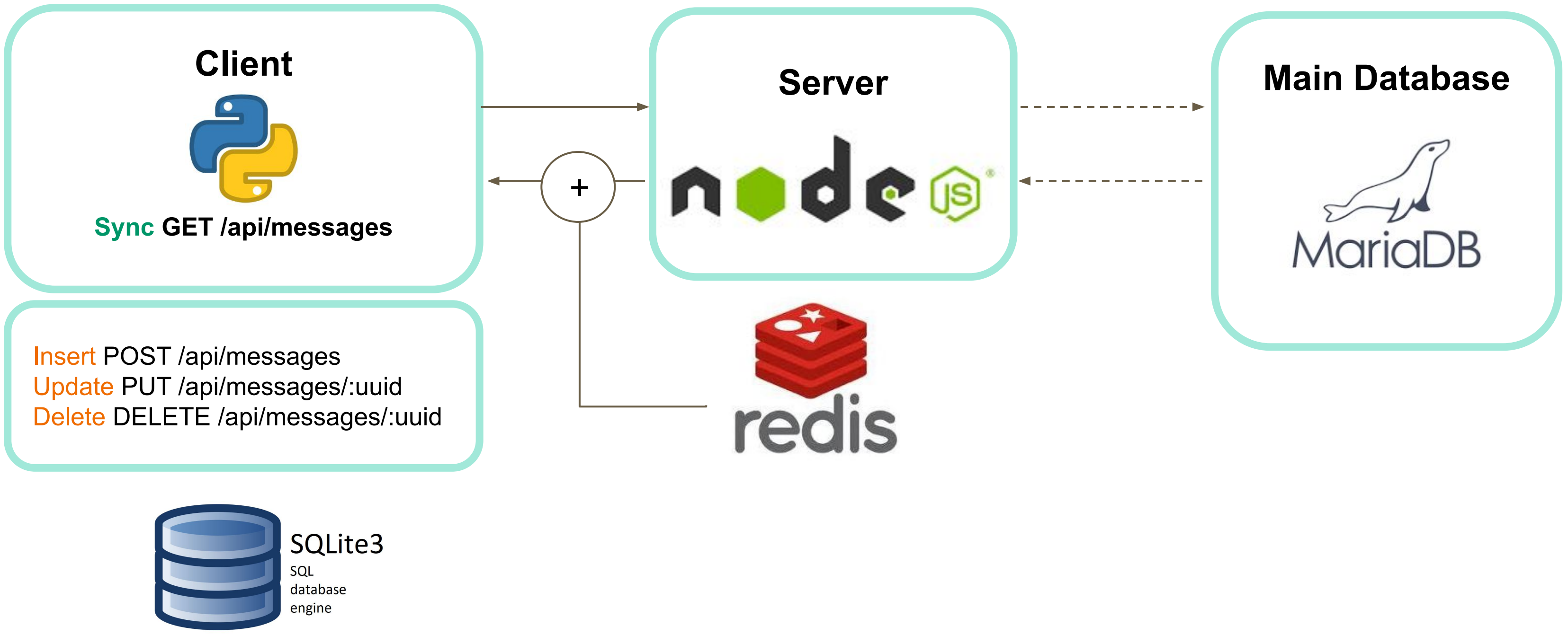
Midterm Project - Data Sync

Chitvisut Poomontree
6272017021

Agenda

- ❏ Overall Architecture
- ❏ Performance Optimization
- ❏ Cost Minimization
- ❏ How it all works

“Overall Architecture”


Client
Sync GET /api/messages
Server
Main Database
MariaDB
+
redis
Insert POST /api/messages
Update PUT /api/messages/:uuid
Delete DELETE /api/messages/:uuid
SQLite3
SQL
database
engine

“Overall Architecture”

Client

SQLite3
SQL database engine

data table (recent sync)

uuid	author	message	likes	count

Server

redis

Key : count

uuid

Key : data (limit 200k)

JSON.stringufy(data)

Main Database

MariaDB

data table (All data)

uuid	author	message	likes	isdelete	count

“Performance Optimization”

To speed up WRITE/UPDATE/READ/DELETE

- **WRITE/UPDATE**
  - Check duplicate post or nonexist put on cache (no need to check at DB)

- **READ**

  Caching
  - If the data is cache in the Node server, client can read data from in-memory database (Redis)

  Indexing
  - If the data is not in cache, the server shall query from the main DB (MariaDB) which its schema is indexing with “count” field since the query is condition by range of “count”

- **DELETE**

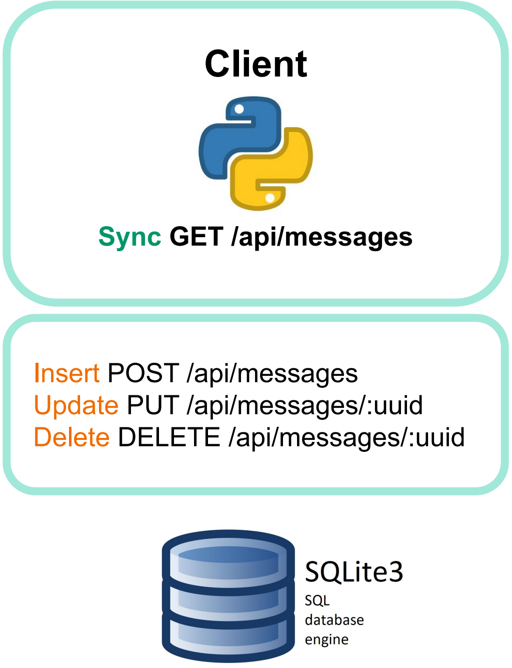
  Flag
  - The main database shall not completely delete data from the table, but it will flag the delete row with “isdelete” field

“Cost Minimization”

To Minimize Data Transfer

- **Client self-database (SQLite)**
  - If the data in SQLite is still valid (checking from response), Client will read from it DB (no replicate data send from server)
- **Server Cache (Redis)**
  - If the server has all the request data, it shall query from it own cache (no request send to main DB)
  - If the server has a partial of the request data, it shall request only the missing data form DB

“How it all works - Client”


Client
Sync GET /api/messages
Insert POST /api/messages
Update PUT /api/messages/:uuid
Delete DELETE /api/messages/:uuid
SQLite3
SQL database engine

- ❏ To send a GET request, client will first check it latest update transaction count (“rcount”) and send that count with the GET request
- ❏ On receiving the response, Client will check the response count number (“count”). If rcount == count, client will check the valid list (“valid”) from the response and query all valid uuid from it own database then make a csv file
- ❏ If there are new transaction (rcount<count), client will check the valid list (“valid”) from the response and query all valid uuid from it own database and combine with the data receive from server then make a csv file
- ❏ After the csv file is created, Client will update it database
  - ❏ DELETE all non valid data
  - ❏ INSERT all received data from server

“How it all works - Server”

Server

- For POST, the server will first check whether the uuid is already in cache. If that the case the server will send status 409 to client. On another hand, server will check the highest count number on that cache count table then update data table with count index. After that server will
  Insert row to main DB
- For PUT, the server will first check whether the uuid is exist in count table cache, if not the server will send 404 to client. If exist, the server will update it count table cache and data table cache with new count index then update data in main DB
- For Delete, the server will check directly in mainDB whether it has the request uuid then flag with isdelete
- And the update count table and delete data in data table

“How it all works - Server”

Server

- For GET, the server will first check the rcount number request. If rcount == count number the server will immediately response
- If there are new data, the server will check it available data. If it has all the data needed will will resend data in response. On the other hand, if it need more data from main database, it will send range of count index to DB and query on that range condition the combine all data and send back to client

Others Note

- Create SWAP space for Client and Server instance
- Cache policy
  - All transaction count in count table (only uuid with index so it is very small)
  - <=200,000 lates data in data table