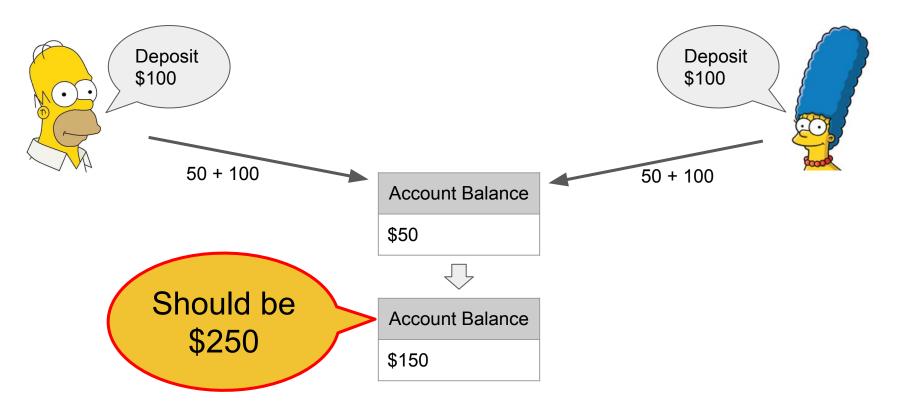
Infinite Resources for Optimistic Concurrency Control with NOCC

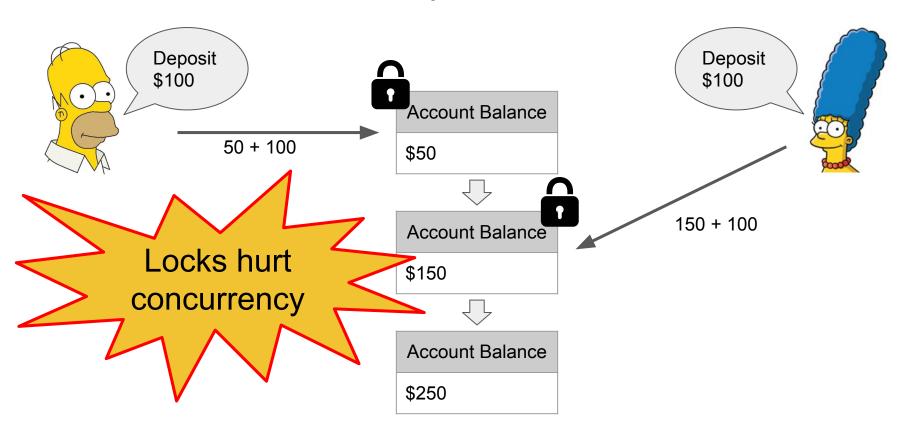
Theo Jepsen, Leandro Pacheco de Sousa, Masoud Moshref, Fernando Pedone, Robert Soulé

Università della Svizzera italiana (USI) and Barefoot Networks

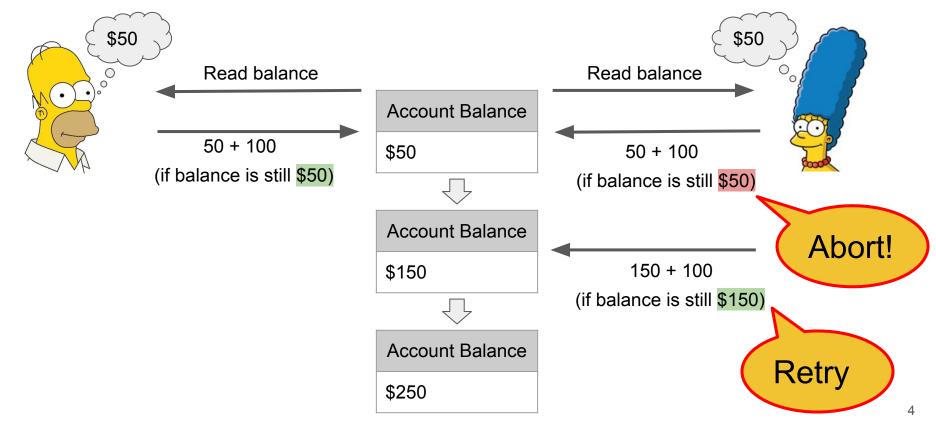
Why Do We Need Concurrency Control?



Pessimistic Concurrency Control



Optimistic Concurrency Control



Pessimistic vs Optimistic Concurrency Control

Limitations of Concurrer Concurrency Control Performance Modeling: Processing

PETER FRANASZEK and JOHN TAROE

IBM Thomas J. Watson Research hter

together with better! E(n,p) is define

do usef concurrency control $E(n, p) \leq$ uling, and (3) optimistic me and (3) $1 + ((1-p)/p)\ln(p(n))$ $(1) + 1) \leq C(n, p) \leq$ $n \to \infty$, (1) $E \to 0$ for standard locking methods, (2)

Pessimistic This paper of tralized concu algorithm-independent is better! developed in ord

Alternatives and Implications

RAKESH AGRAWAL

AT&T Bell Laboratories

MICHAEL J. CAREY and MIRON LIVNY

University of Wisconsin

A number of recent studie database managen to be contradictory. In we critically investigate the tions. We employ a

performance of the amerent ap modeling assumptions. The three conflicts are dealt with, and the amption resources, how transaction restarts are m concurrency control algorithm about transa

It depends...

essed

and the

algorithms for nive, have tended om performance study," and their implicahe relative oblem under a variety of emes in how transaction to the natural f the database system's hount of information available to the ns' reference trings. We show that differences in the

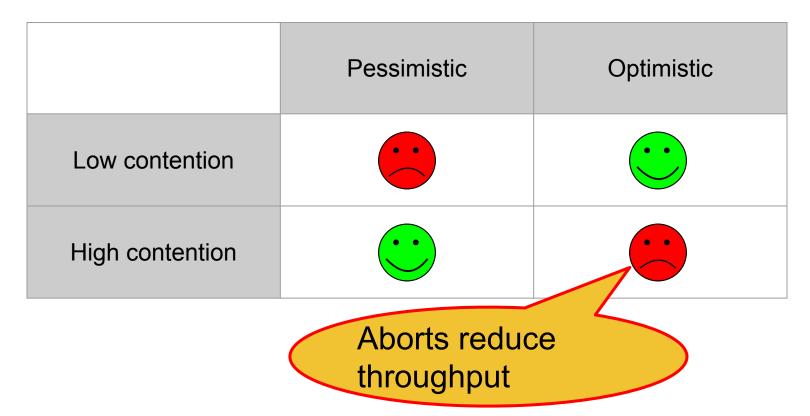
leads to a rule of thumb on how much data contention should be permitted in a system. Throughput Gilcan exceed this bound if a transaction is restarted whenever it encounters a conflict, provided restart control algorithms, the database system

this framework in detail results which were obtain a for wh

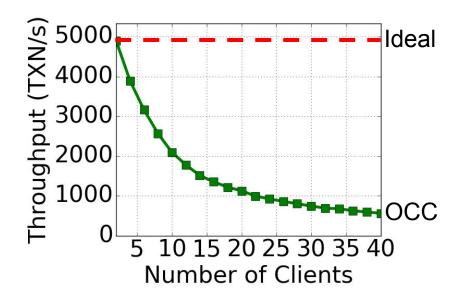
various concurrency contro

Given b

Pessimistic vs Optimistic Concurrency Control



OCC: Aborts are Expensive



more clients→contention→more aborts→lower tput

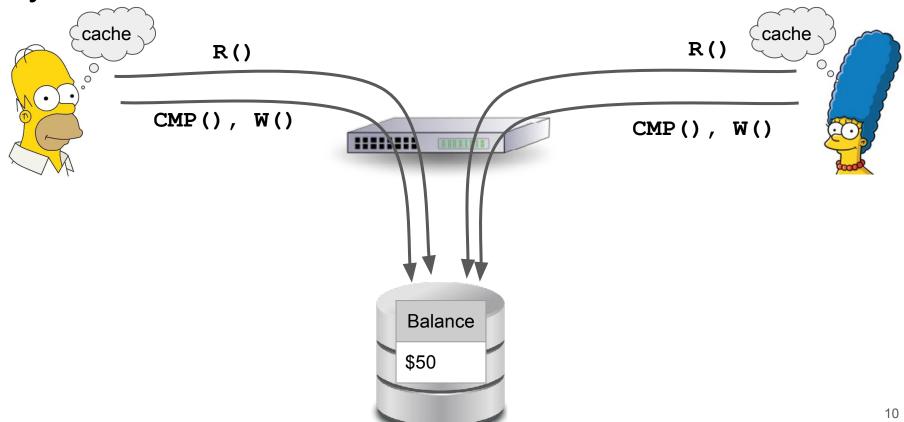
OCC With Infinite Resources

- 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)

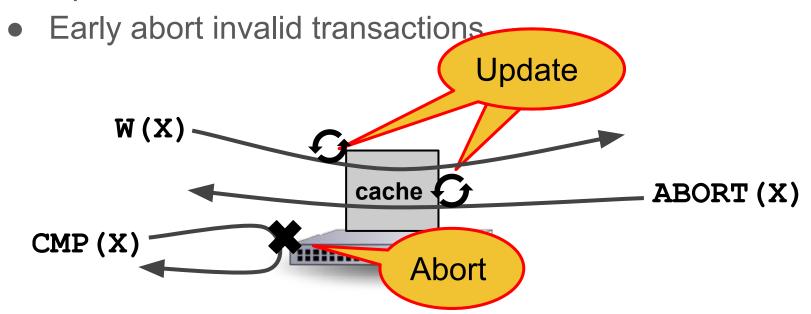
- Offload transaction verification to the switch
- High parallelism for high-contention workloads
- Reduces server load for workloads (like TPC-C)

System Model

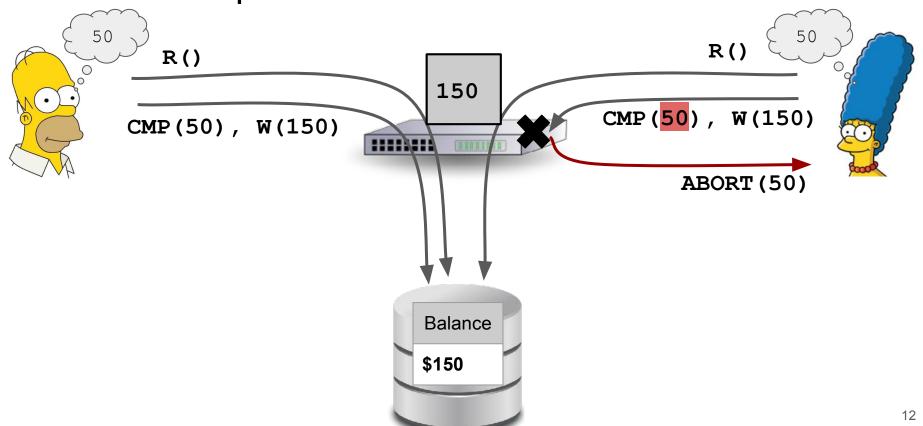


The NOCC Approach

- Update cache with write values
- Update cache with ABORT values



NOCC Example



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:

```
o 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

Reg1	Reg2	Reg3	Reg4
val[031]	val[3263]	val[6495]	val[95128]

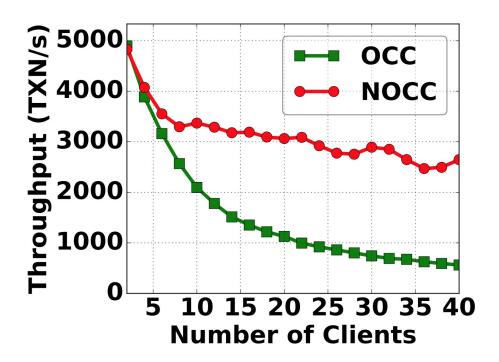
Evaluation on Hardware

Experimental Setup

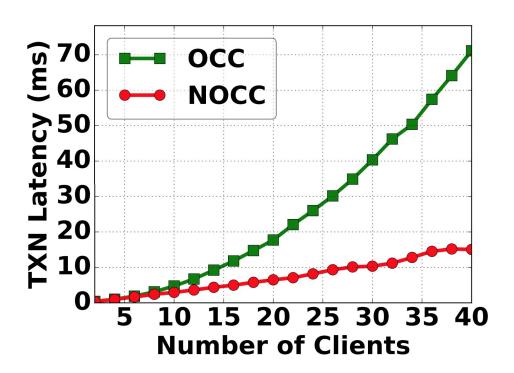
- Clients and store run on seperate servers
- Connected via a Barefoot Tofino switch running NOCC
- Evaluated with microbenchmarks and TPC-C



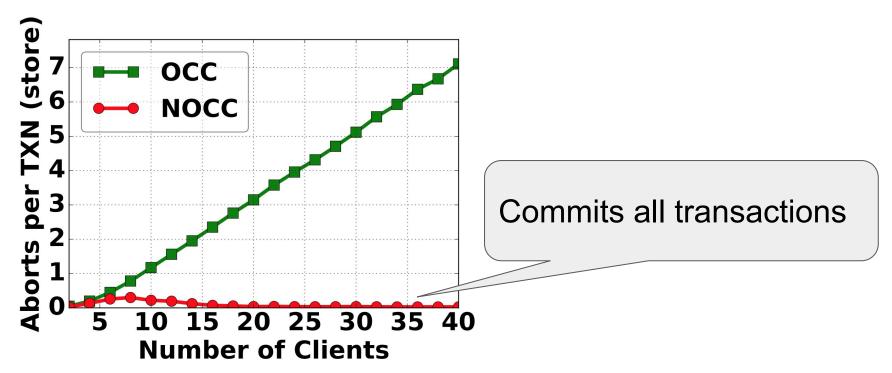
NOCC has Higher Throughput



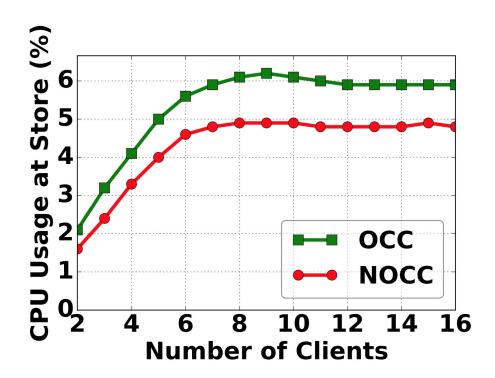
NOCC Reduces End-to-End Latency



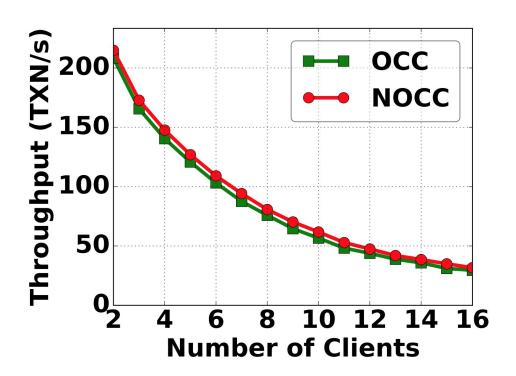
NOCC Reduces Aborts from the Store



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