Lab 4 Proposal - Distributed Database Platform

Team: Two Generals. Members: Johnny Chang, Ming Xin

Design

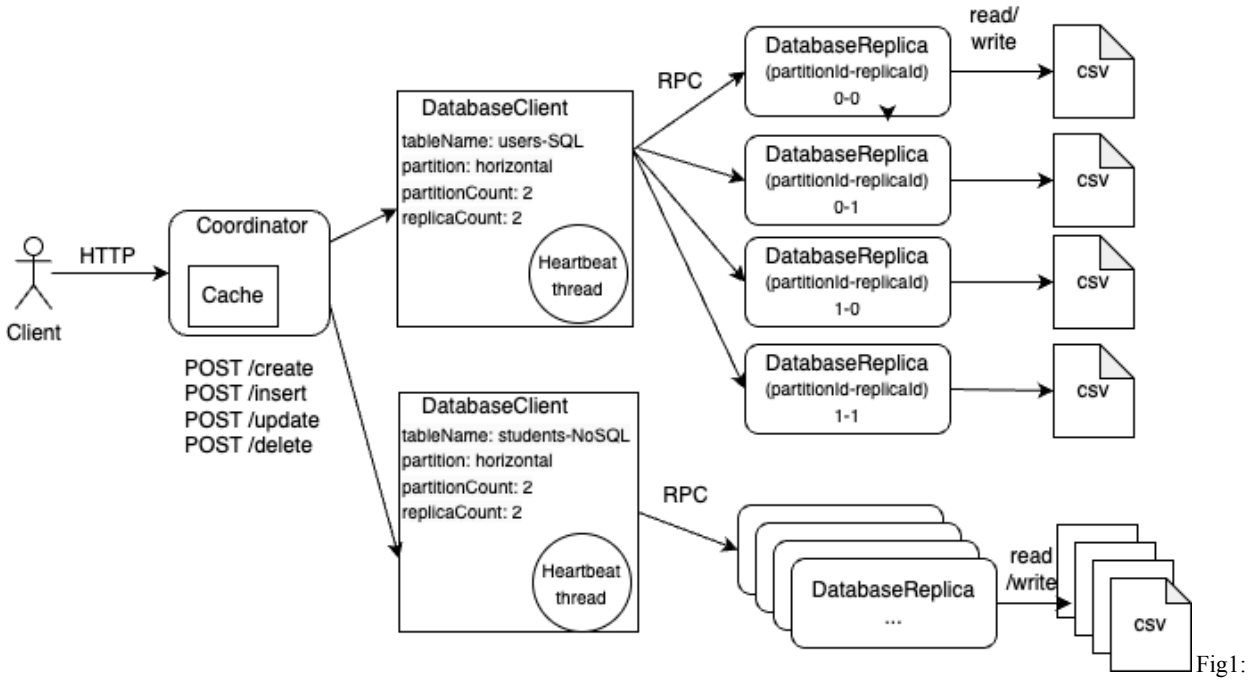

Fig1: Architecture

Overall

Clients interact with our distributed database platform through the HTTP interface of the Coordinator. Upon receiving a table configuration (which includes table name, database type (SQL or NoSQL), partition type, partition count, and replica count), the Coordinator initializes DatabaseClient and DatabaseReplica instances. These instances communicate via the RPC protocol. The DatabaseReplica handles file system read and write operations in CSV format. Please refer to Appendix 1 for HTTP API documentation.

Query Languages

The system supports two types of query languages:

SQL: Standard SQL syntax.

NoSQL: A custom query language defined as follows:

- Create Table: CREATE TABLE tableName
- Select: SELECT tableName
- Insert: INSERT tableName id, idValue(int), key1, value1, key2, value2…

- Update: UPDATE tableName key1, value1, key2, value2… WHERE id idValue(int)

There are three limitations to our query languages:

1. Only supports SELECT *.This means it is not possible to select specific columns or use conditions to filter records directly in the query.
2. UPDATE and DELETE only support a single WHERE clause, for example, WHERE a = 1 AND b = 2 will not work.
3. During vertical partitioning, UPDATE operations are restricted such that only columns within the same partition as the WHERE clause can be updated.
4. INSERT must include ID as the first column for SQL, and first key for NoSQL.

Partitioning

The system offers support for both vertical and horizontal partitioning:

- Vertical Partitioning: Available only for SQL queries.
- Horizontal Partitioning: Supported for both SQL and NoSQL queries.

Partitioning is static, once a database is configured, the partition count cannot be altered.

Consistency

To maintain system consistency, a heartbeat thread in the DatabaseClient periodically checks the status of the replica. In scenarios where a replica is down, the system shifts to a read-only mode where only read operations are permitted until all replicas are operational again.

Caching

In-memory cache is implemented at Coordinator, caching the result of SELECT request.

Concurrency

The system is able to handle concurrent requests.

Technical Approach

- Programming Language: Java
- RPC framework: Java RMI
- SQL Parsing: JSqlParser (https://github.com/JSQLParser/JSqlParser)

Test cases

We have developed 11 test cases for our distributed database system, with a total grading allocation of 180 points. Below is a detailed breakdown of each test case with grades assigned in the parentheses.

1. Test CRUD operations of SQL database with no partitioning. (12.5)

2. Test CRUD operations of NoSQL database with no partitioning. (12.5)
3. Test replicas are in sync for a SQL database. Will read from the csv files to check all files are written the same data. (15)
4. Test replicas are in sync for a NoSQL database. Will read from the csv files to check all files are written the same data. (15)
5. Test read-only mode enabled when one of the replicas are down, and back to normal mode when all the replicas are healthy for a SQL database. (25)
6. Test read-only mode enabled when one of the replicas are down, and back to normal mode when all the replicas are healthy for a NoSQL database. (25)
7. Test CRUD operations of a SQL database with horizontal partitioning. Will read from the csv files to check the partition works correctly. (15)
8. Test CRUD operations of NoSQL database with horizontal partitioning. Will read from the csv files to check the partition works correctly. (15)
9. Test CRUD operations of SQL database with vertical partitioning. Will read from the csv files to check the partition works correctly. (25)
10. Test caching. Will read from the Coordinator cache and check the cache refreshed correctly after reading and writing. (5)
11. Test concurrency of the system by checking data correctness after sending multiple INSERT requests concurrently. (15)

The grading for each test case reflects the complexity and time required for implementation of the associated features. Two of the most challenging features are vertical partitioning and the read-only mode:

- Vertical partitioning: Implementing vertical partitioning is particularly challenging due to the need for reassembling data from multiple partitions. Each read operation requires gathering and rearranging columns from different partitions.
- Read-only mode: The read-only mode requires a heartbeat mechanism to monitor the health of the database replicas. When a replica is identified as down, the system must automatically reroute read requests to healthy replicas while simultaneously blocking all write operations.

Appendix 1: HTTP API document

1./create

Method: POST

Body:

- statement (String, required): The SQL statement or command.
- databaseType (String, required): The type of database, valid values are "SQL" or "NoSQL".
- replicaCount (int, required): The number of replicas.
- partitionType (String, required): The partition type, valid values are "horizontal", "vertical", or "none".
- numPartitions (int, required): The number of partitions.
- verticalPartitionColumns (List of List of String, required): Specific to vertical partitioning, lists the columns used for partitioning.

2. /insert

Method: POST

Body:

- statement (String, required): The SQL insert statement.
- databaseType (String, required): The type of database ("SQL" or "NoSQL").

3. /select

Method: POST

Body:

- statement (String, required): The SQL select statement.
- databaseType (String, required): The type of database ("SQL" or "NoSQL").

4. /update

Method: POST

Body:

- statement (String, required): The SQL update statement.
- databaseType (String, required): The type of database ("SQL" or "NoSQL").

5. /delete

Method: POST

Body:

- statement (String, required): The SQL delete statement.
- databaseType (String, required): The type of database ("SQL" or "NoSQL").