Centiman : Elastic, High Performance Optimistic Concurrency Control by Watermarking

Bailu Ding, Lucja Kot, Alan Demers
Cornell University

Johannes Gehrke
Microsoft Corp.
Motivation

• Transaction processing systems face new challenges in the cloud
• Elasticity: want to scale up or down on the fly based on workload
• Modularity: need to decouple storage and processing, must be easy to deploy on commodity infrastructure
• Design Principle: simplicity and loose coupling
Outline

• Motivation
• OCC Background
• Challenges in Distributed OCC
• Sharded Validation with Watermarks
• Elasticity
• Conclusion
Optimistic Concurrency Control

- Read Phase
- Validation Phase
- Write Phase

Critical Section

Validation

T0, T1, T2, T3, T4, T5
Validation in OCC

- Compare read set with all previously committed write sets
Validation in OCC: State in the Validator

- Need to garbage collect write sets eventually
- If garbage collect too early, may need to abort due to insufficient information
Outline

• Motivation
• OCC Background
• Challenges in Distributed OCC
• Sharded Validation with Watermarks
• Elasticity
• Conclusion
Go Distributed!

Distributed Processing

Distributed Validation

Distributed Storage
Go Distributed!

Distributed Storage

Distributed Validation

Processor
Processor
Processor
Go Distributed!

Processor
Processor
Processor

Distributed Validation

Versioned KV Store
Versioned KV Store
Versioned KV Store
Versioned KV Store
Problem 1 : Reading Inconsistent Snapshots

• Updates not installed in storage atomically
• Check against inconsistent reads at validation

![Diagram showing the process of reading inconsistent snapshots](image)

- **T0**
  - Write Phase: Write X, Write Y
  - Read Phase: Read X, Read Y

- **T1**
  - Write Phase: Write X, Write Y
  - Read Phase: Read X, Read Y

- **T2**
  - Write Phase: Write X
  - Read Phase: Read X, Read Y

Partial Updates

Inconsistent Snapshot
Go Distributed!

Processor
Processor
Processor

Distributed Validation

Versioned KV Store
Versioned KV Store
Versioned KV Store
Versioned KV Store
Go Distributed!

- Processor
- Processor
- Processor

- Validator
- Validator
- Validator

- Versioned KV Store
- Versioned KV Store
- Versioned KV Store
- Versioned KV Store
Problem 2: Divergent Decisions

T3 R(...), W(X), W(Y)

Validator A
X

No conflict!

Abort!

Validator B
Y

Conflict: abort!

T4 R(X), W(X)

 Validator A
X

Thinks T3 committed, detects conflict between T3 and T4

Spurious Abort Due to Spurious Update!
Eliminate Spurious Aborts: Proactive

- Processor informs validators about the final decision
- Synchronous: slow down the system
- Asynchronous: additional complexity
  - Revoke updates, handle loss of messages

T3 R(...), W(X), W(Y)

Validator A
X

No conflict!

Validator B
Y

Conflict: abort!

Distributed Processing

Validation Request
Final Decision
Response

Distributed Storage

Distributed Validation
Eliminate Spurious Aborts: Reactive

• Spurious updates are garbage collected after a while
  • Loose coordination between processors and validators
• Tradeoff: aborts due to insufficient info vs. spurious aborts
• Our approach: watermarks

Validator A
X

Validator B
Y

Distributed Storage

Distributed Processing

T3 R(...), W(X), W(Y)

No conflict!

Conflict: abort!

Validation Request
w/ Watermarks

Response

Distributed Validation
Outline

• Motivation
• OCC Background
• Challenges in Distributed OCC
  • Sharded Validation with Watermarks
• Elasticity
• Conclusion
Watermark Abstraction

- Transactions are validated in timestamp order
- The watermark is of the same type as the timestamp
- System associates a read on a record with a watermark
- Guarantee: all updates made by transactions to the record before the watermark have been reflected in the read

```
Version   Watermark
T20       15
Read X
```

```
T19
T10
T8
T3

Transactions Updating X
```

Watermark 15
Watermarks Reduce Spurious Aborts

• Spurious updates aged out when watermark advances beyond the updating transactions

• Lazy and flexible truncation of history
Implement Watermarks

Transaction
Completion Watermark Processor
Completion Watermark KV Pair

Transaction
Completion Watermark Processor
Completion Watermark KV Pair

Transaction
Completion Watermark Processor
Completion Watermark KV Pair

Storage
Completion Watermark KV Pair
Completion Watermark KV Pair
Completion Watermark KV Pair

Read Watermark: Min of Processor Completion Watermarks
Go Distributed!  →  Centiman Architecture

- Processor
- Processor
- Processor

- Sharded Validator with Watermark
- Sharded Validator with Watermark
- Sharded Validator with Watermark

- Versioned KV Store
- Versioned KV Store
- Versioned KV Store
- Versioned KV Store
Experiment Settings for TPC-C

• Each machine: 20 EC2 compute units, 7GB RAM
• 50 processor and 50 storage instances
Experiment Result on TPC-C

![Graph showing the relationship between number of validators and throughput in thousands.](image)

- **X-axis:** Number of Validators
- **Y-axis:** Throughput (in thousands)

The graph illustrates a linear increase in throughput as the number of validators increases.
Watermarks Optimize Read-Only Transactions

![Graph showing throughput vs. percent of updating transactions. The x-axis represents percent of updating transactions ranging from 10% to 90%, and the y-axis represents throughput in thousands. Three lines are shown: one for Raw Storage, one for Centiman w/o Read-only Opt, and one for Centiman w/ Read-only Opt. The graph illustrates that as the percent of updating transactions increases, the throughput decreases for all three categories, with Centiman w/ Read-only Opt showing the least decrease.]
Experiment Result on TATP

![Graph showing experiment results on TATP]

- Throughput in Millions
- Number of Validators
- Centiman w/o Read-only Opt
- Centiman w/ Read-only Opt
Outline

• Motivation
• OCC Background
• Challenges in Distributed OCC
• Sharded Validation with Watermarks
  • Elasticity
• Conclusion
Elasticity

Processor
Processor
Processor

Sharded Validator with Watermark
Sharded Validator with Watermark
Sharded Validator with Watermark

Versioned KV Store
Versioned KV Store
Versioned KV Store
Versioned KV Store
Elasticity of the Validators

Before Migration

Processor

Validator

During Migration

Processor

Odd (Old) Validator

Even (New) Validator

After Migration

Processor

Odd Validator

Even Validator
Elastic Validation: 3 Validators to 4 Validators

Migration

Unknown Rate (%)
At New Validator

Abort Rate (%)
Conclusions

• Centiman is a decoupled and elastic distributed transaction processing system based on optimistic concurrency control
• Watermark abstraction reduces spurious aborts, optimizes read-only transactions, enables elastic validation
• Extensive experiments in the paper: high performance
Question?