Taking Omid to the Clouds: Fast, Scalable Transactions for Real-Time Cloud Analytics

O. Shacham • Y. Gottesman • A. Bergman • E. Bortnikov • E. Hillel • I. Keidar
Yahoo Research
Agenda

Introduction to Omid

Omid low latency optimizations

Apache Phoenix integration
Agenda

Introduction to Omid

Omid low latency optimizations

Apache Phoenix integration
Omid (Hope in Persian)

**Transactional API** over **NoSQL** key value

Client Library + Runtime Service

**Snapshot Isolation** consistency

Open source **Apache incubator**
Transactions and Snapshot Isolation

Aborts only \textbf{write-write} conflict
Omid Architecture - Write

Client

API

Transaction Manager

Put(k1,v1,tsr)

Commit(k1,k2...)

Data store

Commit table

Conflict Detection

Timestamp - tsr

Timestamp - tsc
Omid Architecture – Read

Client -> API

Begin

Timestamp – tsr'

Get(k1,t<=tsr')

Get tsc

Data store

Data store

Data store

Commit table

tsrr | tsc

K1 | V1 | tsr
Omid Bottleneck

- Omid preferred throughput over latency – batch
- Stopped Omid from scaling
- High latency
Taking Omid to the Cloud

Introduction to Omid

Omid low latency optimizations

Apache Phoenix integration
Omid Low Latency

Distribute commit table writes
  • Remove Omid bottleneck

Fast path transactions
  • New API for single key transactions
Distribute Commit Table Updates

Not so trivial!
SI Violation Example

Client 2 API → Data store
- K1 V1 tsr

Client 1 API
- Data store → Put
- Get(k2)

Transaction Manager
- Begin(k1,k2...)
- Timestamp - tsr
- Commit(k1,k2...)
- Timestamp - tsc

Data store
- K2 V2 tsr

Commit table
- tsr tsc
SI Violation Solution - Invalidation

Transaction Manager

Timestamp - ts

Commit(k1,k2...)

Invalidate tsr

Get(k1)

Client 2 API

Client 1 API

Data store

Data store

K1 | V1 | tsr

K2 | V2 | tsr

Data store

tsr | INVALID
Commit table
Fast Path Transactions

Many workloads have single key transactions

Wasteful access to TM for timestamps

New API – Only access data table without TM

- brc(key)
- bwc(key, val)
- br(key) + wc(key, val)
Fast Path Transactions

Client

API

Transaction Manager

Global Clock

Local Clock

Data store

Local Clock

Data store

Commit table

bwc(k1,v1)

K1 V1 tsr tsc
SI Requires Local Validation

Client 2 API

Client 1 API

Transaction Manager

Global Clock

Data store

Conflicts Detection

Local Clock

Put

bwc(k1,v1)

Data store

Commit table

Begin

Timestamp - tsr
Evaluation

HBase cluster

YCSB

Transaction sizes 1-10
Throughput Latency

Transaction Size = 1

Transaction Size = 10
Latency Breakdown

Single key transactions

- **Begin Time**
- **Data Time**
- **Commit Time**
Agenda

Introduction to Omid

Omid low latency optimizations

Apache Phoenix integration
Apache Phoenix

SQL interface over HBase
Transforms SQL queries into native HBase API calls
Omid Integration

Support Phoenix coprocessors
  • Integrate Omid’s code within HBase coprocessors

Add functionality
  • Construction of secondary indexes
  • Snapshot isolation exclude current - SIX
Secondary Index Creation

Phoenix on the fly index creation
- Secondary index from non empty table
- During creation table get updated

New fence API
- Abort transaction that overlap fence
BEGIN;

INSERT INTO T
    SELECT ID+10 FROM T;

COMMIT;
Checkpoints

BEGIN;

INSERT INTO T
  SELECT ID+10 FROM T;

COMMIT;
Summary

Omid is a mature transactional layer over HBase

Omid low latency improves throughput latency and scalability

Integrated into Phoenix with new features