MDCC: Multi-Data Center Consistency

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

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

Why multi-data center?
- Growing capacity over time
- Providing global reach with minimum latency
- Maintaining performance and availability
  1. Providing additional instances for resiliency
  2. Providing a facility for disaster recovery
Introduction

Few Data centres' failure examples:

- Gmail servers outage – September 1, 2009
- Amazon’s Elastic Compute and Relational Database Service - August 7, 2011
- Dallas –Fort Worth Data Center Power outrages – June 29, 2009
Introduction

What is MDCC?

- Multi-Data Center Consistency is also called MDCC
- It is a database which provides transactions with
  1. Strong consistency
  2. Synchronous replication for fault-tolerant durability
The two kind of components:

- **Stateful components**
  - They are dispersed as a distributed record manager.
  - Can be scaled via methods like range partitioning

- **Stateless component**
  - Queries and transactions fall under this category and they can be deployed in any app server
  - Can be replicated freely as it is stateless
The transaction manager can either:

- Claim ownership of the records
- Ask the current master to do it (Black arrows)
- Ignore the master and update directly (red arrows)
Paxos Background

- Classic Paxos:
Multi Paxos:

- Maintains the leader position for multiple rounds, hence removing the need for phase 1 messages:
First let us look at the animation and understand the concept:
The MDCC Protocol

About MDCC Transactions:

- Features:
  - Atomic Durability
  - Detection of write-write conflicts
  - Commit Visibility

- Uses Paxos to “accept” an option for an update instead of writing the value

- Waiting for the app server to asynchronously commit or abort
A transaction updating a record creates a new version, which is represented in the form of $V_{\text{read}} \rightarrow V_{\text{write}}$

The transaction only allows one outstanding option per record, which stays invisible until the option is executed.
The MDCC Protocol

➢ The app server tries to get the options accepted for all the updates. Proposing the options to the Paxos, instances of each record.

➢ Depending on the Vread value the nodes actively decide whether to accept or reject. Unlike Paxos which uses ballot number.
The MDCC Protocol

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

So far we have achieved:

1. 1 round trip commit, assuming all the masters are local.
2. 2 round trip commit when the masters are not local.
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 R2.
- Pessimistically T1 learn is accepted and T2 learn is rejected in the next phase.
- In a case of deadlock it leads to both transactions to reject.
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.
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.
Paxos Background

- Generalized Paxos
  - Combines fast and classic Paxos.
  - Each round accepts a sequence of values.
  - Sequence has to be identical on all acceptors.
The MDCC Protocol

Let’s look into another animation of MDCC Demarcation Protocol:

ANIMATION
MDCC usage of generalized Paxos

- Single record Paxos instances, meaning no sequence for normal operations.
- Sequence is only available for commutative operations.
Guarantees

- **Read Committed Without Lost Updates**
  - It only allows a transaction to read learned options.
  - It can detect all write-write conflicts so that a Lost Update option gets rejected.

- Currently MS SQL server, Oracle database, IBM DB2 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.
Guarantees

- 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.
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
Thank you