CS6650 Project 2

Stefani Sindarto

GitHub Repository:

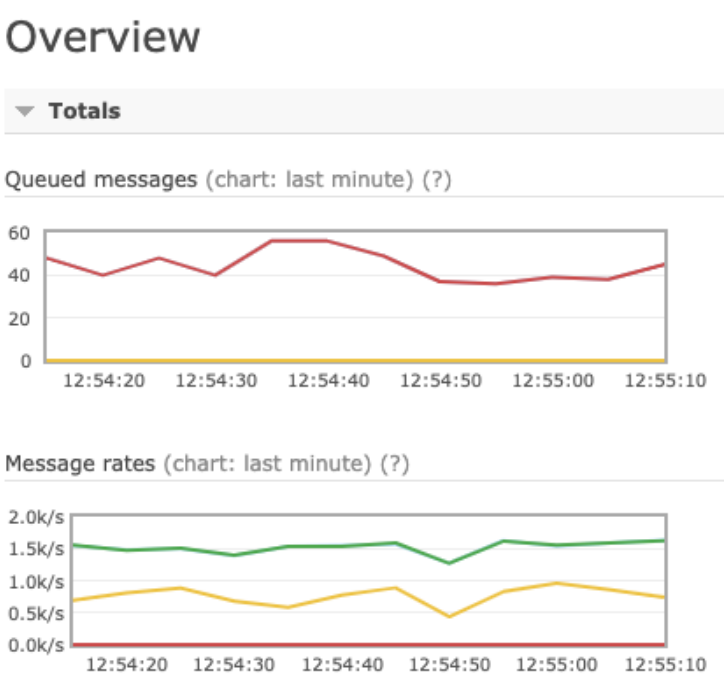
Server - https://github.com/stefanisindarto/Twinder2_Server

Consumers - https://github.com/stefanisindarto/Twinder2_Consumers

Client - https://github.com/stefanisindarto/Twinder

Result 1 (No Load Balancer)

```
Time elapsed: 134.619
Successful requests: 100000
Unsuccessful requests: 0
Throughput: 742.8371923725477
Mean: 26.593149999999525ms
Median: 25.0ms
P99: 55.0ms
Min: 15.0ms
Max: 649.0ms
```



Result 2 (Load Balancer)

```
Time elapsed: 94.773
Successful requests: 50001
Unsuccessful requests: 49999
Throughput: 1055.1528388887132
Mean: 29.279370000001002ms
Median: 27.0ms
P99: 71.0ms
Min: 16.0ms
Max: 1103.0ms
```

Overview

Totals

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

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

Design

- **3 separate projects are created for the purpose of this assignment as listed below:**
  - Twinder_Client (From previous assignment)
  - Twinder_Server (Updated from the previous assignment)
  - Twinder_Consumer (Consists of 2 main java classes - Consumer1 & Consumer2)

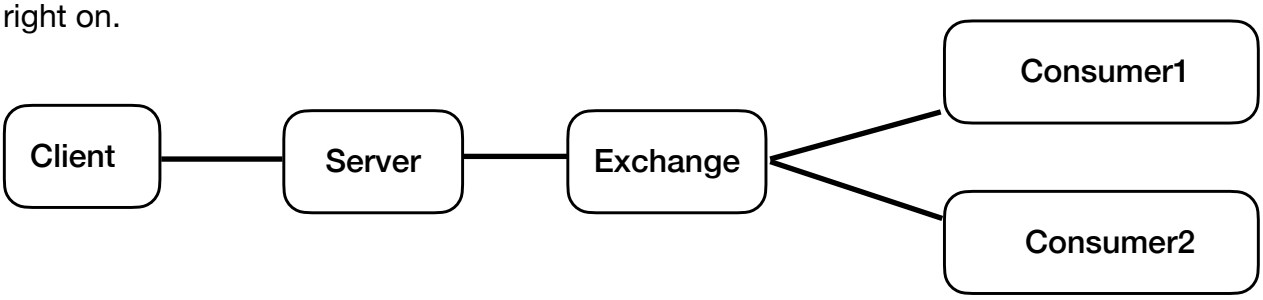
- **High Level Design**

The Twinder_Client project contains of Multithreaded class which create random body and send requests to the server(Twinder server).

In the server/producer class, the url and the body sent from the client is validated before it is published to a RabbitMQ exchange to be consumed by the consumers. The server class initiate an exchange which is used to fanout the messages to the consumer queues in the doPost method.

The Twinder_Consumer project contains a Swipe class and two consumer classes(Consumer1 and Consumer2). Each consumer class will bind the exchange to their respective queues and will consume the message to be converted as a Swipe object before processing the object into their respective hashmaps.

Consumer 1 owns a hashmap which stores the user id/swiper as the key and takes in a list of 2 integers. The list represents the count of like and dislike for each swiper. Consumer 2 owns another hashmap which stores the user id/swiper as the key and takes in a list of the swipees' id which the swiper swiped right on.



Classes

- **Class SwipeApiMultiThread:**
  - **main(String[] args)**
    When the program starts, the main function initiates the threads creation (creation of SwipeApiMultiThread object/thread) and execution. The main function also keeps track of the overall start and end time of the whole programs and calculates the time elapsed, number of successful & unsuccessful request and the throughput of the whole program.
  - **Void run()**
    Run function builds a random request body and left or right parameter, then calls the makeRequest function to the server. It increments the number of successful and unsuccessful requests.
  - **makeRequest(SwipeDetails body, String leftOrRight)**
    The function calls the apiInstance.swipeWithHttpInfo(body, leftOrRight) and returns true if the request is successful. If request is unsuccessful, the function will recursively retry 5 times before counting the request as unsuccessful.
  - **isSuccessful(int statusCode)**
    Returns true if the status code is either 200 or 201
  - **randomComment(Integer length)**
    Returns a random string of a given length
  - **randomNum(Integer range)**
    Returns a random number within a given range
  - **randomLeftRight(Integer randomNum)**
    Returns “left” if the randomNum value is even, otherwise returns “right”.
  - **SwipeApiMultithread(int threadNum, CountDownLatch completed)**
    Constructor of the SwipeApiMultithread instance. When called, it creates a new ApiClient instance and the SwipeApi instance and set the base path to call the server.

- **Class Consumer 1 & 2:**
  - **isLeft(String left right)**
    This is a helper function which returns true if the user/swiper swipes left indicating that the swiper and the swipee does not match. If the user/swiper swipes right, it returns false which indicates that the swiper and the swipee match.
- **Server Class:**

Added RabbitMQ dependency

  - **init()**
    initiate the connection to RabbitMQ, ConnectionFactory and set up the ChannelPool
  - **doPost(HttpServletRequest req, HttpServletResponse res)**
    This function validates the url and the body sent by the client. Once validated, a channel is borrowed from the pool, an exchange is declared, and then the message is sent to the exchange to be consumed by the consumers.