CS6650 Assignment 3 Report

Shengdi Wang - wang.shengd@northeastern.edu

1. GitHub Repo URL

https://github.com/leowang396/distributed-music-service

2. A 1-2 page description of your server design. Include major classes, packages, relationships, how messages get sent/received, etc.

The same sets of technologies are used for the web server and database layer, namely Tomcat/Java Servlets and Amazon DynamoDB. Since we need to incorporate asynchronous processing for likes and dislikes, I added a message broker server (running the RabbitMQ server) and a worker server for processing the message queue in this assignment.

The main packages used are RabbitMQ Java Client, AWS SDK for DynamoDB, Gson, and Java Servlet.

Web Server/API Layer

The server handles HTTP requests from the client. Calls to the albums API are processed synchronously as in Assignment 2. On the other hand, calls to the review API are processed asynchronously by publishing messages to a RabbitMQ queue. This decouples the processing from the main request-response cycle and allows the server to respond quickly to the client.

To handle the new review API on the web server, I added a new servlet (*ReviewServlet* class) to the Tomcat server. To manage and improve the efficiency of server resource usage, I also added the *RMQChannelFactory* class which is a RabbitMQ channel factory.

Message Broker Server

RabbitMQ is used as the message broker. By running it on a separate server, I ensure that the RMQ server has enough file descriptors available for establishing potentially many connections with both the Web Server and the Worker Server. The message broker server receives messages published by the web server and pushes them to available consumer channels on the same queue.

Worker Server for Message Queue

The worker server consumes messages from the same queue that the web server publishes to. Whenever a message is pushed to a worker thread by the message broker, it processes the data by calling updateItem API in DynamoDB. In this server design, the results of the worker operations are not communicated with the client.

There are 2 major classes for the worker server. The *Main* class establishes multiple channels with the message broker using multiple threads, ensuring that there is only 1 channel per thread and every message is properly acknowledged or rejected. Since the worker server interfaces with the DynamoDB directly, I also added a *mqConsumerDdbDao* class which handles DynamoDB connection and requests.

Database

Similar to Assignment 2, I used DynamoDB as the database layer with pre-configured and pre-tuned read/write capacities, which ensures that there is no delay due to the auto-scaling of DynamoDB.

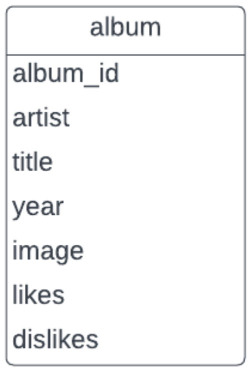


Figure 1: Assignment3 DynamoDB Data Model

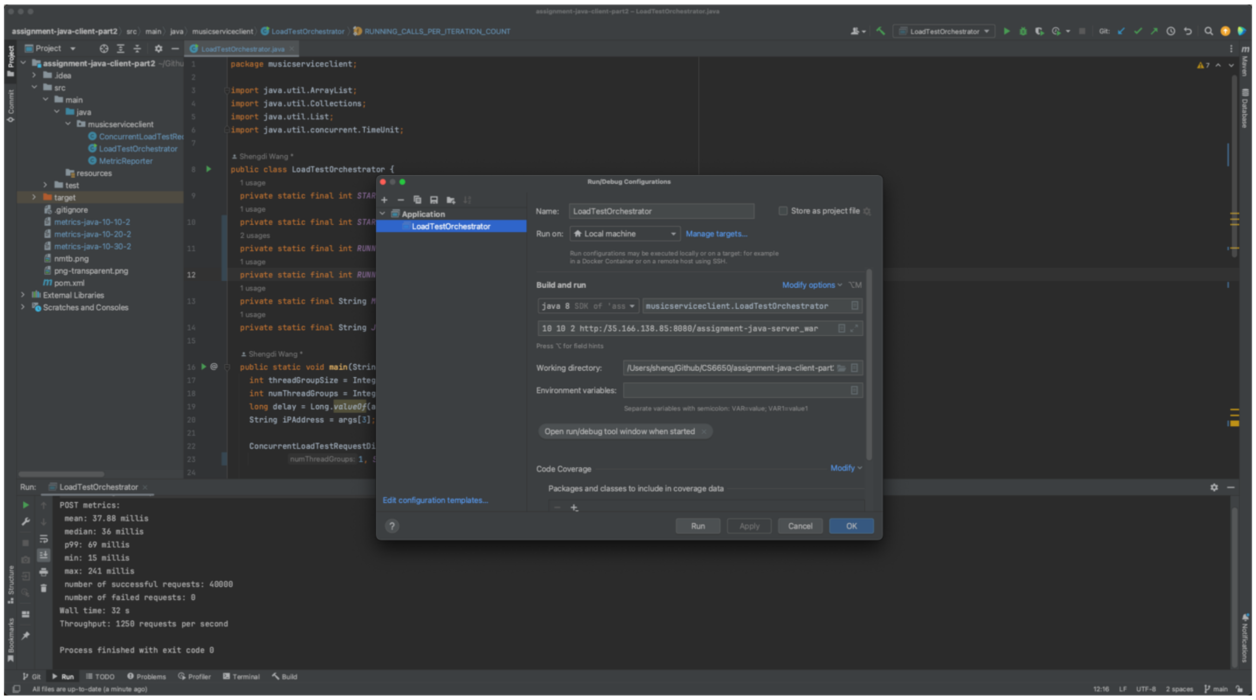
The data model is updated to include 2 new columns, "likes" and "dislikes", which count the number of like and dislike API calls made against the albums. By storing the columns in the same table as other attributes, the review API operations can be fulfilled in a single DynamoDB API call, instead of needing a read call followed by a write call to check whether the album ID is valid before writing it.

3. Test run results (command lines showing metrics, RMQ management windows showing queue size, send/receive rates) showing your best throughput.

My best throughput for each configuration is summarized in the table below. Under all configurations, the current server design achieves a throughput of > 1000 requests per second, which is comparable to that of Assignment 2.

Configuration	Wall Time (s)	Throughput
threadGroupSize = 10, numThreadGroups = 10, delay = 2	32	1250
threadGroupSize = 10, numThreadGroups = 20, delay = 2	53	1509
threadGroupSize = 10, numThreadGroups = 30, delay = 2	85	1411

Below are the screenshots for the command lines and RMQ management windows, which demonstrate that the RMQ queue lengths are kept below 25 throughout all test runs. This indicates that with 50 Tomcat server channels and 200 RabbitMQ consumer channels, the messages are processed efficiently, and the message queue remains small to be stable.



Run/Debug Configurations

Name: LoadTestOrchestrator

Run on: Local machine

Build and run: java 8 SDK of 'ass | musicserviceclient.LoadTestOrchestrator

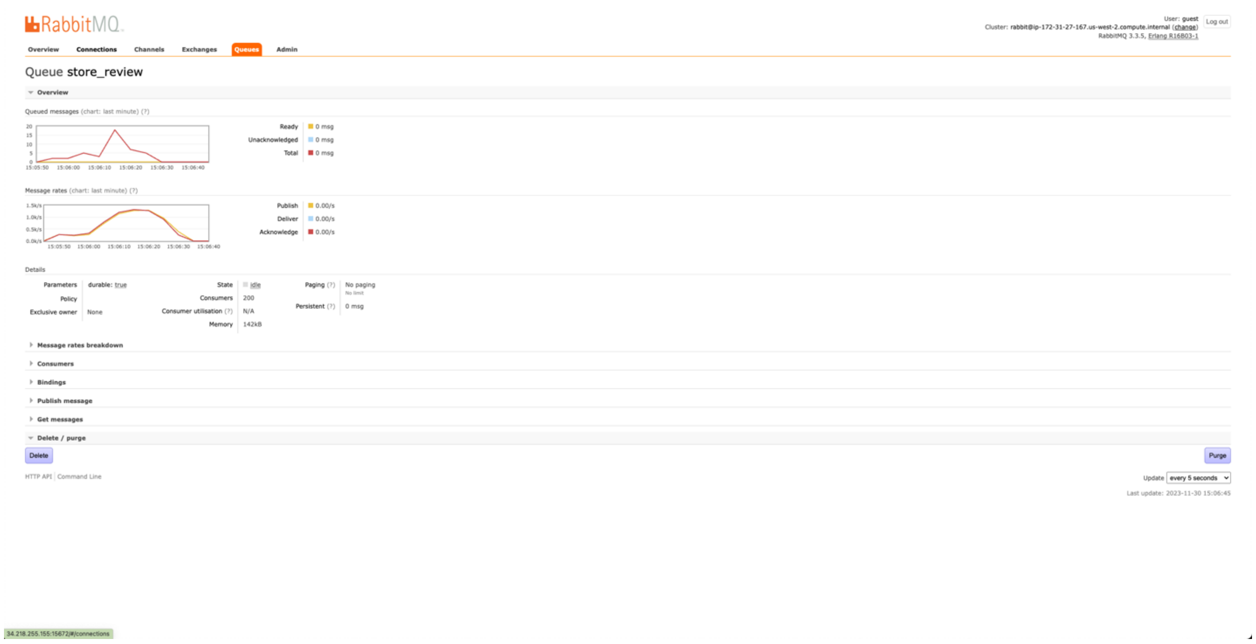
```
10 10 2 http:/35.166.138.85:8080/assignment-java-server_war
```

Working directory: /Users/sheng/Github/CS6650/assignment-java-client-part

```
POST metrics:
 mean: 37.88 millis
 median: 36 millis
 p99: 69 millis
 min: 15 millis
 max: 241 millis
 number of successful requests: 40000
 number of failed requests: 0
Wall time: 32 s
Throughput: 1250 requests per second

Process finished with exit code 0
```

Figure 2: Command line screenshot for numThreadGroups = 10



User: guest
Cluster: rabbit@ip-172-31-27-167.us-west-2.compute.internal (change)
RabbitMQ 3.3.5, Erlang R16B03-1

Queue store_review

Overview

Queued messages (chart: last minute) (?)

Ready	🟨 0 msg
Unacknowledged	🟦 0 msg
Total	🟥 0 msg

Message rates (chart: last minute) (?)

Publish	🟨 0.00/s
Deliver	🟦 0.00/s
Acknowledge	🟥 0.00/s

Details

Parameters	durable: true	State	idle	Paging (?)	No paging No limit
Policy		Consumers	200	Persistent (?)	0 msg
Exclusive owner	None	Consumer utilisation (?)	N/A		
		Memory	142kB		

Message rates breakdown
Consumers
Bindings
Publish message
Get messages
Delete / purge

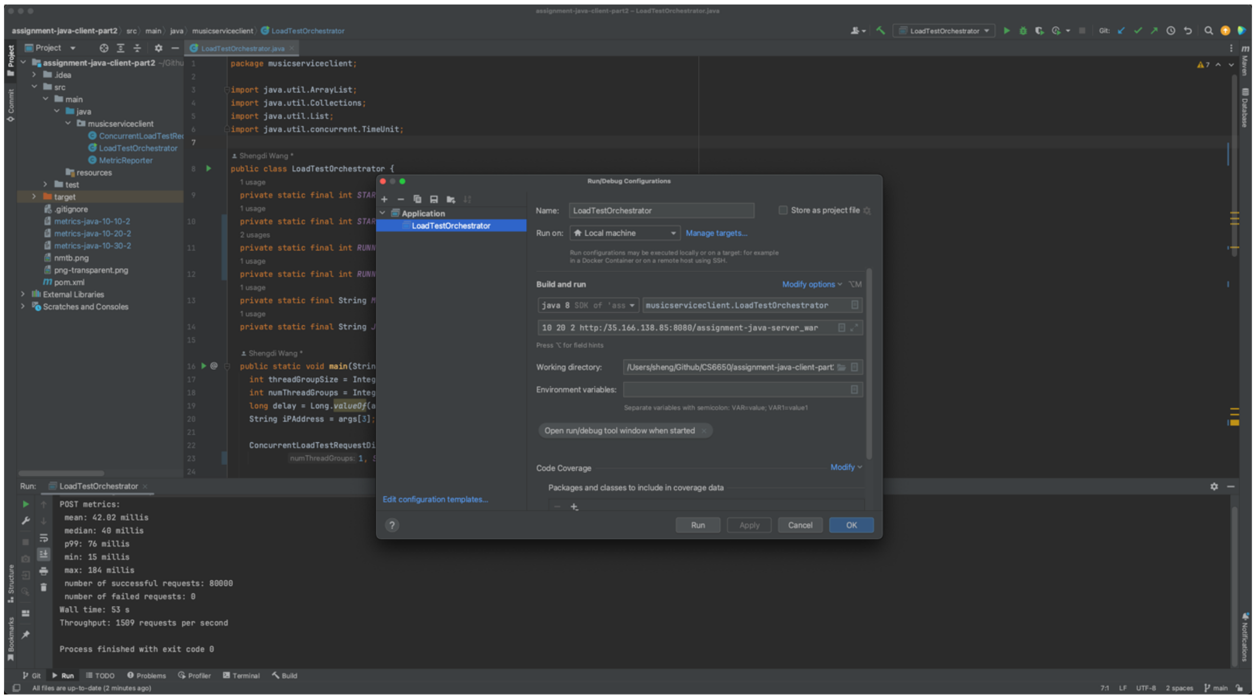
Delete   Purge

HTTP API | Command Line

Update every 5 seconds
Last update: 2023-11-30 15:06:45

34.218.255.155:15672/#/connections

Figure 3: RMQ screenshot for numThreadGroups = 10



```
package musicserviceclient;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.concurrent.TimeUnit;

public class LoadTestOrchestrator {
    private static final int STAR
    private static final int STAR
    private static final int RUNN
    private static final int RUNN
    private static final String M
    private static final String J

    public static void main(Strin
      int threadGroupSize = Integ
      int numThreadGroups = Integ
      long delay = Long.valueOf(a
      String iPAddress = args[3];

      ConcurrentLoadTestRequestDi
              numThreadGroups: 1, S
```

Name: LoadTestOrchestrator
Store as project file
Run on: Local machine   Manage targets...
Run configurations may be executed locally or on a target: for example in a Docker Container or on a remote host using SSH.

Build and run   Modify options
java 8 SDK of 'ass   musicserviceclient.LoadTestOrchestrator
10 20 2 http:/35.166.138.85:8080/assignment-java-server_war
Press ⌥ for field hints
Working directory: /Users/sheng/Github/CS6650/assignment-java-client-part
Environment variables:
Separate variables with semicolon: VAR=value; VAR1=value1
Open run/debug tool window when started
Code Coverage   Modify
Packages and classes to include in coverage data

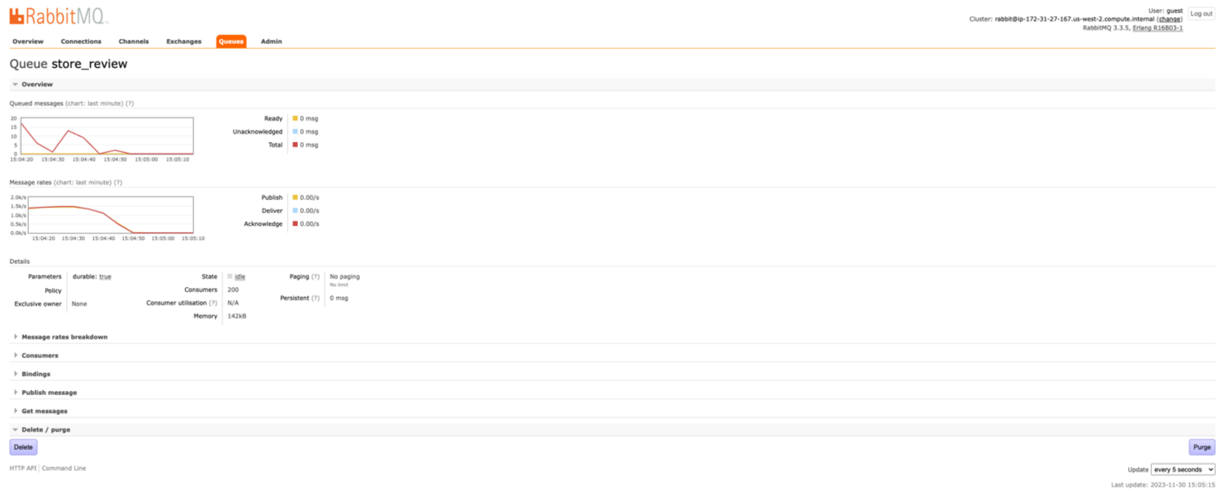
Run   Apply   Cancel   OK

Run: LoadTestOrchestrator

```
POST metrics:
 mean: 42.02 millis
 median: 40 millis
 p99: 76 millis
 min: 15 millis
 max: 184 millis
 number of successful requests: 80000
 number of failed requests: 0
Wall time: 53 s
Throughput: 1509 requests per second

Process finished with exit code 0
```

Figure 4: Command line screenshot for numThreadGroups = 20



User: guest
Cluster: rabbit@ip-172-31-27-167.us-west-2.compute.internal (change)
RabbitMQ 3.3.5, Erlang R16B03-1

Queue store_review

Overview

Queued messages (chart: last minute) (?)

Ready	0 msg
Unacknowledged	0 msg
Total	0 msg

Message rates (chart: last minute) (?)

Publish	0.00/s
Deliver	0.00/s
Acknowledge	0.00/s

Details

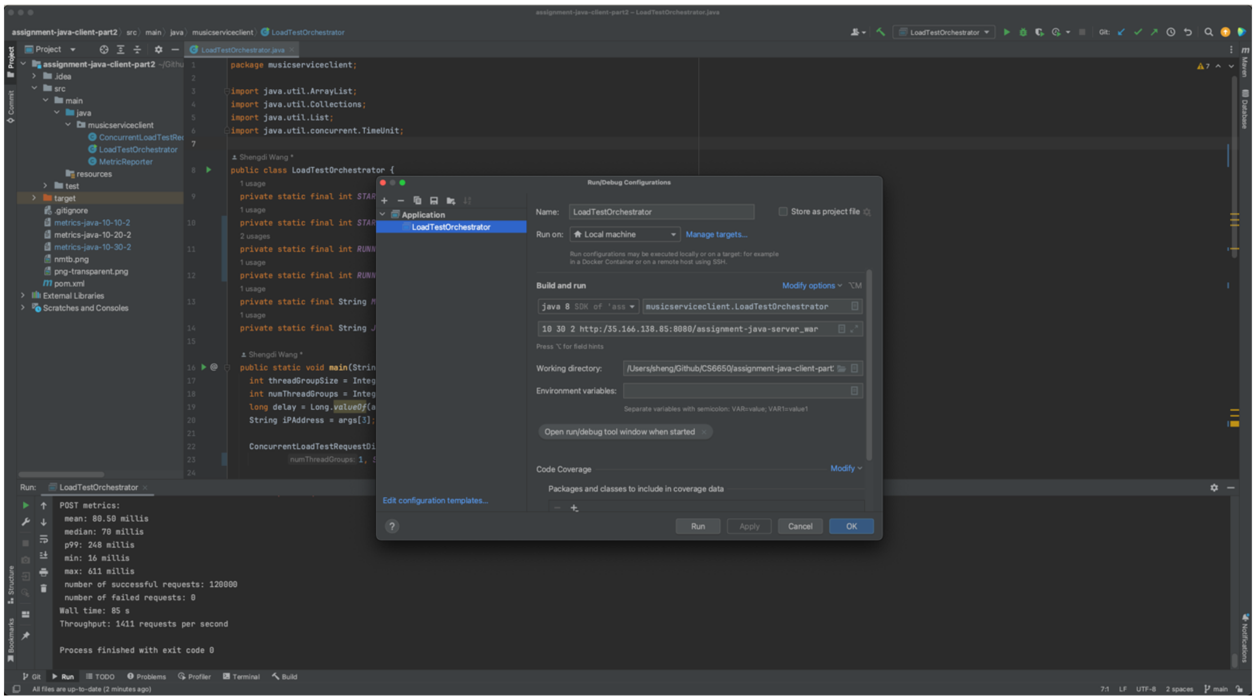
Parameters	durable: true	State	idle	Paging (?)	No paging No limit
Policy		Consumers	200	Persistent (?)	0 msg
Exclusive owner	None	Consumer utilisation (?)	N/A		
		Memory	142kB		

Message rates breakdown
Consumers
Bindings
Publish message
Get messages
Delete / purge

HTTP API | Command Line

Update every 5 seconds
Last update: 2023-11-30 15:05:15

Figure 5: RMQ screenshot for numThreadGroups = 20



```java
package musicserviceclient;

import java.util.ArrayList;
import java.util.Collections;
import java.util.List;
import java.util.concurrent.TimeUnit;

public class LoadTestOrchestrator {
  private static final int STAR
  private static final int STAR
  private static final int RUNN
  private static final int RUNN
  private static final String M
  private static final String J

  public static void main(Strin
    int threadGroupSize = Integ
    int numThreadGroups = Integ
    long delay = Long.valueOf(a
    String iPAddress = args[3];

    ConcurrentLoadTestRequestDi
            numThreadGroups: 1, 
```

Run/Debug Configurations

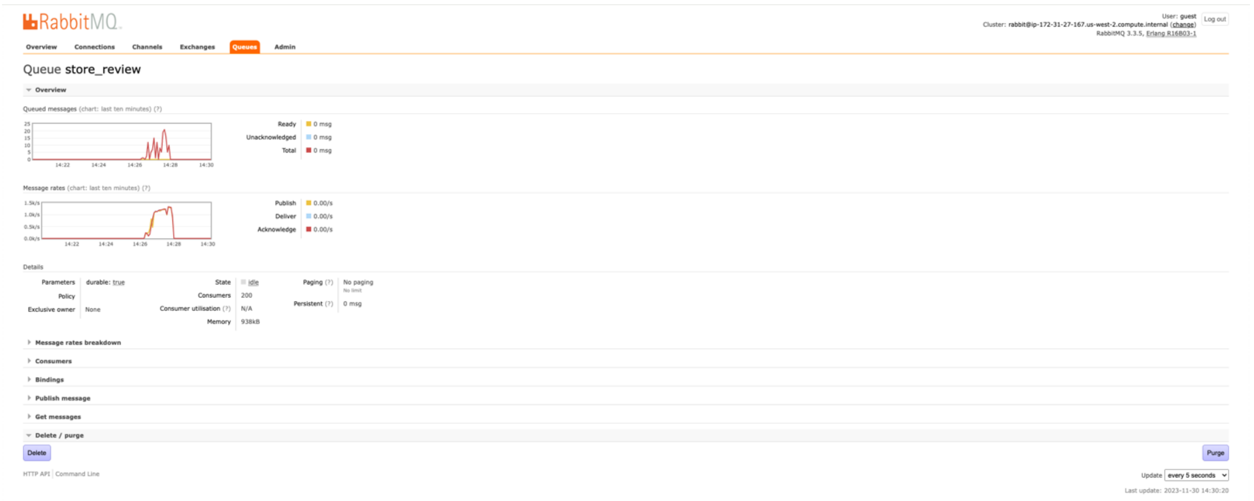
Name: LoadTestOrchestrator
Run on: Local machine
Build and run: java 8 SDK of 'ass | musicserviceclient.LoadTestOrchestrator
Program arguments: `10 30 2 http:/35.166.138.85:8080/assignment-java-server_war`
Working directory: /Users/sheng/Github/CS6650/assignment-java-client-part
Environment variables:
Open run/debug tool window when started
Code Coverage
Packages and classes to include in coverage data

Run: LoadTestOrchestrator

```
POST metrics:
 mean: 80.50 millis
 median: 70 millis
 p99: 248 millis
 min: 16 millis
 max: 611 millis
 number of successful requests: 120000
 number of failed requests: 0
Wall time: 85 s
Throughput: 1411 requests per second

Process finished with exit code 0
```

Figure 6: Command line screenshot for numThreadGroups = 30



User: guest
Cluster: rabbit@ip-172-31-27-167.us-west-2.compute.internal (change)
RabbitMQ 3.3.5, Erlang R16B03-1

Queue store_review

Overview

Queued messages (chart: last ten minutes) (?)

Ready	🟨 0 msg
Unacknowledged	🟦 0 msg
Total	🟥 0 msg

Message rates (chart: last ten minutes) (?)

Publish	🟨 0.00/s
Deliver	🟦 0.00/s
Acknowledge	🟥 0.00/s

Details

Parameters	durable: true	State	idle	Paging (?)	No paging No limit
Policy		Consumers	200	Persistent (?)	0 msg
Exclusive owner	None	Consumer utilisation (?)	N/A		
		Memory	938kB		

Message rates breakdown

Consumers

Bindings

Publish message

Get messages

Delete / purge

Update: every 5 seconds

Last update: 2023-11-30 14:30:20

Figure 7: RMQ screenshot for numThreadGroups = 30