Leveraging Lock Contention to Improve Transaction Applications

Cong Yan     Alvin Cheung
University of Washington
Background

• Database transactions
  • Airline ticket reservation, banking, online shopping...
Background

• Database transactions
  • Airline ticket reservation, banking, online shopping...
Background

- Database transactions
  - Parallelism under high data contention? atomicity and consistency
  - Concurrency protocols:
Two-phase Locking

• Example transaction: online shopping

• Alice(A) and Cong(C) buying Echo at the same time

```
select(Echo)
update(Echo.num)
sel
select(Alice)
update(Alice.bal)
```
Two-phase Locking

- Example transaction: online shopping
  - Alice(A) and Cong(C) buying Echo at the same time

```
select(Echo)
update(Echo.num)
select(Alice)
update(Alice.bal)
```

```
select(Echo)
update(Echo.num)
select(Cong)
update(Cong.bal)
```
Two-phase Locking

Execution time

--(T1)--

select(Echo)

--(T2)--
Two-phase Locking

Execution time

---(T1)---
- select(Echo)
- update(Echo.num)

---(T2)---
- select(Echo)

X
Two-phase Locking

---(T1)---
- select(Echo)
- update(Echo.num)
- select(Alice)

---(T2)---
- select(Echo)

(Waiting for lock)
Two-phase Locking

Execution time

--(T1)--
- select(Echo)
- update(Echo.num)
- select(Alice)
- update(Alice.bal)

--(T2)--
- (Waiting for lock)
- select(Echo)

Waiting for lock
Two-phase Locking

Execution time

--(T1)--

select(Echo)

update(Echo.num)

select(Alice)

update(Alice.bal)

--(T2)--

(Waiting for lock)

select(Echo)
Two-phase Locking

Execution time

--(T1)--

select(Echo)

update(Echo.num)

select(Alice)

update(Alice.bal)

--(T2)--

(Waiting for lock)

select(Echo)
Two-phase Locking

Execution time

---(T1)---

- select(Echo)
- update(Echo.num)
- select(Alice)
- update(Alice.bal)

---(T2)---

- select(Echo)
- update(Echo.num)

(Waiting for lock)
Two-phase Locking

Execution time

---(T1)---

- select(Echo)
- update(Echo.num)
- select(Alice)
- update(Alice.bal)

---(T2)---

(update(Echo.num)
select(Echo)
select(Cong)

(Waiting for lock)
Two-phase Locking

Execution time

---(T1)---
- select(Echo)
- update(Echo.num)
- select(Alice)
- update(Alice.bal)

---(T2)---
- select(Echo)
- update(Echo.num)
- select(Cong)
- update(Cong.bal)

(Waiting for lock)
Two-phase Locking

- Problem: serialization of all transactions
  - Changing the order of queries within transactions shortens lock waiting time

- Other concurrency control protocols: OCC, MVCC
  - 2PL is more efficient under high data contention(*)

---

*: “Staring into the abyss, An Evaluation of Concurrency Control with One-thousand Cores”, VLDB-14
Shortening Lock Waiting

Execution time

--(T1)--

- select(Echo)
- update(Echo.num)
- select(Alice)
- update(Alice.bal)

--(T2)--

- select(Echo)
- update(Echo.num)
- select(Alice)
- update(Alice.bal)

--(T1)--

- select(Alice)

--(T2)--

- select(Cong)
- update(Cong.bal)

(Waiting for lock)
Shortening Lock Waiting

Execution time

--(T1)--

select(Echo)
update(Echo.num)
select(Alice)
update(Alice.bal)

--(T2)--

(Waiting for lock)

--(T1)--

select(Alice)

--(T2)--

select(Cong)
update(Cong.num)
select(Cong)
update(Cong.bal)
Shortening Lock Waiting

Execution time

--(T1)--

select(Echo)
update(Echo.num)
select(Alice)
update(Alice.bal)

--(T2)--

select(Alice)
update(Alice.bal)

--(T1)--

select(Echo)
update(Echo.num)
select(Cong)
update(Cong.bal)

--(T2)--

select(Cong)
Shortening Lock Waiting

---(T1)---
select(Echo)
update(Echo.num)
select(Alice)
update(Alice.bal)

---(T2)---
(update for lock)
select(Echo)
select(Cong)
(update for lock)
select(Alice)
update(Alice.bal)

---(T1)---
select(Cong)
update(Cong.bal)

---(T2)---
select(Alice)
update(Alice.bal)

Execution time
Shortening Lock Waiting

Execution time

--(T1)--
- select(Echo)
- update(Echo.num)
- select(Alice)
- update(Alice.bal)

--(T2)--
- select(Echo)
- update(Echo.num)
- select(Alice)
- update(Alice.bal)

--(T1)--
- select(Alice)
- update(Alice.bal)
- select(Echo)
- update(Echo.num)

--(T2)--
- select(Cong)
- update(Cong.bal)
- select(Echo)
Shortening Lock Waiting

Execution time

--(T1)--

select(Echo)
update(Echo.num)
select(Alice)
update(Alice.bal)

--(T2)--

select(Echo)
update(Echo.num)
select(Alice)
update(Alice.bal)

--(T1)--

select(Alice)
update(Alice.bal)
select(Echo)
update(Echo.num)

--(T2)--

select(Cong)
update(Cong.bal)
select(Echo)

(Waiting for lock)

(Waiting for lock)
Shortening Lock Waiting

Execution time

---(T1)---
- select(Echo)
- update(Echo.num)
- select(Alice)
- update(Alice.bal)

---(T2)---
- (Waiting for lock)

---(T1)---
- select(Alice)
- update(Alice.bal)
- select(Echo)
- update(Echo.num)

---(T2)---
- select(Cong)
- update(Cong.bal)
- select(Echo)
- update(Echo.num)

(Waiting for lock)
Shortening Lock Waiting

More threads...
Shortening Lock Waiting

# Shorter latency
# Higher throughput

More threads...
• Manually reordering queries is hard
• QURO: a query-aware compiler
  • Automatically reorders queries in transactions based on data contention
  • Preserves original program semantics
**QURO**

**Input:** C++ transaction code with embedded SQL queries

- Profile the application
- Analyze application code
- Reorder queries

**Output:** C++ code with reordered SQL queries
Profile the applications

Know which queries are likely to access contentious data

Calculate the variance of running time for each query
• Analyze application code

• Reordering preserves program semantics

• Data dependency among program variables

1. \( v_1 = \text{select} \left( \text{"table1"} \right) \);  
2. \( v_2 = \text{select} \left( \text{"table2"}, v_1 \right) \);  
3. \( \text{update} \left( \text{"table1"}, \text{input} \right) \);

Statement 1 should appear before statement 2

• Database constraints (same table, view, foreign key...)

1. \( v_1 = \text{select} \left( \text{"table1"} \right) \);  
2. \( v_2 = \text{select} \left( \text{"table2"}, v_1 \right) \);  
3. \( \text{update} \left( \text{"table1"}, \text{input} \right) \);

Statement 1 should appear before statement 3
• Goal: contentious queries appear as late as possible in transactions
• Constraint: data dependencies & database constraints
QURO

• Goal: contentious queries appear as late as possible in transactions

• Constraint: data dependencies & database constraints

Optimization problem!
QURO

- Goal: contentious queries appear as late as possible in transactions
- Constraint: data dependencies & database constraints

Optimization problem!

- Formalize into ILP
- Optimizations: reordering long transactions within seconds
Evaluation

- Experiment overview:
  - Benchmarks: TPC-C, TPC-E
  - Throughput: original Vs. reordered implementation (by QURO)
  - Increasing data contention
    - smaller data size
    - more threads
Evaluation

- Benchmark: TPC-C payment transaction
- changing data size
- scaling to more threads

contention                              contention

latency: -83%                            latency: -70%

transactions/sec                         transactions/sec

<table>
<thead>
<tr>
<th>number of warehouses</th>
<th>reordered</th>
<th>original</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>6.53x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>6.11x</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>3.52x</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2.01x</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>1.35x</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>1.23x</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>number of trades</th>
<th>reordered</th>
<th>original</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.05x</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1.09x</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1.29x</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>2.06x</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>3.09x</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>3.52x</td>
<td></td>
</tr>
<tr>
<td>64</td>
<td>2.49x</td>
<td></td>
</tr>
<tr>
<td>128</td>
<td>2.10x</td>
<td></td>
</tr>
</tbody>
</table>
Evaluation

- Benchmark: TPC-E trade update transaction
- changing data size
- scaling to more threads

contention

<table>
<thead>
<tr>
<th>number of trades</th>
<th>transactions/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>1K</td>
<td>4.35x</td>
</tr>
<tr>
<td>4K</td>
<td>3.35x</td>
</tr>
<tr>
<td>16K</td>
<td>1.79x</td>
</tr>
<tr>
<td>64K</td>
<td>1.36x</td>
</tr>
<tr>
<td>256K</td>
<td>1.15x</td>
</tr>
<tr>
<td>&gt;512K</td>
<td>1.14x</td>
</tr>
</tbody>
</table>

contention

<table>
<thead>
<tr>
<th>number of trades</th>
<th>transactions/sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1.02x</td>
</tr>
<tr>
<td>2</td>
<td>1.24x</td>
</tr>
<tr>
<td>4</td>
<td>1.36x</td>
</tr>
<tr>
<td>8</td>
<td>1.63x</td>
</tr>
<tr>
<td>16</td>
<td>2.74x</td>
</tr>
<tr>
<td>32</td>
<td>3.35x</td>
</tr>
<tr>
<td>64</td>
<td>3.51x</td>
</tr>
<tr>
<td>128</td>
<td>2.93x</td>
</tr>
</tbody>
</table>

latency: -75%

latency: -66%
Experiments

- Mix of different transactions
- TPC-C standard mix: 5 types of transactions

![Graph showing performance comparison between reordered and original transactions across different numbers of trades. The graph displays a x-axis for number of trades (1 to 128) and a y-axis for transactions per second. The legend indicates two categories: reordered and original. The graph includes bars for each number of trades, with labels indicating performance increase factors (1.06x, 1.09x, 1.04x, 1.08x, 1.12x, 1.08x, 1.18x) for reordered vs original transactions.]
Experiments

- More complicated transactions
- TPC-E trade order and result
- Each >500 lines of code, >20 queries, complicated logic
“AND TO THINK, JUST THE OTHER DAY I WAS WORRIED ABOUT STRANGERS TOUCHING MY JUNK AT THE AIRPORT.”
"AND TO THINK, JUST THE OTHER DAY I WAS WORRIED ABOUT STRANGERS TOUCHING MY JUNK AT THE AIRPORT."
Better black friday!
Conclusion

• The order of query has large impact on transaction performance.

• QURO leverages information about query contention, and automatically reorders the queries.

• Reordered code generated by QURO can improve throughput up to 6.53x, and can be applied to a wide range of applications.

• We are in the process of releasing code (congy@cs.washington.edu).