions

Abyss removed?
Summary

Core counts will keep increasing

Conventional concurrency control protocols do not scale
  • Lock trashing
  • Timestamp allocation

Need software hardware codesign

(software-only solutions can go a long way)
Group Discussion

What are the pros and cons of timestamp ordering over two-phase locking? Can you think of other examples of using timestamps in other fields of CS?

What are the main pros and cons of a multi-version concurrency control (MVCC) protocol? How is MVCC related to HTAP (Hybrid transactional/analytical processing)?

Can you think of any hardware changes to a multicore CPU that can improve the performance/scalability of concurrency control?
Before Next Lecture

Submit discussion summary to https://wisc-CS839-NGdb20.hotcrp.com
• Deadline: Friday 11:59pm

Submit review for

Speedy Transactions in Multicore In-Memory Databases
[optional] TicToc: Time Traveling Optimistic Concurrency Control