CS6650: Building Scalable Distributed Systems
Northeastern University
Fall 2023
Prof. Gorton

Assignment 3

Shangli Yu

1. *URL for your code repo.*

   https://github.com/uppb/cs6650_assignment3

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

   My design primarily consists of a servlet, RabbitMQ for message queuing, and a separate consumer service for database operations. The database is used to store all persistent information. This design decouples the webapp from data processing and storage.

   Data Flow The Servlet acts as a producer, sending messages to RabbitMQ. RabbitMQ queues messages, which are then consumed by the MessageConsumer. The Consumer processes messages and interacts with the database for data persistence.

   Main Components:

   (a) Servlet(ReviewServlet.java)

   Functionality:
   Handles POST requests, specifically for reactions (like/dislike) to albums. Extracts albumID and reaction from the request. Converts data into a JSON string and sends it to a RabbitMQ queue.

   Methods:

   - init: Initializes RabbitMQ connection.
   - doPost: Processes POST requests and publishes messages to the queue.
   - publishToQueue: Sends the message to the RabbitMQ queue.
   - destroy: Closes the RabbitMQ connection upon servlet destruction.

   (b) RabbitMQ Queue

   Queue Name:
   "reaction" (shared between the Servlet and the Consumer).

   Functionality:

   - Temporarily holds messages sent by the ReviewServlet.
   - Delivers messages to the consumer for processing.

   (c) Consumer(MQConsumer.java and MessageConsumer.java)

   Classes:

   - MQConsumer: Sets up the RabbitMQ consumer environment.

- MessageConsumer: Extends DefaultConsumer to handle message consumption.

Functionality:

- MQConsumer creates multiple consumer threads to process messages concurrently.
- MessageConsumer processes each message, extracts data, and saves it to the database.

Database Interaction:

Uses JDBC and a connection pool (BasicDataSource) for database operations.

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

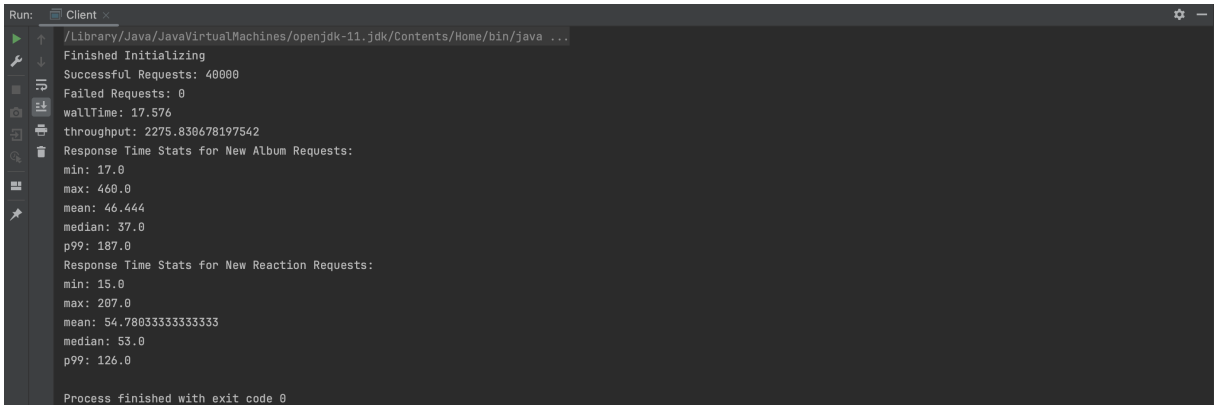

Figure 1: Single Servlet - threadGroupSize = 10, numThreadGroups = 10, delay = 2, consumers=10

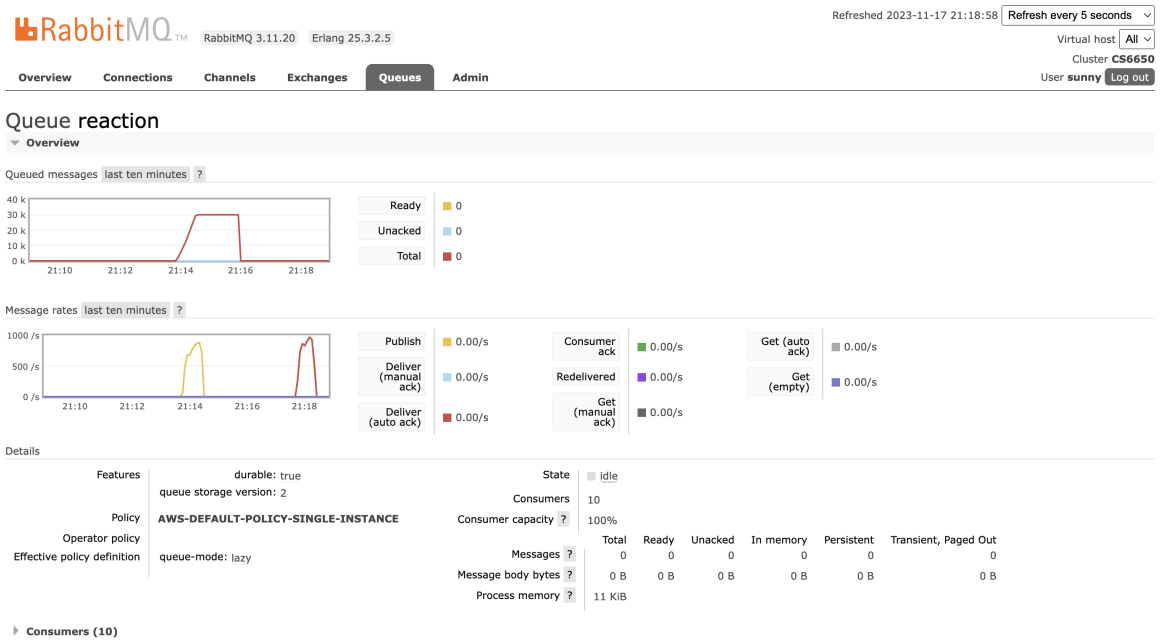

Figure 2: MQ Console - threadGroupSize = 10, numThreadGroups = 10, delay = 2, consumers=10 (The run around 21:18)

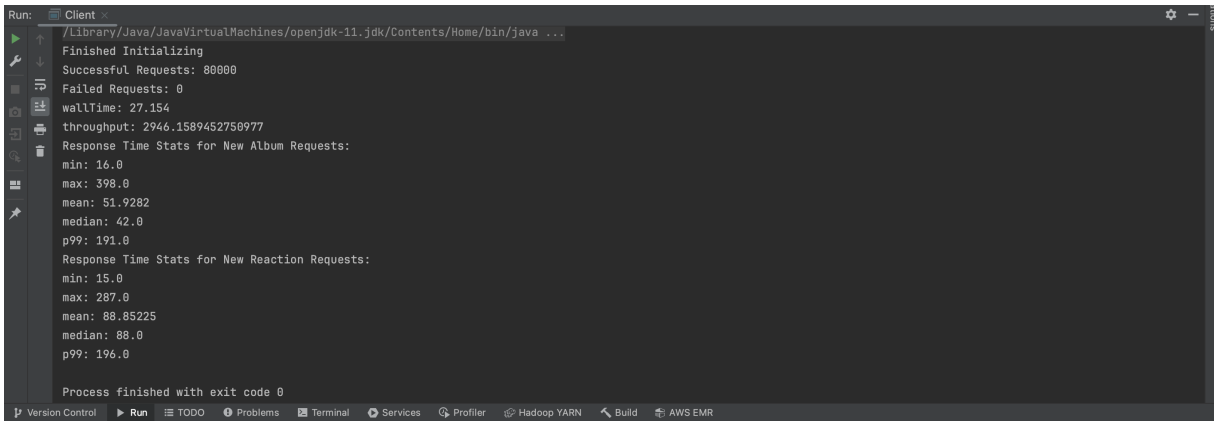

Figure 3: Single Servlet - threadGroupSize = 10, numThreadGroups = 20, delay = 2, consumers=10

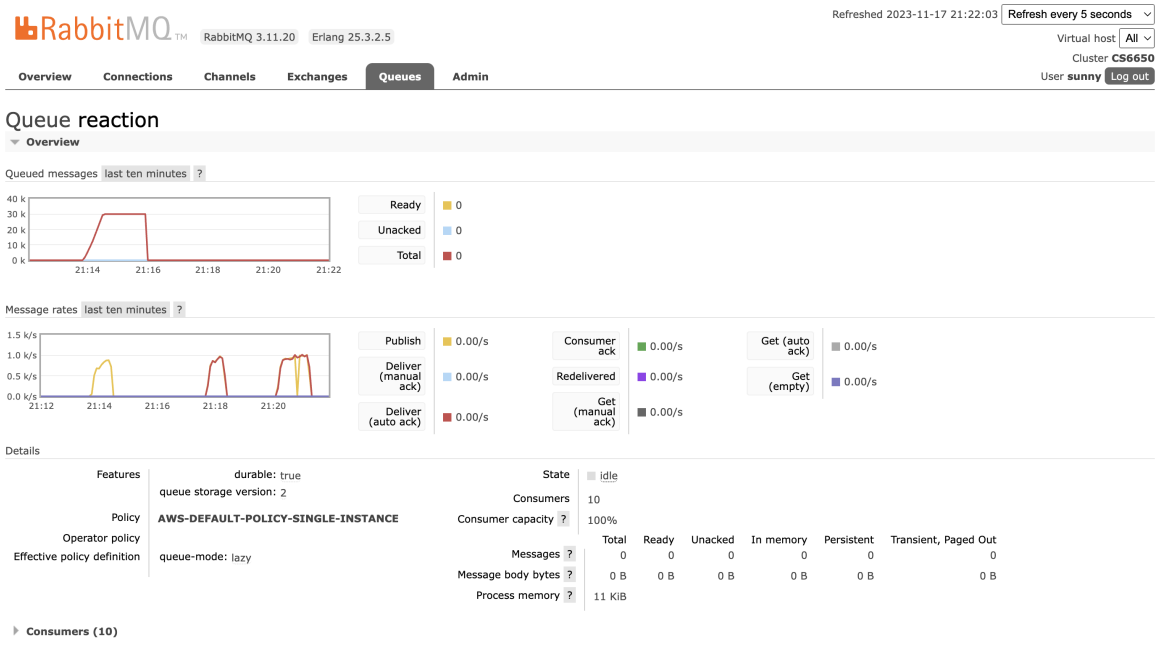

Figure 4: MQ Console - threadGroupSize = 10, numThreadGroups = 20, delay = 2, consumers=10

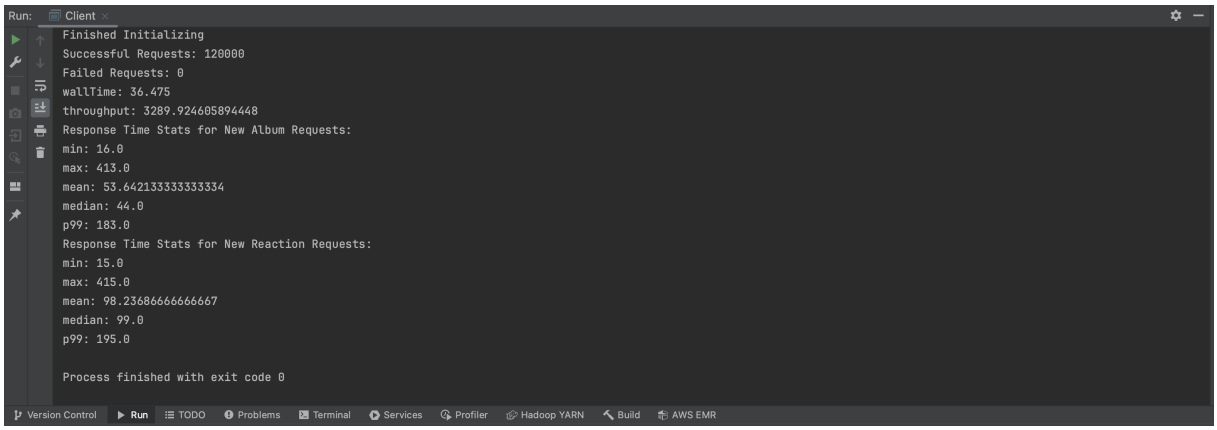

Figure 5: Single Servlet - threadGroupSize = 10, numThreadGroups = 30, delay = 2, consumers=10

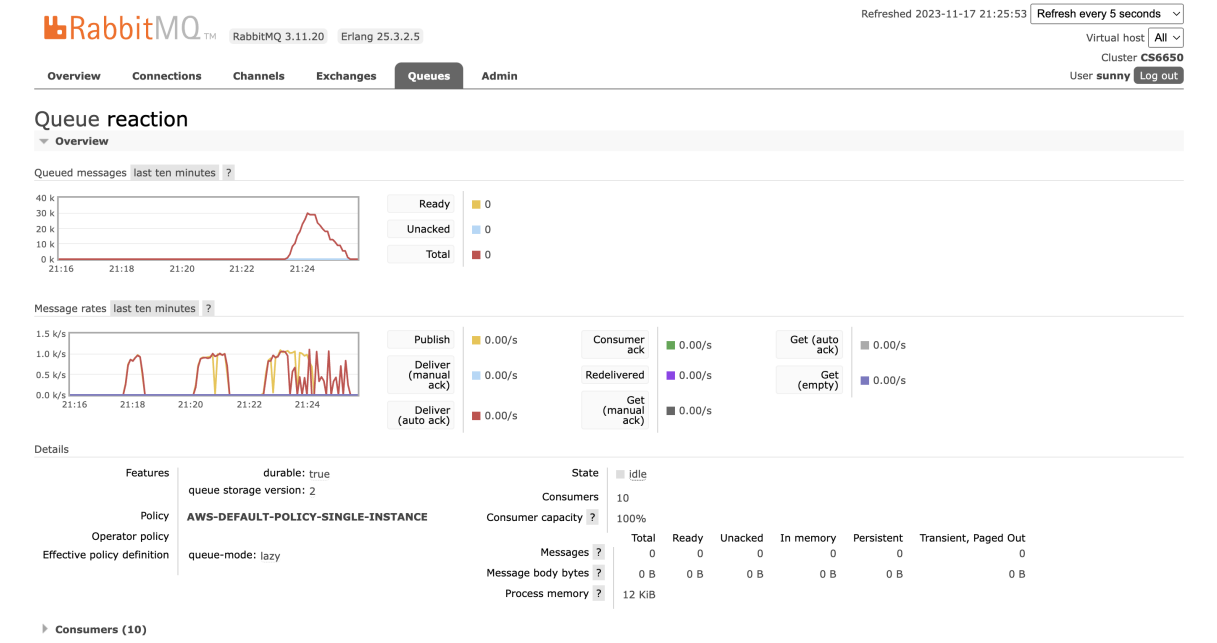

Figure 6: MQ Console - threadGroupSize = 10, numThreadGroups = 30, delay = 2, consumers=10

		Single Servlet		
Configuration		10/10/2	10/20/2	10/30/2
Wall Time		18	27	36
Throughput		2276	2946	3290
Post Album	Min	17	16	16
	Max	460	398	413
	Mean	46	52	54
	Median	37	42	44
	P99	187	191	183
Post Album Reaction	Min	15	15	15
	Max	207	287	415
	Mean	55	89	98
	Median	53	88	99
	P99	126	196	195

In order to maximize the system throughput, I monitored the servlet's cpu utilization. In the runs above, I initialized the Executor Service to have a fixed pool of 150 threads.

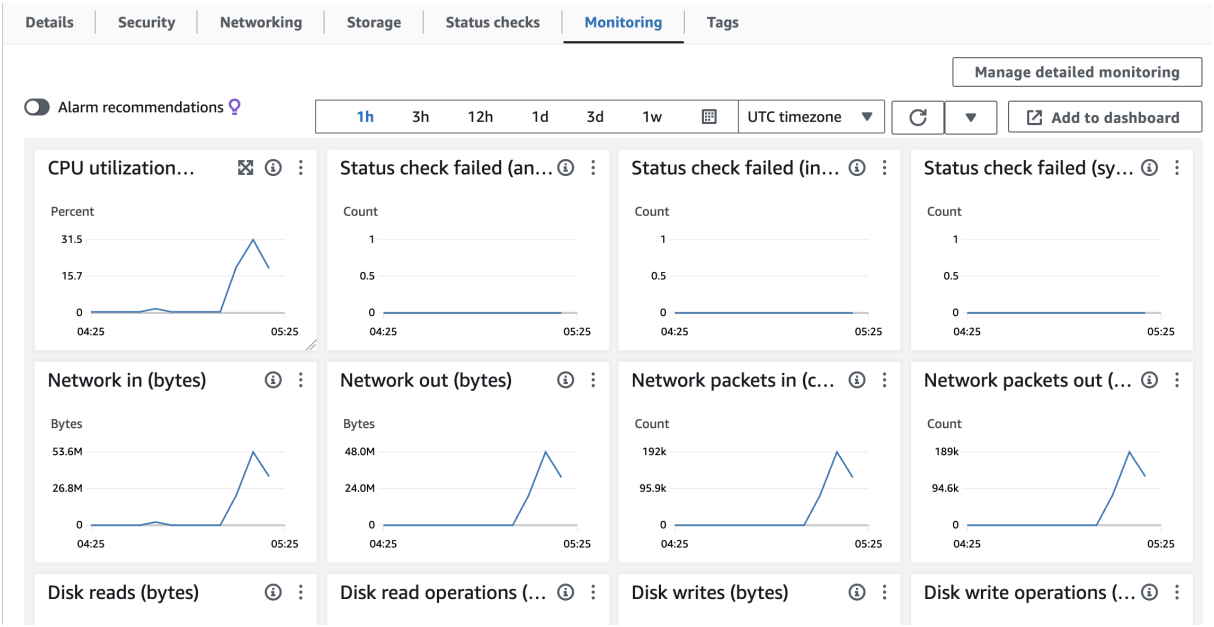

Figure 7: CPU Utilization when at most 150 threads firing post requests

As we can see, the cpu utilization was about 31.5%. The server was not under heavy load, so we should be able to load the server more. I then increased the number to 200.

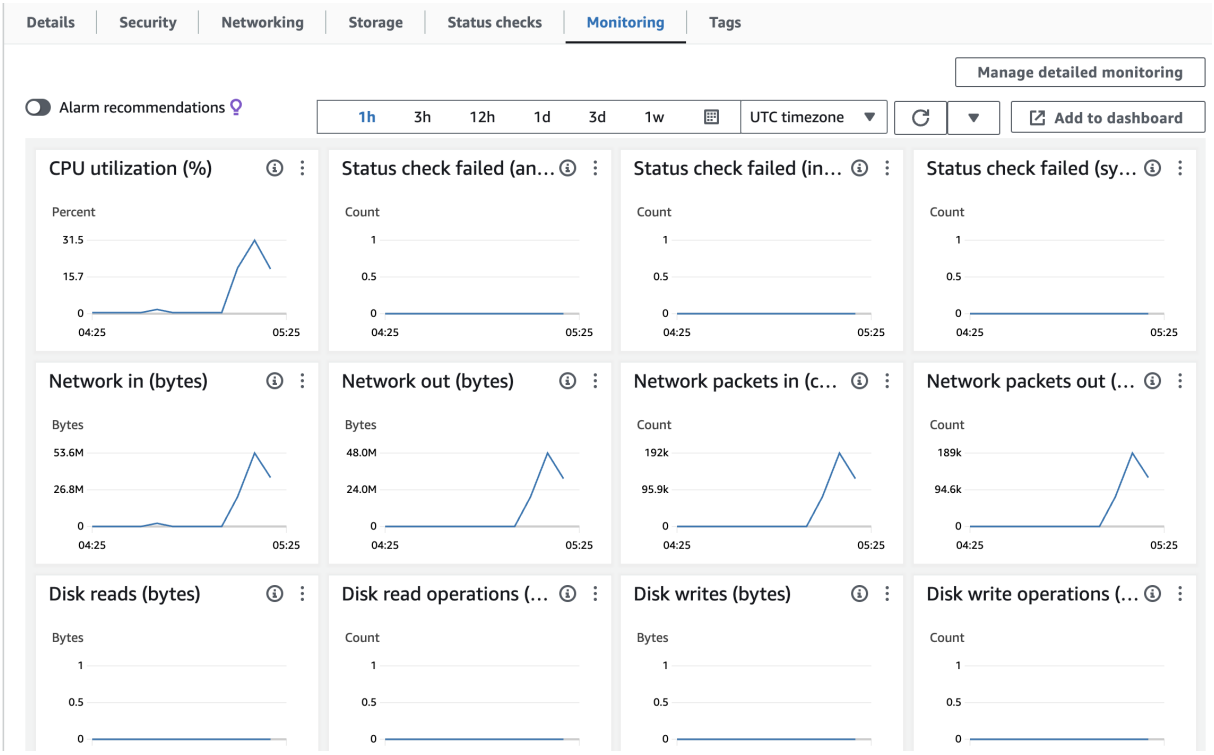

Figure 8: CPU Utilization when at most 200 threads firing post requests

From the utilization, we see that there is no significant difference after increasing the number. This is indicating that using 150 clients already maximized the throughput.

For the 10/30/2 configuration, we see a spike in messages queued that peaked around 30k messages. This indicates that we do not have enough consumers to process the information in time. Therefore, I monitored the database to confirm that the database server is not overloaded.

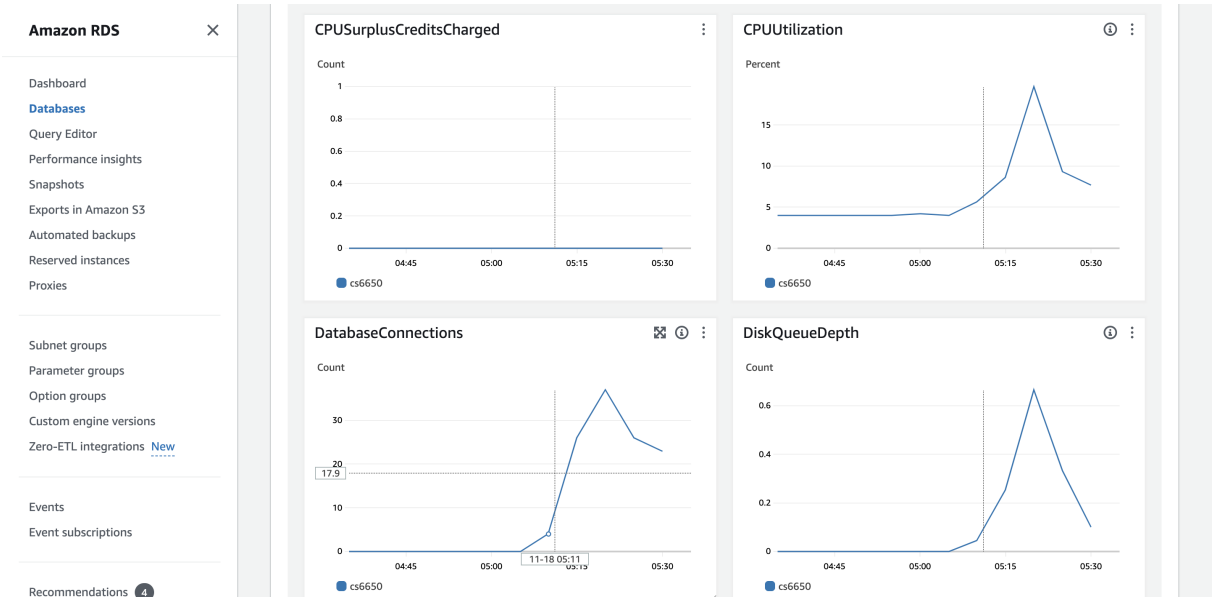

Figure 9: CPU Utilization on Database Server using 10 consumers

The cpu utilization and the number of connections are fairly low, so we can proceed to increase the number of consumers in our program. I doubled the amount of consumers to 20.

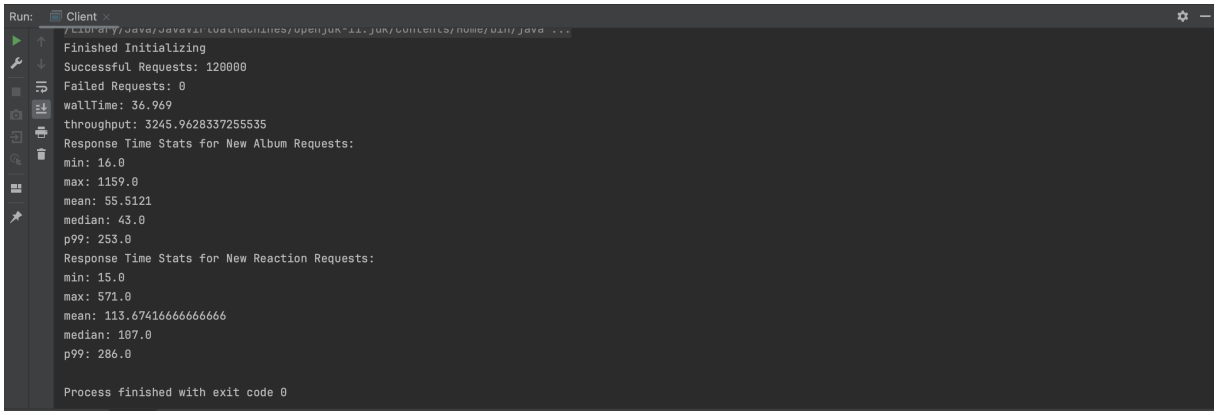

Figure 10: Single Servlet - threadGroupSize = 10, numThreadGroups = 30, delay = 2, consumers=20

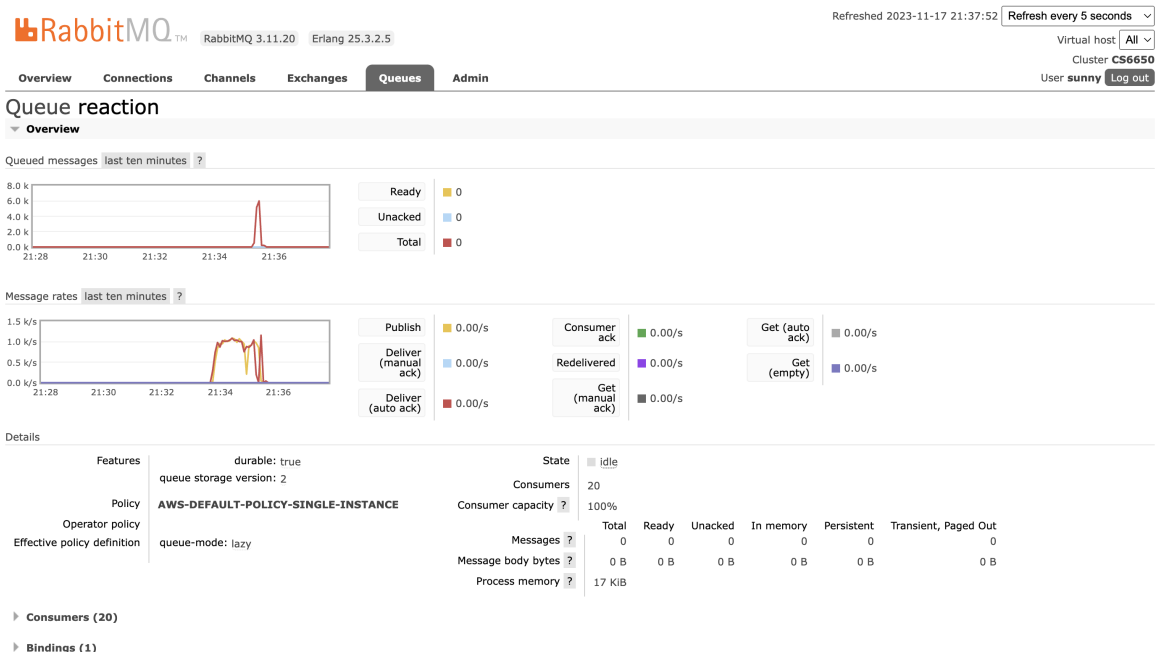

Figure 11: MQ Console - threadGroupSize = 10, numThreadGroups = 30, delay = 2, consumers=20 (The run around 21:18)

From the console, we can spot that the consume rate is still not catching up with the producing rate, with at most 6k messages in the queue. So I again added 10 consumers.

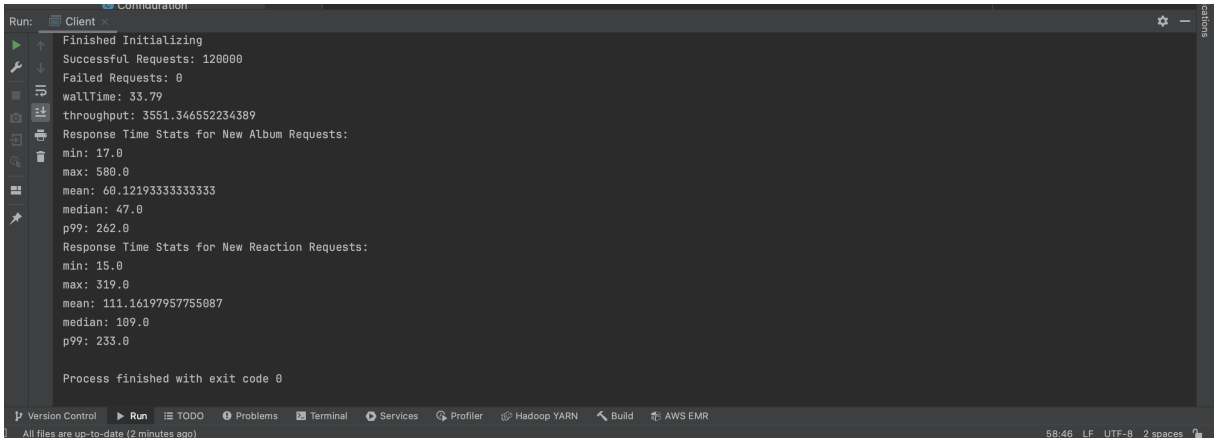

Figure 12: Single Servlet - threadGroupSize = 10, numThreadGroups = 30, delay = 2, consumers=30

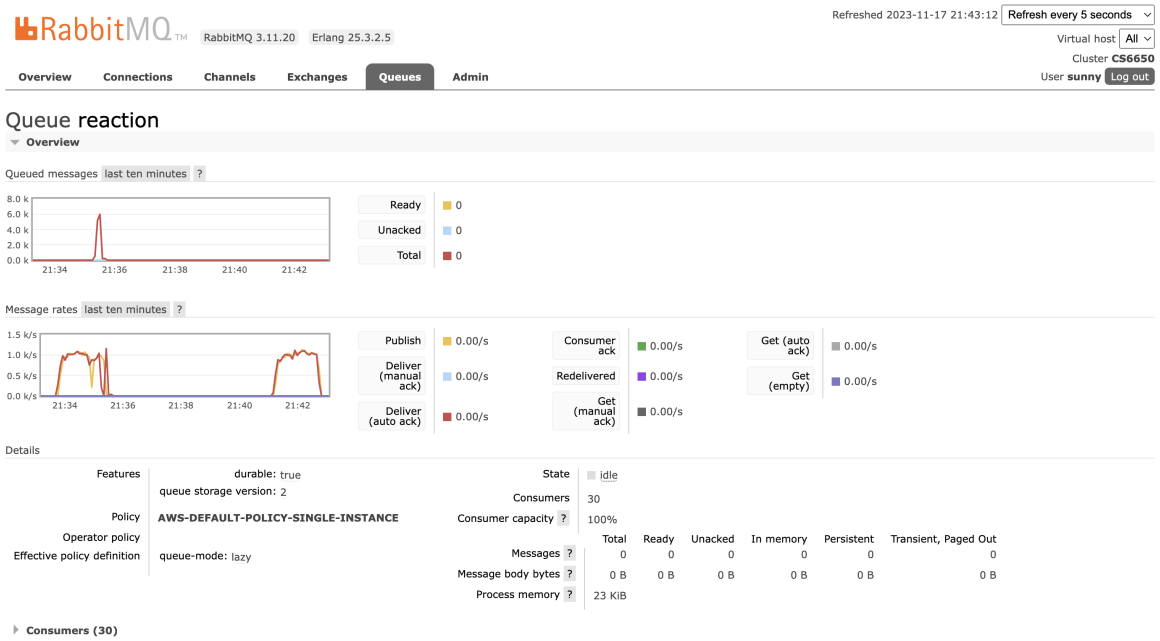

Figure 13: MQ Console - threadGroupSize = 10, numThreadGroups = 30, delay = 2, consumers=30

With 30 consumers, the number of messages queued in RabbitMQ is close to zero, meaning the consuming rate is greater or equal to the producing rate.