MDCC
Multi Data Center Consistency

Tim Kraska, Gene Pang, Michael Franklin, Samuel Madden, Alan Fekete
g pang@cs.berkeley.edu
MOTIVATION
June 29, 2009: Rackspace power outage of “approximately 40 minutes”
June 29, 2009:
Rackspace power outage of “approximately 40 minutes”

April 21, 2011:
AWS East outage of over 2 hours, with data loss
June 29, 2009: Rackspace power outage of “approximately 40 minutes”

April 21, 2011: AWS East outage of over 2 hours, with data loss

UNRELIABLE DATA CENTERS
SOLUTION: GEO-REPLICATION
HIGH NETWORK LATENCY
THE ENVIRONMENT

UNRELIABLE DATA CENTERS + HIGH NETWORK LATENCY
Our Challenge

Fast and Reliable Database

Unreliable Data Centers + High Network Latency
MDCC

**MULTI DATA CENTER CONSISTENCY**

new distributed commit protocol

**STRONG CONSISTENCY ACROSS DATA CENTERS**

**LOW LATENCY**

**NO DATA LOSS**
## Existing Solutions

<table>
<thead>
<tr>
<th>Service</th>
<th>Consistency</th>
<th>Transactional Unit</th>
<th>Commit Latency</th>
<th>Data Loss Possible?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon Dynamo</td>
<td>Eventual</td>
<td>None</td>
<td>1 round trip</td>
<td>Not possible</td>
</tr>
<tr>
<td>Yahoo PNUTS</td>
<td>Timeline per key</td>
<td>Single key</td>
<td>1 round trip</td>
<td>Possible</td>
</tr>
<tr>
<td>COPS</td>
<td>Causality</td>
<td>Multi-record</td>
<td>1 round trip</td>
<td>Possible</td>
</tr>
<tr>
<td>MySQL (async)</td>
<td>Serializable</td>
<td>Static partition</td>
<td>1 round trip</td>
<td>Possible</td>
</tr>
<tr>
<td>Google Megastore</td>
<td>Serializable</td>
<td>Static partition</td>
<td>2 round trips</td>
<td>Not possible</td>
</tr>
<tr>
<td>Google Spanner</td>
<td>Snapshot Isolation</td>
<td>Partition</td>
<td>2 round trips</td>
<td>Not possible</td>
</tr>
<tr>
<td>Walter</td>
<td>Parallel Snapshot Isolation</td>
<td>Multi-record</td>
<td>2 round trips</td>
<td>Not possible</td>
</tr>
</tbody>
</table>
## Existing Solutions + MDCC

<table>
<thead>
<tr>
<th>Service</th>
<th>Consistency</th>
<th>Transactional Unit</th>
<th>Commit Latency</th>
<th>Data Loss Possible?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon Dynamo</td>
<td>Eventual</td>
<td>None</td>
<td>1 round trip</td>
<td>Not possible</td>
</tr>
<tr>
<td>Yahoo PNUTS</td>
<td>Timeline per key</td>
<td>Single key</td>
<td>1 round trip</td>
<td>Possible</td>
</tr>
<tr>
<td>COPS</td>
<td>Causality</td>
<td>Multi-record</td>
<td>1 round trip</td>
<td>Possible</td>
</tr>
<tr>
<td>MySQL (async)</td>
<td>Serializable</td>
<td>Static partition</td>
<td>1 round trip</td>
<td>Possible</td>
</tr>
<tr>
<td>Google Megastore</td>
<td>Serializable</td>
<td>Static partition</td>
<td>2 round trips</td>
<td>Not possible</td>
</tr>
<tr>
<td>Google Spanner</td>
<td>Snapshot Isolation</td>
<td>Partition</td>
<td>2 round trips</td>
<td>Not possible</td>
</tr>
<tr>
<td>Walter</td>
<td>Parallel Snapshot Isolation</td>
<td>Multi-record</td>
<td>2 round trips</td>
<td>Not possible</td>
</tr>
<tr>
<td>MDCC</td>
<td>Read-committed without lost updates</td>
<td>Multi-record</td>
<td>1* round trip</td>
<td>Not possible</td>
</tr>
</tbody>
</table>
WHY MDCC WORKS

OPTIMIZES FOR TWO KEY OBSERVATIONS ON WORKLOADS
Why MDCC Works

Conflicts are rare

Everyone updates their own data
CONFLICTS ARE RARE
everyone updates their own data

CONFLICTS COMMUTE
order of concurrent updates isn’t important

Why MDCC Works
MDCC Outline

Two-Phase Commit: Background

MDCC Distributed Transactions

Optimization: Conflicts are Rare

Optimization: Conflicts Commute

Evaluation
TWO-PHASE COMMIT: BACKGROUND
TWO-PHASE COMMIT (2PC)

STANDARD METHOD FOR DISTRIBUTED TRANSACTIONS

PREPARE PHASE + COMMIT PHASE

SYSTEMS TYPICALLY USE 2PC BETWEEN PARTITIONS
AN EXAMPLE TRANSACTION

buy 1  book-A (4 in stock)

buy 3  book-B (9 in stock)
TWO-PHASE COMMIT

coordinator

node
node
node
node
book-A
4
4
4
9

node
node
node
node
book-B
4
4
9
9

9
Two-Phase Commit

Coordinator initiates transaction commit by sending PREPARE to all nodes
TWO-PHASE COMMIT

nodes vote “yes” or “no” for the commit
**Two-Phase Commit**

Coordinator makes final decision (commit/abort)

Can commit only when **ALL** votes are “yes”
**Two-Phase Commit**

- All nodes must respond for commit.
- Coordinator makes final decision (commit/abort).
- Can commit only when **ALL** votes are “yes”.

Diagram showing coordinator, nodes, and books.
TWO-PHASE COMMIT

- coordinator makes final decision (commit/abort)
- can commit only when **ALL** votes are “yes”
Two-Phase Commit

Coordinator stores final decision.
TWO-PHASE COMMIT

transaction committed ✔

decision stored only coordinator

coordinator stores final decision

node 4
node 4
node 4
node 9
node 9
node 9
book-A
book-B
TWO-PHASE COMMIT

coordination stores final decision

transaction committed

coordinator failure could block protocol
TWO-PHASE COMMIT

coordinator informs nodes with COMMIT phase

node
node
node
node
node
node

book-A
book-B
MDCC DISTRIBUTED TRANSACTIONS
MDCC TRANSACTIONS

coordinator

book-A

4

node

node

node

book-B

9

node

node

node
MDCC Transactions

separate Paxos instance per record
each executes Paxos replicated state machine
coordinator proposes **options** (potential updates)
MDCC TRANSACTIONS

nodes validate the options
check with all permutations of pending **options**
check: write-write conflicts, integrity constraints
nodes tag **options** as “accepted” or “rejected”
Paxos learners learn the result
MDCC Transactions

transaction committed

leader-A

leader-B

node

node

node

node

node

book-A

book-B

commit when all options are learned
MDCC TRANSACTIONS

transaction committed

leader-A

node node node

4 3 ✔

node

4 3 ✔ book-A

node

9 6 ✔ book-B

node

node

9 6 ✔

leader-B

node

9 6 ✔

node

9 6 ✔

coordinator cannot change transaction outcome
Once learned, **options** are never unlearned.
MDCC Transactions

notify nodes to execute **option**, to make visible
**MDCC Transactions**

- **Node**: book-A
- **Visibility**: 4 3 ✔️
- **Node**: book-B
- **Visibility**: 9 6 ✔️
- **Option visibility can bypass the coordinator**
**options** enable read-committed isolation
# MDCC vs. Two-Phase Commit

<table>
<thead>
<tr>
<th>MDCC</th>
<th>TWO-PHASE COMMIT</th>
</tr>
</thead>
<tbody>
<tr>
<td>makes distributed commit decision</td>
<td>makes commit decision on coordinator</td>
</tr>
<tr>
<td>stores distributed commit decision</td>
<td>stores commit decision on coordinator</td>
</tr>
<tr>
<td>can tolerate node failures/delays</td>
<td>node failures/delays can block protocol</td>
</tr>
</tbody>
</table>
OPTIMIZATION: CONFLICTS ARE RARE
MDCC: Fast Paxos

First to use Fast Paxos for transactions

Possible to bypass the leader

Can reduce number of message round-trips, when conflicts are rare
Classical Paxos:
Clients propose through the leader.

Fast Paxos:
Clients propose directly to nodes.
THE “FAST” IN FAST PAXOS

BYPASS THE LEADER

CONSSENSUS IN 1 ROUND-TRIP
THE "SLOW" IN FAST PAXOS

CONCURRENT UPDATES CAN PREVENT CONSENSUS

LEADER MUST STEP IN TO RESOLVE CONFLICTS

2 ADDITIONAL MESSAGE ROUNDS TO RESOLVE

CONSENSUS MAY TAKE 3 ROUND-TRIPS
LARGER QUORUMS IN FAST PAXOS

CLASSIC PAXOS

\[
\left\lceil \frac{N}{2} \right\rceil
\]

responses needed

FAST PAXOS

\[
\left\lceil \frac{2N}{3} \right\rceil + 1
\]

responses needed
MDCC: Fast Paxos Implications

Transactions can bypass the leaders

Transactions can commit in 1 round-trip, when conflicts are rare
OPTIMIZATION: CONFLICTS COMMUTE
Commutative Updates

commutative updates do not need to be totally ordered
**Commutative Updates**

Commutative updates do not need to be totally ordered.
Commutative Updates

commutative updates do not need to be totally ordered

-3  -1  -4

=  

-1  -3  -4
Commutative Updates

Commutative updates do not need to be totally ordered

\[
\begin{array}{ccc}
-3 & -1 & -4 \\
-1 & -3 & -4 \\
-4 & -3 & -1 \\
\end{array}
\]
MDCC: GENERALIZED PAXOS

GENERALIZATION OF FAST PAXOS

INSTANCES AGREE ON COMMAND STRUCTURES (C-STRUCTS)

C-STRUCTS ARE SEQUENCES OF COMMANDS, NOT JUST A SINGLE COMMAND
MDCC: Commutative Updates

perfect for c-structs of Generalized Paxos

can use Fast Paxos (fast commits) without worrying about the order!

however, order matters with integrity constraints
EXAMPLE OF COMMUTATIVE UPDATES

online bookstore: 4 books in stock

integrity constraint: stock ≥ 0

5 replicas

Fast Paxos quorum size of 4
COMMUTATIVE UPDATES

node

node

node

node

node
COMMUTATIVE UPDATES
COMMUTATIVE UPDATES

1 book sold!
1 book sold!
2 books sold!
2 books sold!
COMMUTATIVE UPDATES

3 books sold!
COMMUTATIVE UPDATES

3 books sold!

node

node

node

node
COMMUTATIVE UPDATES

4 books sold!

node

node

node

node
COMMUTATIVE UPDATES

4 books sold!
COMMUTATIVE UPDATES

5 books sold!

node
node
node
node
node
node
COMMUTATIVE UPDATES

5 books sold! VIOLATED INTEGRITY CONSTRAINTS!
**Solution: Quorum Demarcation**

adjust the constraint \((stock \geq 0)\) to keep replicas consistent without coordination

 commits require quorums, so coordination can piggyback on quorum messages

quorum demarcation: \(stock \geq \frac{N - Q}{N} \text{ value}\)
$stock \geq \frac{N - Q}{N}$ value $= 0.8$
3 books sold!

\[ stock \geq \frac{N - Q}{N} \Rightarrow value = 0.8 \]
COMMUTATIVE UPDATES

3 books sold!

\[ stock \geq \frac{N - Q}{N} \text{ value } = 0.8 \]
MDCC: Commutative Updates

Fast commits even when conflicts occur, when order is not important

Maintain consistency and fast commits, with quorum demarcation
EVALUATION
MDCC EXPERIMENTAL SETUP

implemented on SCADS, a distributed key-value store

fully replicated to 5 Amazon EC2 regions (California, Virginia, Ireland, Singapore, Tokyo)

TPC-W benchmark for general performance
micro benchmark for protocol investigation
TPC-W BENCHMARK
WRITE RESPONSE TIME CDF

Percentage of Transactions

Write Transaction Response Times, log-scale (ms)

QW-3
QW-4
MDCC
2PC
Megastore*
MICRO BENCHMARK
## Various MDCC Configurations

<table>
<thead>
<tr>
<th></th>
<th>Supports Multi-Paxos</th>
<th>Supports Fast Paxos</th>
<th>Optimizes Commutative Updates</th>
</tr>
</thead>
<tbody>
<tr>
<td>MDCC</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Fast</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Multi</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
</tbody>
</table>

85
WRITE RESPONSE TIME CDF

Percentage of Transactions

Write Transaction Response Times (ms)

MDCC
Fast
Multi
2PC
**Varying Conflict Rate**

Commits/Aborts (in thousands)

- Hotspot Size
  - 90%
  - 50%
  - 20%
  - 10%
  - 5%
  - 2%

- 2PC Multi
- Fast Multi
- MDCC
MDCC - NEW COMMIT PROTOCOL

CONFLICTS ARE RARE -OR- CONFLICTS COMMUTE

STRONG CONSISTENCY ACROSS DATA CENTERS

1* ROUND-TRIP DURABLE COMMITS

COMPARABLE PERFORMANCE TO EVENTUALLY CONSISTENT PROTOCOLS

THANK YOU!

GENE PANG - GPANG@CS.BERKELEY.EDU