Github Repo

https://github.com/boweill/cs6650/tree/main/Assignment3

Server Design

Pkg: Bean (for plain Java Objects)

DynamoDBHandler

Handles AWS credentials and creates DynamoDB client with those credentials.

Matches

- Corresponds to Matches schema speicified in API spec (https://app.swaggerhub.com/apis/IGORTON/Twinder/1.2)

MatchStats

- Corresponds to MatchStates schema specified in API spec (https://app.swaggerhub.com/apis/IGORTON/Twinder/1.2)

ResponseMsg

- Corresponds to the ResponseMsg schema specified in API spec (https://app.swaggerhub.com/apis/IGORTON/Twinder/1.2)

SwipeDetails

- Corresponds to the SwipeDetails schema specified in API spec (https://app.swaggerhub.com/apis/IGORTON/Twinder/1.2)
- Field 'action' added so that consumer can know whether the swiper liked or disliked swipe and use that information for data aggregation.

User

- Data model that describes a user in the database. Mapped to the User table in DynamoDB.
- Attributes:
  userId: String
  likedUsers: Set<String>
  likes: Integer
  dislikes: Integer
  matches: Set<String>

Pkg: QueueUtils

RMQChannelFactory

- Responsible for constructing individual channels used to publish to the queues.
- Reference from text book Ch 7.

RMQChannelPool

- A fixed sized channel pool where the PublisherRunnable threads can borrow channels from to publish messages.

- In this server the channel pool has a size of 10;
- Reference from text book Ch 7.

Pkg: Servlet

SwipeServlet

- The Servlet that handles swipe requests from client.
- In its init() method it establishes connection to RabbitMQ and sets up a channel pool with a fixed amount of channels initialized;
- In its doPost() method it borrows a channel from its channel pool, publishes to a fanout exchange which will push the message to all the queues it's aware of, returns the channel to the pool, and send back a response in the format of "${swiper} liked / disliked ${swipee}".

MatchesServlet

- The Servlet that handles get requests to /matches/{userId}.
- In its init() method it uses DynamoDBHandler to create a DynamoDB client and then uses the client to create a DynamoDBMapper which helps with mapping POJO to DynamoDB tables.
- In its doGet() method it loads the user with {userId}, gets the matches of the user and returns them in an array in a Matches object.

StatsServlet

- The Servlet that handles get requests to /stats/{userId}.
- In its init() method it uses DynamoDBHandler to create a DynamoDB client and then uses the client to create a DynamoDBMapper which helps with mapping POJO to DynamoDB tables.
- In its doGet() method it loads the user with {userId}, gets the likes and dislikes of the user and returns them in MatchesStats object.

Consumer Design

Pkg: Bean (for plain Java Objects)

DataWriter

- The class that handles writing to database.
- Create a DynamoDB client and DynamoDBMapper in its constructor.
- The handleSwipeDetails method writes individual records to DynamoDB, due to higher latency caused by frequent reads/writes was not used in load testing but the code was kept as a reference.
- The batchProcess method is called to write to the database all the users that have been updated during message polling. In this design the consumer calls batch process once each time it has collected 500 messages or has waited more than 10 seconds.

DynamoDBHandler

- Handles AWS credentials and creates DynamoDB client with those credentials.

- It's createTable() method is used by the consumer to create the User table in cases where the table does not exist.

SwipeDetails

- Corresponds to the SwipeDetails schema specified in API spec (https://app.swaggerhub.com/apis/IGORTON/Twinder/1.2);
- Field 'action' added so that consumer can know whether the swiper liked or disliked swipe and use that information for data aggregation.

Pkg: Consumer

LikeConsumer

- The class that establishes connection RabbitMQ and starts numOfThreads number of individual consumer threads that uses give exchangeName, queueName to consume from the queue.
- Contains a main method which starts a LikeConsumer object with the command line arguments (exchangeName, queueName, numOfThreads) it received.

Pkg: Handler

LikeHandlerRunnable

- The Runnable that provides the functionality for each individual consumer threads.
- In its run() method, it creates a channel from the connection initiated by LikeConsumer and registers a callback function to consume from the queue specified by the queueName passed in from LikeConsumer. In this assignment it calls methods of the DataWriter to write to DynamoDB in its registered callback.

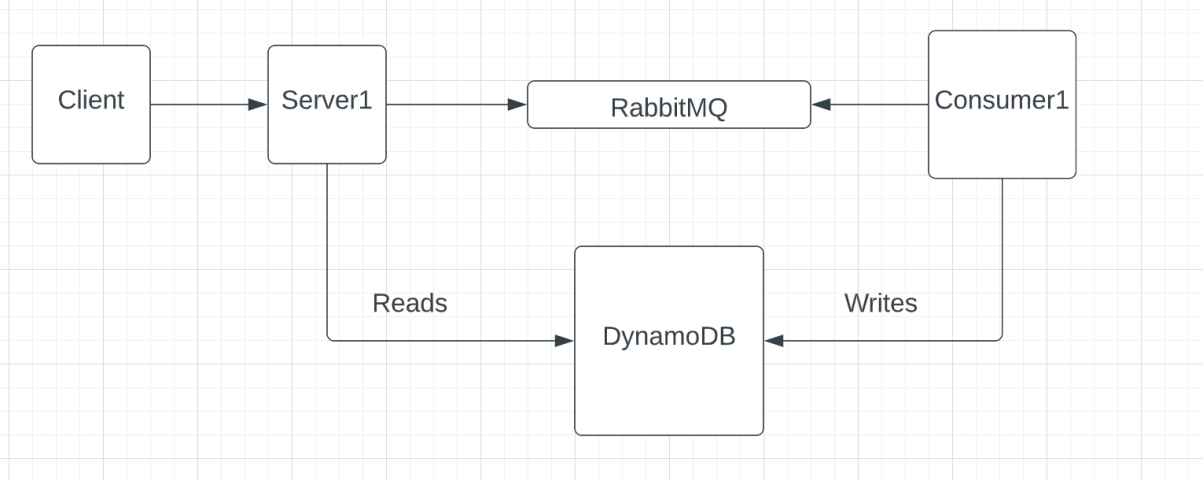
Database Design

Each document in the User table uses a set to store the IDs of the users they liked and another set to store their matches;

With each swipe request, we update the likedUsers set of the swiper and then checks if the swipee has liked the swiper back. If that is the case, we update the matches set for both of them. In this way we avoid the get requests getting increasingly expensive as we data volume increases.

Set Up

Client thread count: 50
Consumer thread count 100
DynamoDB provisioned Read/Write Capacity: 2000



Results

(Estimated throughput is acquired through sending 1000 requests with a single thread and calculating the estimate through Little's Law.)

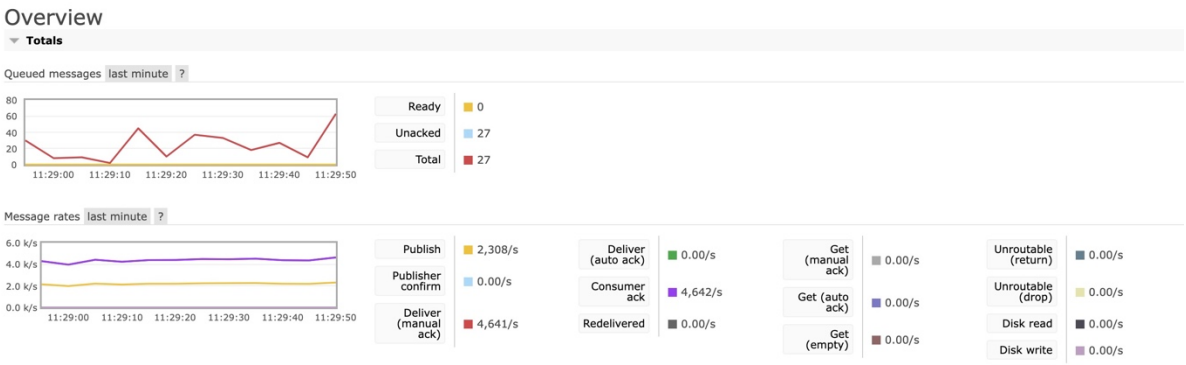
Assignment2 - 50 Client threads 100 Consumer threads

```
"/Applications/IntelliJ IDEA.app/Contents/jbr/Contents/Home/bin/java" ...
The requests took 225356 milliseconds to complete.
With 500000 successes and 0 failures.
Throughput: 2218.71
Mean response time: 22.52
Median response time: 21.00
99 percentile response time: 46
Minimum response time: 11
Maximum response time: 940

Number of Threads: 50
Estimated throughput: 2491.90

Process finished with exit code 0
```

Since we are using a fanout exchange with 2 queues bound we can see that the Deliver rate is roughly twice the publish rate.



Assignment 3 - 50 Client threads 100 Consumer threads (Shows 51 because we also have a GetThread)

(Estimated throughput is for POST requests)

```
The requests took 215228 milliseconds to complete.
With 500000 successes and 0 failures.
Throughput: 2323.12
Mean response time: 21.43
Median response time: 20.00
99 percentile response time: 46
Minimum response time: 11
Maximum response time: 203

Get Request Stats
Min latency was: 14
Max latency was: 130
Mean latency was: 33.14
Number of Threads: 51
Estimated throughput: 2867.63
```

We can see that the performance on client side remains robust after adding the database layer. And with batchWrite we were able to balance the latency difference in client sending requests and database operations on the client side.