Toward Coordination-free and Reconfigurable Mixed Concurrency Control

Dixin Tang                                               Aaron J. Elmore

THE UNIVERSITY OF
CHICAGO
Hardware Development Changes Database Architecture

Disk-based Database

- A Few Cores
- Small Memory
- Large Disk

Main Memory Database

- More Concurrent Read/Write Operations
- Concurrency Control Becomes the New Bottleneck
- Elimination of Disk Stalls

Disk Stalls Dominate the Performance
A Closer Look at Concurrency Control

Transaction
A sequence of read/write operations to be executed atomically

Working Threads

Concurrency Control Comes Into Play

Two Goals of Concurrency Control

• Interleaving concurrent operations to maximize the performance
• Guarantee consistency

Serializability
The result of interleaved execution of concurrent transactions is equivalent to the result of executing these transactions in one serial order
PartCC (Partition-based single-thread concurrency control)

- Perform well under partitionable workloads
- Cross-partition transactions hurt the performance
One Concurrency Control Does not Fit All

**OCC** (Optimistic Concurrency Control)
- Perform better under low-conflict workloads
- Conflicts can hurt the performance because transactions need to be restarted if validation fails

**2PL** (Two Phase Locking)
- Perform better under highly-conflicted workloads
- Concurrency control overhead and synchronization overhead
experiment on our main-memory database prototype using YCSB workloads

- OCC from Silo [1]
- 2PL using VLL [2]
- PartCC from H-Store [3]

Each Protocol excels in different scenarios.

---

A General Solution: Mixed Concurrency Control

Concurrency Control Protocols

Database

Workload

$CC_1$ $CC_2$ $CC_3$

Two Benefits

• Each protocol can process the part of workload it is optimized for

• Each protocol can avoid being brittle to workload where it does not perform well
Two Challenges of Mixed Concurrency Control

- How to partition a workload and mix multiple concurrency control protocols efficiently

- How to reconfigure a protocol when the workload changes
Previous Approach of Mixed Concurrency Control: Partition Stored Procedures by Conflicts

**Stored Procedure (SP)**
A Parameterized Transaction Template

SQL

Semantic Information
- Tables to be read/written
- Orders of reading/writing tables

Generate transactions by providing parameters

**Previous Approach [1]**
- Group stored procedures by conflicts extracted from their semantic information
- Assign each group a protocol to process conflicts within that group
- Need additional concurrency control protocols to process conflicts across groups

Previous Approach of Mixed Concurrency Control: Partition Stored Procedures by Conflicts

Overhead of multiple CC execution for a single operation

Rely on static semantic information, cannot adapt to varied workloads

Process conflicts across groups

Process conflicts within group
A New Perspective: Partition Records by Access Characteristics

Partition-based single-thread approach

Lowly-conflicted Records

Highly-conflicted Records

Partitionable

Validation-based Approach

Lock-based Approach
CormCC: Coordination-free and Reconfigurable Mixed Concurrency Control

Our Approach

- Partition database records and assign each partition a single protocol
- A single protocol is used to process all operations for that partition of records

To achieve the two goals, we need to answer four questions:

- How does CormCC execute
- How to maintain serializability
- How to guarantee deadlock free
- How to enable online protocol switch

Reconfigurable: partitioning records depend on real-time data access characteristics, make online protocol reconfiguration possible
CormCC Execution

Break transaction life cycle into four phases

Preprocess | Execution | Validation | Commit
---|---|---|---
TXN | R TXN | N TXN 2 | TXN
TXN | TXN | TXN | TXN
TXN | W TXN | TXN | TXN
TXN | W | TXN | TXN

CC1
CC2
CC3
Correctness of CormCC – Serializability

- COCSR (Commit ordering conflict serializable)
  - Sufficient condition of serializable
  - Commit ordering respects conflicts
- If all protocols are COCSR, then CormCC is COCSR
  - Proof can be found in the paper

![Time Line]

- TXN1: Read One Record
- TXN1 Commits before TXN2
- TXN1: Commit
- TXN2: Write
- TXN2: Commit
CormCC – Deadlock Free

Deadlock Detection

Our Approach: Deadlock Prevention

- We require each protocol can only exclusively let transactions wait in no more than one phase
  - No deadlock within one phase
  - Transactions in earlier phases can wait for later phases, but not the other way around

Diagram:
- Preprocess
- Execution
- Validation
- Commit

Transactions: TXN, CC1, CC2, CC3
- TXN1, TXN2, TXN3
CormCC – Online Protocol Reconfiguration

Online reconfiguration can cause inconsistency

A straightforward solution: stop all

- Waiting all working threads to complete their current transactions, and stop them from receiving new transactions
- Decrease the performance of database
Our solution: using a mediated protocol that is compatible to both old and new protocols.
CormCC – Online Protocol Reconfiguration

How to Build a Mediated Protocol

- The mediated protocol executes the logics of both old and new protocol

- Example: Mediated Protocol between OCC and 2PL

Mediated Protocol between OCC and 2PL

Read
- Read timestamp (OCC)
- Apply read lock (2PL)

Write
- Write to a local buffer (OCC)
- Apply write lock (2PL)

Validate
- Execute Validate phase of OCC
Prototype Design

- Supporting PartCC from H-Store [3], OCC from Silo [1], and 2PL from VLL [2]

- Partition the whole database and apply each partition a single protocol

References:

Prototype Design

- Selecting the ideal protocol for each partition
  - Feature Engineering: design several features to capture the performance difference of candidate protocols
  - Classifiers: building a two-layer classifier
Experiments

Experiment Settings

- A Machine with 32 cores, 256 GB main memory

Workload

- TPC-C: Order processing application with 5 stored procedures, 32 warehouses
- YCSB: One Table with 1 million records, each with 25 columns and 20 bytes for each column; One type of transaction mixed with read or read-and-modify operations
- Partition into 32 partitions (i.e. equal to the number of cores)
Experiments

Our experiments compare CormCC with

- Single candidate protocols
- Tebaldi - A stored procedure-oriented general framework of mixed concurrency control [3]

---

[2] SHANG, Z., et al. Graph analytics through fine-grained parallelism. SIGMOD’16
Experiments – Compare with Tebaldi

Comparison under different partitionability (TPC-C)

- Start with well-partitionable workload
  - Each partition receives 100% single-partition transactions
- Increase the number of partitions receiving 100% cross-partition transactions

Comparison under different partitionability

Throughput (Millions/sec)

Number of partitions that receive 100% cross-partition transactions
Experiments – Compare with Tebaldi

Comparison under different conflict rates (TPC-C)

- We modify TPC-C to increase access skewness
- We use Zipfian distribution and increase its theta parameter to introduce higher conflict rate

![Comparison under different conflict rates](image)

Comparison under different conflict rates

- 2PL
- Tebaldi
- CormCC

Throughput (Millions/sec)

Theta of Zipfian Access Distribution

- 0
- 0.3
- 0.6
- 0.9
- 1.2
- 1.5
Experiments – Varied workloads

- Vary parameters every 5 seconds:
  - Transactions mix,
  - Percentages of cross-partition transactions,
  - Access skewness (i.e. theta of Zipf)

Tests over varied workloads (TPC-C)

- PartCC
- OCC
- 2PL
- Hybrid
- CormCC
Experiments – A closer look at the same results of varied workload tests

- We aggregate the throughput of every 5s and report CormCC throughput ratio to single protocols
  - Max: Speedup over the worst single protocols
  - Min: Speedup over the best single protocols
  - Average: Speedup over average throughput of single protocols

CormCC throughput ratio to single protocols

<table>
<thead>
<tr>
<th>Speedup over Single Protocols</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>5 10 15 20 25 30 35 40 45 50 55 60 65 70 75 80 85 90 95 100</td>
</tr>
<tr>
<td>Min</td>
<td></td>
</tr>
<tr>
<td>Average</td>
<td></td>
</tr>
</tbody>
</table>
Experiments – Mediated Protocol Switch

- Switch from OCC to 2PL using YCSB workloads
- Report the throughput during protocol switch
- Compared with StopALL
- Test a workload with short-only transactions and workloads with one long transaction of different duration
  - 0.5s, 1s, 2s, 4s
- Test different switching points
  - At the beginning of the long transaction
  - At the middle of the long transaction
  - At the end of the long transaction

Throughput (Millions/sec)

Testing Mediated Switching

<table>
<thead>
<tr>
<th>Throughput (Millions/sec)</th>
<th>Short-only</th>
<th>Long(0.5s)</th>
<th>Long(1 s)</th>
<th>Long(2 s)</th>
<th>Long(4 s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>OCC</td>
<td>Blue</td>
<td>Red</td>
<td>Green</td>
<td>Purple</td>
<td>Purple</td>
</tr>
<tr>
<td>2PL</td>
<td>Red</td>
<td>Blue</td>
<td>Green</td>
<td>Purple</td>
<td>Purple</td>
</tr>
<tr>
<td>StopALL</td>
<td>Green</td>
<td>Purple</td>
<td>Blue</td>
<td>Red</td>
<td>Red</td>
</tr>
<tr>
<td>Mediated Switch</td>
<td>Purple</td>
<td>Blue</td>
<td>Red</td>
<td>Green</td>
<td>Green</td>
</tr>
</tbody>
</table>

At the beginning of the long transaction
At the middle of the long transaction
At the end of the long transaction
Conclusion

- CormCC, a general mixed concurrency control framework that does not introduce any coordination overhead and supports online reconfiguration.

- Experiments show that CormCC can achieve significant throughput improvement over single static protocols and state-of-the-art mixed approaches.

Thanks!
Backup Slides
CormCC Execution in Prototype

<table>
<thead>
<tr>
<th></th>
<th>Preprocess</th>
<th>Execution</th>
<th>Validation</th>
<th>Commit</th>
</tr>
</thead>
<tbody>
<tr>
<td>PartCC</td>
<td>✓</td>
<td>x</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>OCC</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>2PL</td>
<td></td>
<td>✓</td>
<td>x</td>
<td>✓</td>
</tr>
</tbody>
</table>

- ✓ This protocol includes this phase
- x This protocol makes transaction wait in this phase
Experiments – Varied workloads (YCSB)

- Vary parameters every 5 seconds:
  - Transactions mix, Percentages of cross-partition transactions, Access skewness (i.e. theta of Zipf)

Tests over varied workloads (YCSB)

- PartCC
- OCC
- 2PL
- Hybrid
- CormCC
Experiments – Aggregated Results of Varied workloads (YCSB)

CormCC throughput ratio to single protocols (YCSB)

- Max
- Min
- Average

Throughput (Millions/sec) vs Time (s)