MDCC:
Multi-data Center Consistency

Authors: Tim Kraska, Gene Pang, Michael J. Franklin, Samuel Madden, Alan Fekete
Presenter: Eliran Shema
Outline

- Introduction
- Architecture
- The MDCC Protocol
- Guarantees
- Evaluation
Outline

• Introduction
• Architecture
• The MDCC Protocol
• Guarantees
• Evaluation
Introduction

• Why multi-data center?
  – Growing capacity over time
  – Providing global reach with minimum latency for users
  – Maintaining performance and availability
    • Providing additional instances for resiliency
    • Providing a facility for disaster recovery
    • Providing a hot-swap standby capability
Introduction

• Few Data centers failure examples:
  – September 1, 2009 – gmail servers outage
  – August 7, 2011 - Lightning strikes cloud:
    • Amazon's Elastic Cloud Compute and Relational Database Service.
    • Microsoft’s BPOS (A service that was replaced by Office 365).
  – June 29, 2009 - Dallas-Fort Worth Data Center Power outages
MDCC (Multi-Data Center Consistency) is a new database solution which provides full transactions with strong consistency, and synchronous replication for fault-tolerant durability.
Outline

• Introduction
• Architecture
• The MDCC Protocol
• Guarantees
• Evaluation
• Stateful components dispersed as a distributed record manager.
  – Can be scaled via any standard method such as range partitioning
• Queries and transactions are provided via stateless DB library which can be deployed on any app server.
  – Can be replicated freely as it is stateless
• Unlike other systems MDCC supports individual master per record (storage node or app server).
• Allows the transaction manager to either:
  – Claim ownership of the records.
  – Ask the current master to do it (black arrows)
  – Ignore the master and update directly (red arrows)
Outline

- Introduction
- Architecture
- The MDCC Protocol
- Guarantees
- Evaluation
Paxos Background

- Classic Paxos

![Paxos Diagram](image.png)
• **Classic Paxos**

1: proposer
1-n: acceptors
1-n: acceptors

1: proposer
1-n: acceptors
1-n: acceptors

(“prepare”, 1) ➔ 1
(“ack”, 1, ⊥, ⊥) ➔ n

(“accept”, 1, v₁) ➔ 1
(“accept”, 1, v₁) ➔ 1
(“accept”, 1, v₁) ➔ 1
(“accept”, 1, v₁) ➔ 1

decide v₁
Paxos Background

- **Multi-Paxos**
  - Maintains the leader position for multiple rounds, hence removing the need for phase 1 messages
The MDCC Protocol

• MDCC Transactions
  – Multi record transactions with read committed isolation and without lost update problem.
  – Features:
    • Atomic durability
    • Detection of write write conflicts
    • Commit visibility
  – Achieved by paxos instance per record to “accept” an option for an update instead of writing the value.
  – Waiting for the app server to asynchronously commit or abort
The MDCC Protocol

• A transaction updating a record creates a new version, which is represented in the form of $v_{\text{read}} \rightarrow v_{\text{write}}$.

• We only allow one outstanding option per record, which stays invisible until the option is executed.
The app server tries to get the options accepted for all updates. Proposing the options to the Paxos instances of each record.

Depending on the $v_{\text{read}}$ value the nodes actively decide whether to accept or reject. Unlike Paxos which uses ballot number.
The MDCC Protocol

- The app-server learns an option only if a majority of storage nodes agrees on the option.
- No clients or app-servers aborts, only abort if an option is rejected.
- If the app-server determines that the transaction is aborted or committed, it informs involved storage nodes through an asynchronous learned message about the decision.
The MDCC Protocol

• So far we achieved:
  – 1 round trip commit, assuming all the masters are local. (Ignoring communication with local master).
  – 2 round trip commit when the masters aren’t local.

• ANIMATION
Avoiding Deadlocks

- Assuming T1 and T2 want to learn an option for both R1 and R2.
- T1 learns v0->v1 for R1 and T2 tries to acquire v0->v2 for R1.
- Pessimistically T1 learn is accepted and T2 learn is rejected. Leading to both transactions rejection.
The MDCC Protocol

• Failure recovery
  – Failure of a storage node is masked by the use of quorums.
  – Master failure can be recovered by reselecting a master after a timeout.
The MDCC Protocol

- App-server failure
  - All options include a unique transaction-id + all primary keys of the write-set.
  - A log of all learned options is kept at the storage node.
  - After a set timeout, any node can reconstruct the state by reading from a quorum of storage nodes for every key in the transaction.
  - Data center failure-all nodes failed.
• Fast Paxos
  – Removes the need to become the leader, allowing any node to propose the value.
  – Requires larger quorum size.
  – May reach deadlock leading to the leader running a classic Paxos instance to resolve it.
The MDCC Protocol

• Transactions Bypassing Master
  – Using fast Paxos we assume all versions start with a fast ballot number, until a master change it into classic via phase1 message.
  – Any storage node agrees to accept the first proposed value.
• **Collision recovery**
  – Fast quorum can fail, which leads to a classic ballot from the master.
  – Fast policy:
    • Assume all instances start as fast.
    • After a collision set the next X (default 100) instances as classic.
    • After X instances go back to fast again.
• **Generalized Paxos**
  – Improves fast Paxos.
  – Each round accepts a sequence of values.
The MDCC Protocol

• MDCC usage of generalized Paxos
  – Single record Paxos instances, meaning no sequence for normal operations.
  – Sequence is only available for commutative operations.

– Animation
  – Demarcation protocol: lowerLimit \( \geq \frac{N - Q}{N} \cdot V \)
    • N=#nodes, Q=Quorum size
• Introduction
• Architecture
• The MDCC Protocol
• Guarantees
• Evaluation
Guarantees

• **Read Committed Without Lost Updates**
  – We only allow to read learned options.
  – We detect all write-write conflicts so that a Lost Update option gets rejected.
  
  – Currently MS SQL server, Oracle database, IBM DB2 andpostgresql all use Read Committed by default.
Guarantees

• Staleness
  – We allow reads from any node, but the read might be stale if the node missed updates.
  – A safe read, requires reading a majority of the nodes.
  – There are methods which allow up to date reads:
    • From Megastore: psuedo master which is always part of the quorum
• **Atomic visibility**
  
  – MDCC supports atomic durability, but not visibility, this is the same for two-phase commit.
  
  – MDCC could use a read/write locking service or snapshot isolation (used in Spanner) to achieve Atomic Visibility.
Outline

• Introduction
• Architecture
• The MDCC Protocol
• Guarantees
• Evaluation
Evaluation

• Implementation of a mdcc over a key value store across 5 different geographically located datacenters using amazon EC2 cloud.
• For testing, used TPC-W, a transactional benchmark that simulates the workload experienced by an e-commerce web server.
Evaluation

• Competition:
  – Quorum write. (no isolation, atomicity, or transactional guarantee)
  – Two Phase Commit. (cannot deal with node failure)
  – Megastore* (couldn’t compare to the real one, implemented one based on the article about it)
Evaluation

• Setup:
  - 100 evenly geo replicated clients running the benchmark
  - 10,000 items in the database

Figure 3. TPC-W write transaction response times CDF

Figure 4. TPC-W transactions per second scalability
Evaluation

- MDCC compared to itself

**Figure 5.** Micro-benchmark response times CDF

**Figure 6.** Commits/aborts for varying conflict rates
Evaluation

- MDCC compared to itself

**Figure 7.** Response times for varying master locality

**Figure 8.** Time-series of response times during failure
Questions