Infinite Resources for Optimistic Concurrency Control with NOCC

Theo Jepsen, Leandro Pacheco de Sousa, Masoud Moshref, Fernando Pedone, Robert Soulé

Università della Svizzera italiana (USI) and Barefoot Networks
Why Do We Need Concurrency Control?

Deposit $100

50 + 100

Account Balance

$50

Account Balance

$150

50 + 100

Should be $250

Deposit $100
Pessimistic Concurrency Control

Deposit $100

50 + 100

Account Balance
$50

Account Balance
$150

Account Balance
$250

Locks hurt concurrency

150 + 100

Deposit $100
Optimistic Concurrency Control

Account Balance
$50

Account Balance
$150

Account Balance
$250

Read balance
50 + 100
(if balance is still $50)

Read balance
50 + 100
(if balance is still $50)

Read balance
150 + 100
(if balance is still $150)

Abort!

Retry
Pessimistic vs Optimistic Concurrency Control

Limitations of Concurrency Processing

PETTER FRANASZEK and JOHN T. ROE
IBM Thomas J. Watson Research Center

Given that OCC is better...

Concurrency Control Performance Modeling: Alternatives and Implications

RAKESH AGRAWAL
AT&T Bell Laboratories
MICHAEL J. CAREY and MIRON LIVNY
University of Wisconsin

A number of recent studies...
## Pessimistic vs Optimistic Concurrency Control

<table>
<thead>
<tr>
<th></th>
<th>Pessimistic</th>
<th>Optimistic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Low contention</td>
<td>🙁</td>
<td>😃</td>
</tr>
<tr>
<td>High contention</td>
<td>😃</td>
<td>🙁</td>
</tr>
</tbody>
</table>

*Aborts reduce throughput*
OCC: Aborts are Expensive

more clients → contention → more aborts → lower tput
OCC With Infinite Resources

- What if we had infinite CPUs to abort transactions?
- Hardware can process aborts virtually instantly
- This hardware is already in the network
Network OCC (NOCC)

- Offload transaction verification to the switch
- High parallelism for high-contention workloads
- Reduces server load for workloads (like TPC-C)
System Model

R() \rightarrow \text{cache}

R() \rightarrow \text{cache}

\text{CMP()}, \ W() \rightarrow \text{system}

\text{system} \rightarrow \text{Balance}

\text{Balance} \rightarrow \$50
The NOCC Approach

- Update cache with write values
- Update cache with ABORT values
- Early abort invalid transactions
NOCC Example

R()

CMP(50), W(150)

R()

ABORT(50)

Balance

$50

50

50

12
NOCC Correctness

- **Strong consistency:**
  - Reads are not handled by switch – no stale reads

- **Liveness:**
  - Transactions eventually commit
Implementation
Switch Implementation: Key Challenges

- Storing cached values on the switch
- Processing packet headers containing transactions
Processing Transactions

- Each transaction contains one or more operations:
  - read(), cmp(), write()

- The P4 program iterates over the operations:
  - If invalid cmp(), abort transaction
  - If write(), update cache

- P4 doesn’t have iteration primitives
  - So we recirculate the packet
Switch Cache

- We use SRAM registers
- Values (128 bits) are too large for a single register
  - So we shard the value across multiple registers

<table>
<thead>
<tr>
<th>Reg1</th>
<th>Reg2</th>
<th>Reg3</th>
<th>Reg4</th>
</tr>
</thead>
<tbody>
<tr>
<td>val[0...31]</td>
<td>val[32...63]</td>
<td>val[64...95]</td>
<td>val[95...128]</td>
</tr>
</tbody>
</table>
Evaluation on Hardware
Experimental Setup

- Clients and store run on separate servers
- Connected via a Barefoot Tofino switch running NOCC
- Evaluated with microbenchmarks and TPC-C
NOCC has Higher Throughput
NOCC Reduces End-to-End Latency
NOCC Reduces Aborts from the Store

Commits all transactions
NOCC Reduces Server Load for TPC-C
Minimal Throughput Overhead for TPC-C
In Conclusion, NOCC...

- Offloads transaction verification logic to the network
- Provides high throughput under high contention
- Reduces CPU load on the server

https://github.com/usi-systems/nocc
Extra Slides
Packet Header Format

header_type nocc_hdr_t {
    fields {
        bit<1> msg_type; // REQ/RES
        bit<1> from_switch;
        bit<32> txn_id;
        bit<8> frag_seq;
        bit<8> frag_cnt;
        bit<8> status;
        bit<8> op_cnt;
    }
}

The nocc_hdr is followed by a nocc_op header for each operation

header_type nocc_op_t {
    fields {
        bit<8> op_type;
        bit<32> key;
        bit<1024> value;
    }
}

Number of following nocc_op headers