Omid, Reloaded
Scalable and Highly-Available Transaction Processing

Ohad Shacham
Francisco Perez Sorrosal
Edward Bortnikov
Eshcar Hillel
Idit Keidar
Ivan Kelly
Matthieu Morel
Sameer Paranjpye
Dynamic indexing at Yahoo!

Crawl feed → Document store → Indexing Pipeline → Real-Time Search Index

Content Processing

Serving
Web indexing pipeline

- Crawl
- Docproc
- Link Analysis
- Queue
- STORM
- HBase
- Crawl schedule
- Content
- Links
- Queue
Omid – transactions for key-value store

Transactional API over HBase
Snapshot Isolation Consistency
Scalable, Highly Available
Database-neutral

Apache incubator project
Transactions and snapshot isolation

Aborts only on **write-write conflicts**
Architecture

- Client
  - Begin/Commit with Transaction Manager
  - Read/Write with Data stores

- Transaction Manager
  - Get commit info from Client
  - Atomic commit with Commit table

- Data stores
  - Read/Write

- Commit table
  - Atomic commit

- Conflict Detection
Running example

$t_r = t_1$

Write ($k_1, v_1, t_1$)
Write ($k_2, v_2, t_1$)
Read ($k', t' < t_1$)
Running example

\[ t_r = t_1 \]
\[ t_c = t_2 \]

Commit: \( t_1, \{k_1, k_2\} \)

Write \((t_1, t_2)\)
Running example

Transaction Manager

Client

Data store (k1, v1, t1)
Data store (k2, v2, t1)
Data store

Commit table (t1, t2)

Read (k1, t3)

Read (t1)

tr = t3

Running example
Post commit

$t_r = t_1$
$t_c = t_2$

Update commit cells

Client

Transaction Manager

Data store
$(k_1, v_1, t_1, t_2)$

Data store
$(k_2, v_2, t_1, t_2)$

Data store

Commit table
$(t_1, t_2)$
Commit cells

\[ t_r = t_3 \]

Read \((k_1, t_3)\)

Client

Transaction Manager

Data store \((k_1, v_1, t_1, t_2)\)

Data store \((k_2, v_2, t_1, t_2)\)

Data store

Commit table
High availability

Client
- Begin/Commit/Abort
- Results/Timestamp
- Read/Write

Transaction Manager
- Get commit info
- Atomic commit

Data store
- Single point of failure

Commit table
Architecture with HA
Split brain
Our HA solution

- Client
- Transaction Manager
- Recovery state
- Data store
- Commit table
- Infrequent access

ABORT
High availability

- **No runtime overhead** in mainstream execution
  - Minor overhead after failover

- **Leases** for leader election
  - **Local** lease status check before/after writing to Commit Table
Integration with Apache Phoenix

- OLTP on-top-of HBase
- Integrating Omid as Phoenix transaction processing engine
- Augment semantics to support secondary index creation
- Augment Omid for concurrent transaction execution
Summary

Transactions – important use case in stream processing

Snapshot Isolation – scalable consistency model

Omid – an apache incubator TP system for HBase
  Battle-tested, Web-scale, HA