HW1 Task2 Report
Haiping Xue
UIN:122006286

Time
6 hrs

Design Choice

The Key characteristics of most NoSQLs:
1. a more flexible data model (flexibility), higher scalability (scalability), and superior performance
2. most of NoSQL database have discarded the useful things in relational database: expressive query language, secondary indexes and strong consistency

In this task, we are given two NoSQL databases: MongoDB and Redis. We will first go through the features of two database.

MongoDB

1. Document model: MongoDB uses BSON format. It stores data in documents. BSON format enables MongoDB to internally index and map document properties and even nested documents.
2. Aggregation Framework: It enables users to obtain the kind of results for which the SQL GROUP BY clause is use. For batch processing of data and aggregation operations, MapReduce can be used.
3. MongoDB Ad hoc queries: It supports field, range queries, regular expression searches. Queries can return specific fields of documents and also include user-defined functions.
4. MongoDB is Schema – Less: The notation of schema is dynamic: each document can contain different fields. This is very helpful for modeling unstructured and polymorphic data. It makes it easier to evolve an application during development. Additionally, it provides the query robustness that developers have come to expect from relational database.
5. MongoDB Indexing: Indexes are created to improve the performance of searches. The good thing is that any field in a MongoDB document can be indexed with primary and secondary indices.
6. Duplication of data: MongoDB can run over multiple servers. The data is duplicated to keep the system up and also keep its running condition in case of hardware failure.
7. Load balancing: MongoDB has an automatic load balancing configuration because of data placed in shards.

8. Replication: MongoDB provides replication feature by distributing data across different machines. It can have one primary node and one or more secondary nodes. This typology is known as replica set. Replica set is like master-slave replication.

Redis

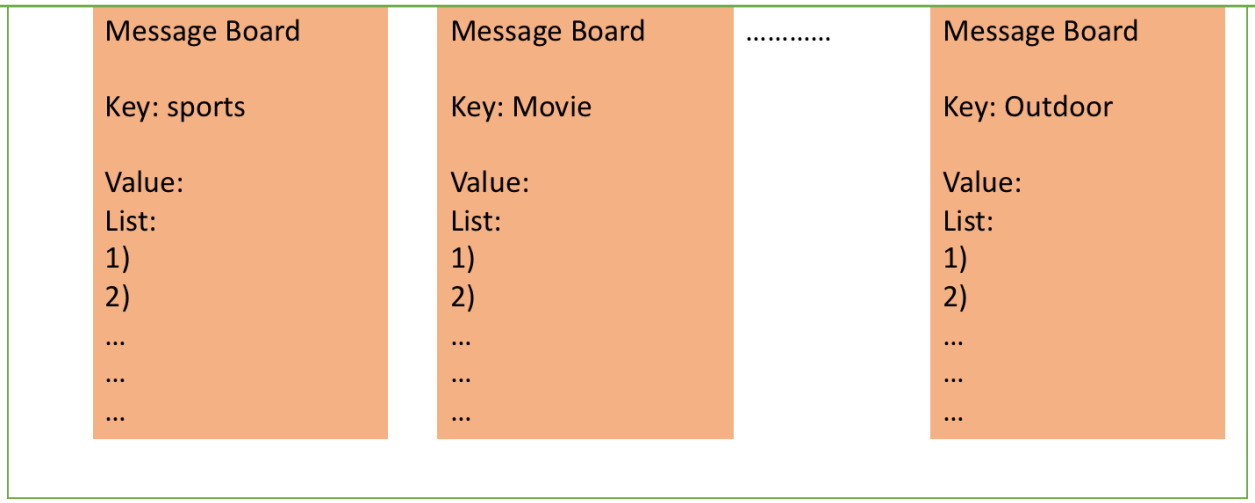
1. Key-value model: Redis stores the whole dataset in primary memory that's why it is extremely fast.
2. Persistence: While all the data lives in memory, changes are asynchronously saved on disk using flexible policies based on elapsed time and/or number of updates since last save.
3. Data Structures: Redis supports various types of data structures such as strings, hashes, sets, lists, sorted sets with range queries, bitmaps, hyperloglogs and geospatial indexes with radius queries.
4. Atomic Operations: Redis operations working on the different Data Types are atomic, so it is safe to set or increase a key, add and remove elements from a set and so on.
5. Master/Slave Replication: Redis follows a very simple and fast Master/Slave replication.

My design choice is to use Redis as my database for application. Here is a list of reasons.

1. Our application only needs the name of the board and retrieval of all the messages in the value. Document model can definitely satisfy this requirement. However, for simplicity and speed, I would use Redis. The redis key value model is simpler and thus will cause less time for development.
2. As many users are using the application at the same time, atomic operations become very important since we don’t want to lose any messages users write.
3. The data structure list is my choice of storage in which I can append all my messages to one list (board).
4. The Redis API has subscribe, unsubscribe and publish functions which fits into the API I need to implement.

Software Architecture

For my application, I choose to use python and Redis database to implement it. For each message board, I use one list in Redis which will store all the messages in one key. Therefore, each board will be mapped to one key. When clients write a new message, the message will be appended to the list of the corresponding key.



The application has a variable **selectedBoard** to keep track of the board client selected.

The **subscribing** variable will tell app if the user start listening to a message board.

The **channel** variable will keep the current channel that the user is listening to.

Commands

1. Select <board name>: when user enter this command, the select board variable will be updated.
2. Read: Since we use list in our implementation, we will run redisServer.lrange(selectedBoard, 0, -1) to read all the messages in the list.
3. Write <message>: The command will invoke a redisServer.lpush(selectedBoard, pushMessage) command that append the message to the corresponding list. Then it will run redisServer.publish(selectedBoard, pushMessage) to let all listeners received the message.
4. Listen: this command will call the following command:

   ```
   subscribing = True
   channel = redisServer.pubsub()
   channel.subscribe(selectedBoard)
   ```

   It set subscribing to true. Create a new pubsub object and choose the channel user want to subscribe. Then it will run channel.listen().
5. For stop, I use ctrl-c.

Running instructions

Specify port number with --port option when starting the redis server in the terminal:

redis-server --port portNumber

Use the same port number for the python script

python MessageBoardsClient.py -p portNumber

The code will use localhost for testing purpose.