Data Synchronization Across Heterogeneous Systems

Sathwik IMT2022045 | Manish IMT2022047 | Srinivas IMT2022066 | Adarsha IMT2022069

I. Introduction

This project involves the design and implementation of a heterogeneous database synchronization system for student grade management across MySQL, MongoDB, and Apache Hive. The system addresses the critical challenge of maintaining data consistency when the same grade records are accessed and modified through different database paradigms. The significance of this work lies in its practical approach to resolving synchronization conflicts in real-world educational systems where institutional data often spans multiple database technologies.

The primary objectives include:

- Developing a unified interface for GET/SET operations across relational (MySQL), document-oriented (MongoDB), and distributed (Hive) databases.
- Implementing timestamp-based merge operations to synchronize databases while resolving update conflicts.
- Maintaining operation logs (oplogs) for auditability and recovery.
- Ensuring eventual consistency without compromising the unique advantages of each database system.

II. System architecture

We have designed a multi-database synchronization system with three distinct database servers (MySQL, MongoDB, and Hive), all maintaining identical student grade data. Each database stores records with the schema $student_id, course_id, grade$ and maintains an operation log (oplog) that tracks all successful updates with timestamps. These oplogs (oplog.sql.txt, oplog.mongo.txt, oplog.hive.txt) are crucial for merge operations and conflict resolution.

Key architectural components include:

- Database Adapters $mysql_db.py, mongo_db.py, hive_db.py$: Handle database-specific operations while presenting a uniform interface.
- Operation Logs: Plaintext files recording all successful SET and GET operations with timestamps.
- Merge Engine: Resolves conflicts by processing oplogs in timestamp order.
- Initialization System $create_databases.py, table.py$: Ensures consistent starting states across all databases.

The system processes operations through a central driver program (driver.py) that:

- Reads test cases containing GET/SET/MERGE instructions.
- Routes each operation to the appropriate database adapter.
- Maintains a global timestamp to ensure chronological ordering.
- Handles merge operations by comparing oplogs across databases.

Unlike traditional client-server architectures, our system implements a unified interface that processes batch operations from test case files (testcase1.in), supports all combinations of database merges (HiveMySQL, MongoDBHive, etc.) and preserves each database's native advantages while ensuring consistency.

The synchronization mechanism uses state-based reconciliation where:

- Each database maintains its current state through regular GET/SET operations
- Merge operations compare oplog timestamps to determine the most recent update
- Conflicting updates are resolved by always selecting the operation with the latest timestamp
- All databases eventually converge to the same state after merge completion

III. Methodology and Implementation

We have implemented three core operations - GET, SET, and MERGE - for grade management across heterogeneous databases. The system processes operations through test case files containing timestamped instructions, ensuring ordered execution across all databases.

GET Operation: When processing a GET request $e.g., 1, SQL.GET(SID1033, CSE016)$:

- Extracts studentid and courseid from arguments
- Routes to the appropriate database adapter $mysql_db.py/mongo_db.py/hive_db.py$
- Executes database-specific query: MySQL: SQL SELECT query, MongoDB: Document lookup, Hive: HiveQL execution.

- Logs successful operations to the respective oplog $oplog.sql.txt/.mongo.txt/.hive.txt$.

SET Operation: For processing the SET requests. $e.g., 2, HIVE.SET((SID1310, CSE020), B)$:

- Validates the existence of the $student_id, course_id$ pair.
- Compares new grade with current value (skipping if unchanged).
- Performs database-specific update: MySQL: SQL UPDATE, MongoDB: Document update, Hive: Hive INSERT OVERWRITE.
- Records successful updates in oplogs with timestamps

MERGE Operation: For processing the MERGE requests. e.g., HIVE.MERGE(SQL):

- Identifies source and target databases from instruction.
- Parses both databases' into respective oplogs.
- For each SET operation in source oplog: If target lacks this operation: adds it, If both have it: keeps the version with latest timestamp.
- Applies selected operations to target database.
- Updates target's oplog with merged operations.
- Increments global timestamp for new operations.

IV. Functionality

Operation log:.

(a) Structure (per database):

- Plaintext files $oplog.sql.txt, oplog.mongo.txt, oplog.hive$.
- Each line format: timestamp, OPERATION(args).
- Example line: 16, SET((SID1033,CSE016),B)

(b) Sample Logs and Test Cases :

- Test Case (from testcase1.in):
  1, HIVE.SET((SID1033,CSE016),A)
  4, SQL.SET((SID1033,CSE016),B)
  HIVE.MERGE(SQL)

- Resulting Oplogs:
  oplog.hive.txt:
  1, SET((SID1033,CSE016),A)

  oplog.sql.txt:
  4, SET((SID1033,CSE016),B)

Merge:.

(a) Merge Excecution flow:

- Reads oplog.hive.txt and oplog.sql.txt
- Detects conflict on (SID1033,CSE016)
- SQL's timestamp (4) greater than Hive's timestamp (1)
- Applies SQL's grade (B) to Hive
- Records merged operation in Hive's oplog with new timestamp (4)

(b) Merge Operation Properties:

- **Associative**: Associativity is guaranteed through timestamp-based conflict resolution where merge operations always select the highest-timestamped value regardless of merge order, ensuring (A u B) u C = A u (B u C) as all sequences converge to the same final state containing the latest updates.
- **Commutative**: A.merge(B) yields same final state as B.merge(A). Guaranteed by timestamp comparison.
- **Idempotent**: Repeated merges between the same databases produce no new changes.
- **Eventually Consistent**: All databases converge to identical state after full merge cycle.
  Example: Merging HiveMySQLMongoDB in any order
- **Preservation of Intent**: Always respects the most recent explicit update
- **Partial Failure Resilience**: Interrupted merges can be restarted (oplogs are append-only).Failed operations don't appear in oplogs.

(c) Merge Logic for conflict resolution: The merge operation follows a timestamp-based last-writer-wins strategy:

- Compares timestamps for the same (studentid, courseid) across oplogs.
- Always selects the operation with the highest timestamp.
- Propagates changes to target database and updates its oplog.

Reasons for choosing daatbases:.
Hive:

- **Scalability**: Hive is built on top of Hadoop, providing scalability by distributing data across multiple nodes in a cluster.
- Integration with Hadoop Ecosystem: Hive seamlessly integrates with Hadoop components such as HDFS and YARN, offering a cohesive big data processing environment.

MongoDB:

- **Flexible Schema Design:** MongoDB's document-oriented structure allows for flexible schema design, suitable for storing triples with varying structures.
- **High Performance:** MongoDB is optimized for high performance with features like indexing and sharding, enhancing query performance and scalability.
- **Horizontal Scalability:** MongoDB supports horizontal scalability through sharding, facilitating the distribution of data across multiple nodes.

MySQL:

- **Reliable Transactions:** Handles updates safely - if an error occurs, changes won't be saved half-way.
- **Fast Performance:** Optimized to quickly find and update student grades, even in large classes.

- **Easy to Use:** Works everywhere with lots of online help available.

V. Evaluation and Results

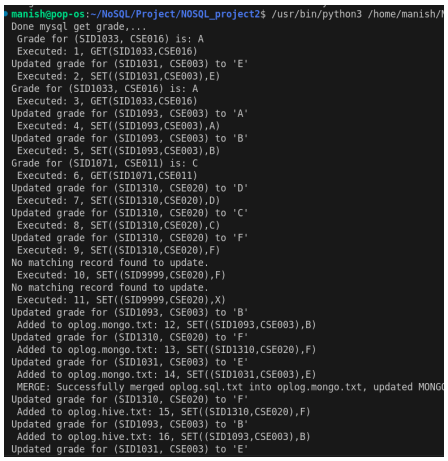

Fig. 1. Output of the testcase file (fig 2)

```
1 , SQL . GET ( SID1033 , CSE016 )
2 , SQL . SET ( (SID1031 , CSE003) , E )
1 , HIVE . GET ( SID1033 , CSE016 )
1 , MONGO . SET (( SID1093 , CSE003 ) , A )
3 , SQL . SET (( SID1093 , CSE003 ) , B )
2 , MONGO . GET ( SID1071 , CSE011 )
2 , HIVE . SET (( SID1310 , CSE020) , D )
3 , MONGO . SET((SID1310 , CSE020) , C)
4 , SQL . SET((SID1310 , CSE020) , F)
5 , SQL . SET((SID9999 , CSE020) , F)
4 , MONGO . SET((SID9999 , CSE020) , X)
MONGO . MERGE( SQL )
HIVE . MERGE( MONGO )
```

Fig. 2. Testcase file

The above screenshots are the 2 testcase files. In each file there are series of set and get commands where the first number in the beginning represents the count of the operation in each specific database.We are treating the lines themselves as the global timestamp.

VI. Conclusion

In conclusion, our project successfully implemented a heterogeneous database synchronization system, demonstrating seamless coordination across MySQL, MongoDB, and Hive using timestamp-based conflict resolution. Key achievements include:

- A unified interface for GET/SET operations and merges, ensuring consistency despite differing database paradigms.
- Oplog-driven synchronization, where operation logs with global timestamps guaranteed eventual consistency.
- Modular adapters for each database, enabling scalable integration while preserving their native strengths (ACID compliance, flexibility, and distributed processing).

Challenges faced during making of project:

- Implementing ACID transactions in HIVE was tough.
- while merging two databases lets say A and B (A.merge(B)) we have to append the set operations of B into oplog of A while merging we should not take the global timestamp as it wont work. Instead we have to take the time at which user wrote it.
- Test-case validation of edge cases to verify merge correctness.

VII. Individual Contributions

- **Sathwik**: 25% of total work —
- **Manish**: 25% of total work —
- **Srinivas**: 25% of total work —
- **Adarsha**: 25% of total work —