Table of Contents

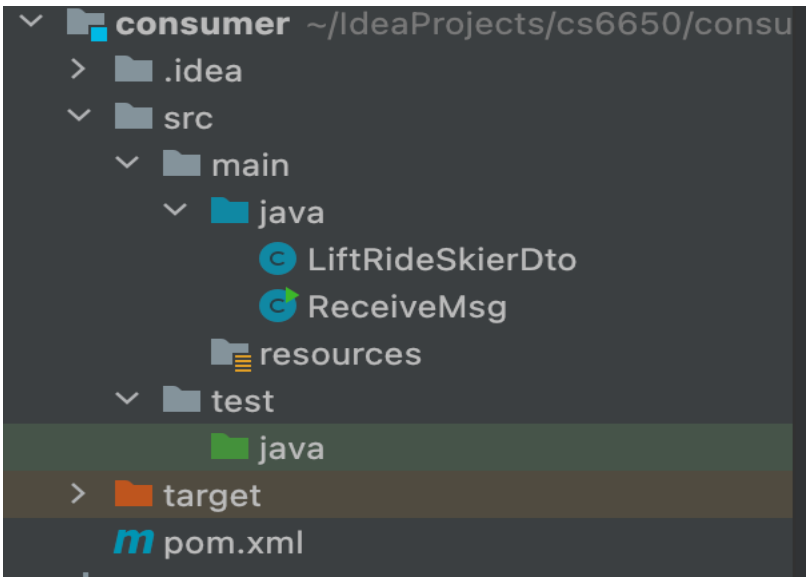
1. Git Repo

https://github.com/Peihao-Zhu/cs6650

2. Description of design

2.1 Consumer design

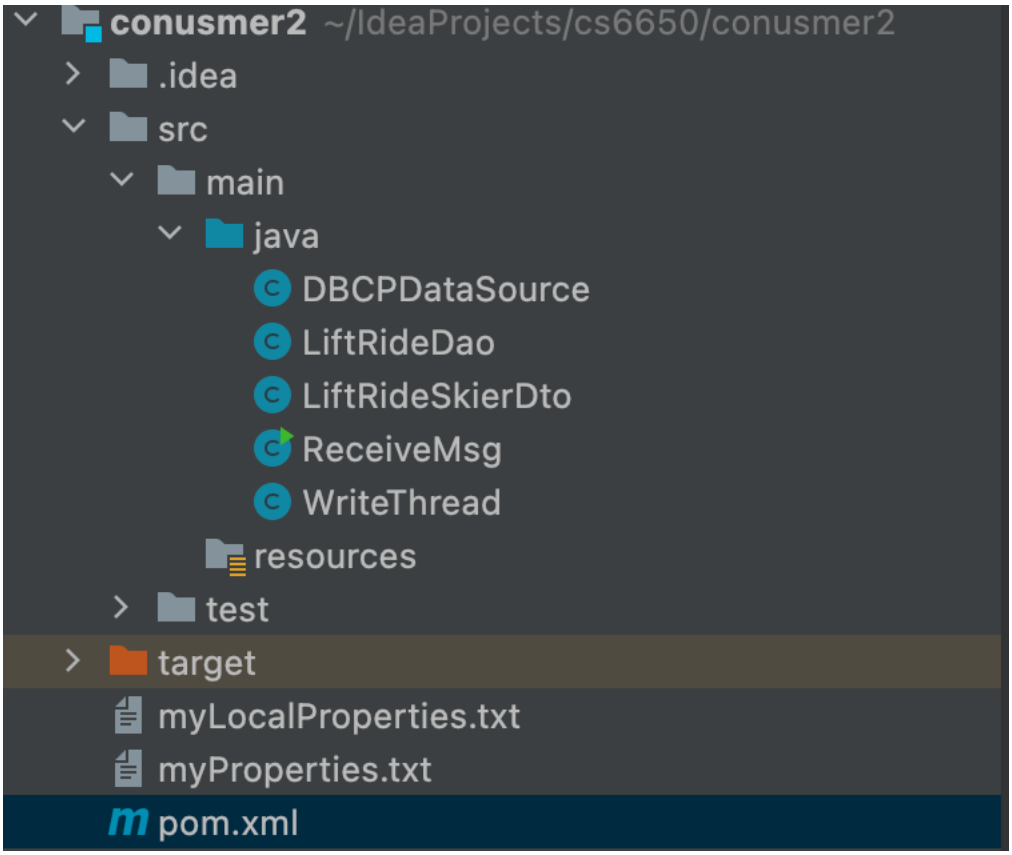
Consumer1



The ReceiveMsg class:
The ReceiveMsg class is a class used to receive message from rabbitmq and put it into Redis.

LiftRideSkierDto class:
LiftRideSkierDto class is an entity class converting the message into the object.

Consumer2



The ReceiveMsg class:
The ReceiveMsg class is a class used to receive message from rabbitmq, and add it into blockingqueue.

DBCPDataSource class:
DBCPDataSource class is a configuration class we get the connection with specific MySQL server.

LiftRideDao class:
LiftRideDao class is a consumer we fetch message from blocking queue.

LiftRideSkierDto class:

LiftRideSkierDto class is an entity class converting the message into the object.

WriteThread class:
WriteThread class is a thread which batch insert specific data into database.

2.2 Database design

In step1, as instruction required, I choose Redis as my persistent store. We know that Redis supports several data structures like string, hash, list, set etc. Actually, I came up with two data models

1. Grouping message by skier id, and value is a list which contains the message (call that **model 1**).

```
RList<LiftRideSkierDto> list = client.getList(String.valueOf(liftRideSkierDto.getSkierID()), new JsonJacksonCodec());
list.add(liftRideSkierDto);
```

In this case, each time I want to get one answer for a query, I just get the whole list of the skier from Redis. And later on, I can iterate over the list and filter the one we want.

2. Designing model for three questions specifically (call that **model 2**)

```
// question "For skier N, how many days have they skied this season?"
// key -> skier id
// value -> map
RMap<Integer, Map<String, Set<Integer>>> map = client.getMap( s: "skied-days-in-season", new CompositeCodec(StringCodec.INSTANCE, new JsonJa
Map<String, Set<Integer>> days = map.getOrDefault(skirID, new HashMap<>());
days.putIfAbsent(seasonIDStr, new HashSet<>());
days.get(seasonIDStr).add(dayID);
map.put(skirID, days);



// question "For skier N, what are the vertical totals for each ski day?" (calculate vertical as liftID*10)
// key -> skier id
// value -> map  key1  - > day id  value1 -> vertical
RMap<Integer, Map<String, Integer>> map1 = client.getMap( s: "vertical-skied-days", new CompositeCodec(StringCodec.INSTANCE, new JsonJacksonC
Map<String, Integer>  verticalMap = map1.getOrDefault(skirID, new HashMap<>());
verticalMap.put(dayIDStr, verticalMap.getOrDefault(dayIDStr,  defaultValue: 0) + liftID * 10);
map1.put(skirID, verticalMap);

// question "For skier N, show me the lifts they rode on each ski day"
// key -> skier id
// value -> map  key1  - > day id  value1 -> lifts number
RMap<Integer, Map<String, Integer>> map2 = client.getMap( s: "lifts-skied-days", new CompositeCodec(StringCodec.INSTANCE, new JsonJacksonCodec
Map<String, Integer>  liftsMap =  map2.getOrDefault(skirID,  new HashMap<>());
liftsMap.put(dayIDStr, liftsMap.getOrDefault(dayIDStr,  defaultValue: 0) + 1);
map2.put(skirID, liftsMap);
```

In this case, we assume these queries are fixed(only three), so I design the model customized. By using hash, I can aggregate those messages for skier id, then aggregate the number for ski day or season. Although it may take some time to put data, query response is very fast compared with the first choice.

In step2, I think the message got from rabbitmq is very structured. Why don't use relational database, and I can answer those questions by writing some Select Statement. Hence I just decided to use AWS RDS(MySQL). The CREATE TABLE STATEMENT in shown below.

DDL for LiftRides.LiftRides

```
CREATE TABLE `LiftRides` (
  `skierId` int DEFAULT NULL,
  `resortId` int DEFAULT NULL,
  `seasonId` int DEFAULT NULL,
  `dayId` int DEFAULT NULL,
  `time` int DEFAULT NULL,
  `liftId` int DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
```

2.3 Deployment topologies on AWS

I put all my applications include server, consumer1 and consumer2 into AWS ec2.

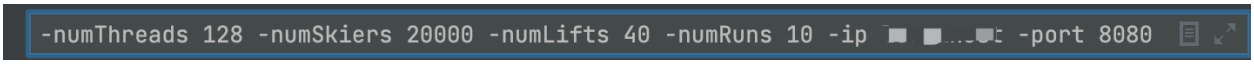
- Server code needs to run on Tomcat, so I start the tomcat first.
- All applications rely on rabbitmq server, so then I start the rabbitmq on EC2.
- Consumer1 will store data into Redis, so I need to run the Redis on EC2.
- Consumer2 need to connect with AWS RDS, so I will launch the RDS within the same VPC as EC2.
- Finally, I just run consumer1 and consumer2 in any order.

Test Results

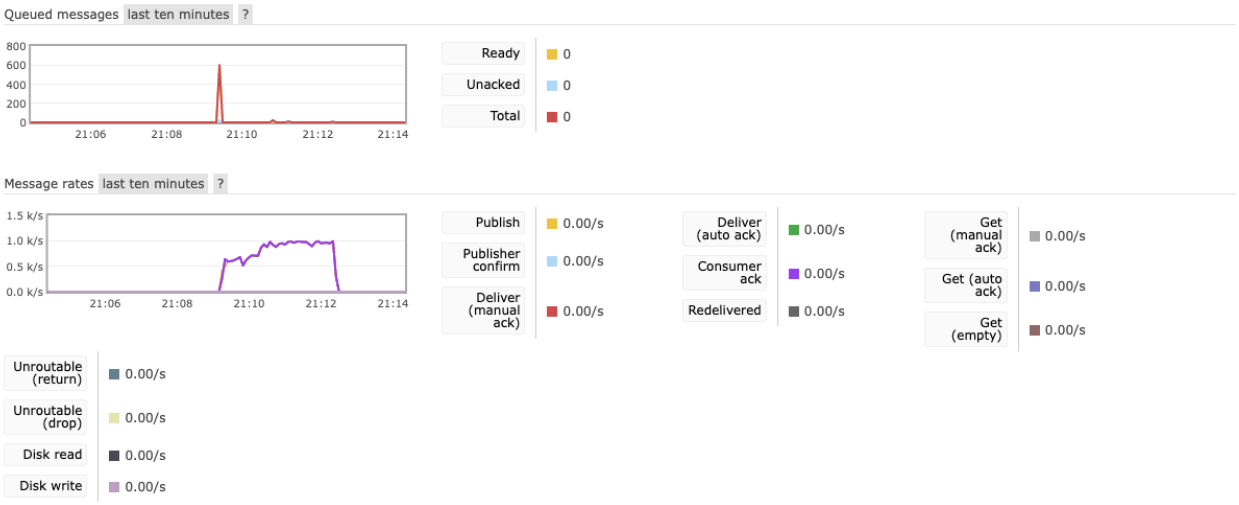
Results for step1

128 client threads

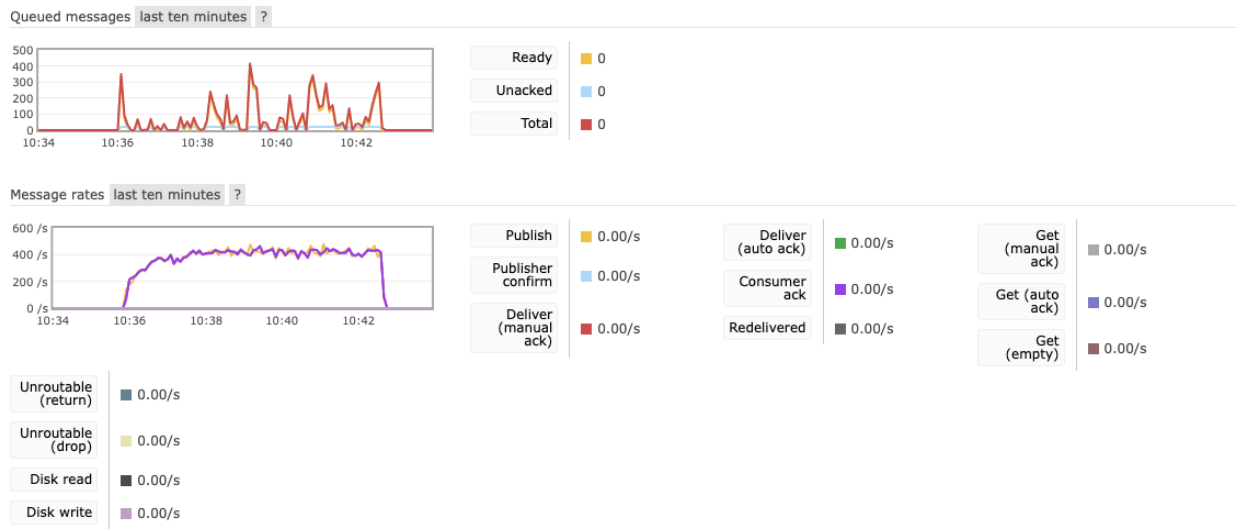
Command window:



RMQ management windows for model1



RMQ management windows for model2

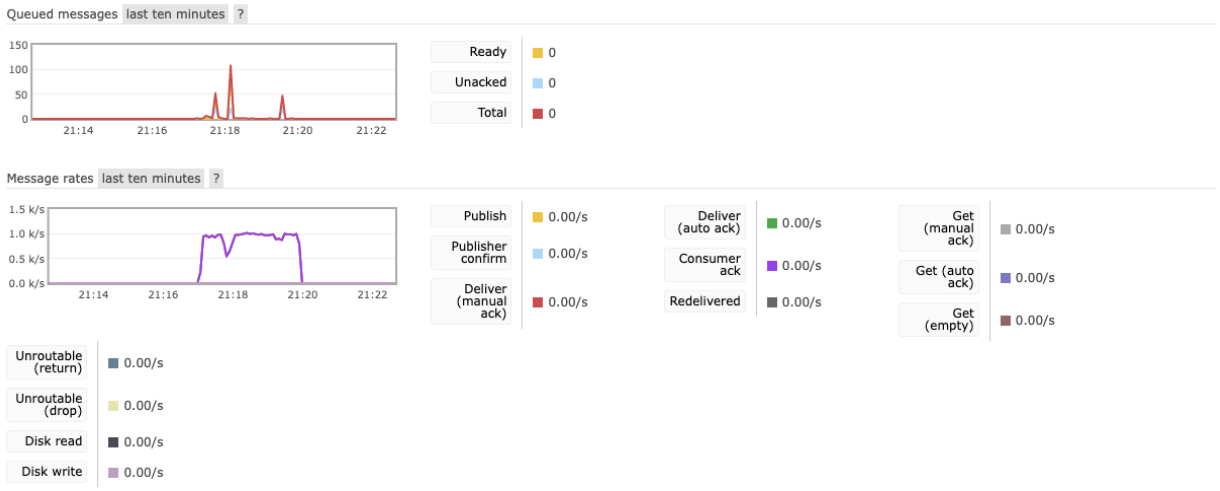


256 client threads

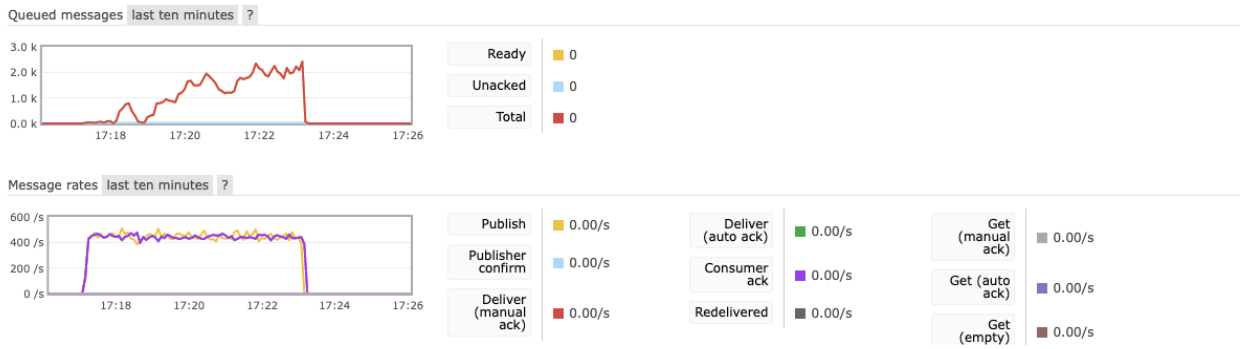
Command window:

```
-numThreads 256 -numSkiers 20000 -numLifts 40 -numRuns 10 -ip [redacted]: -port 8080
```

RMQ management windows for model1



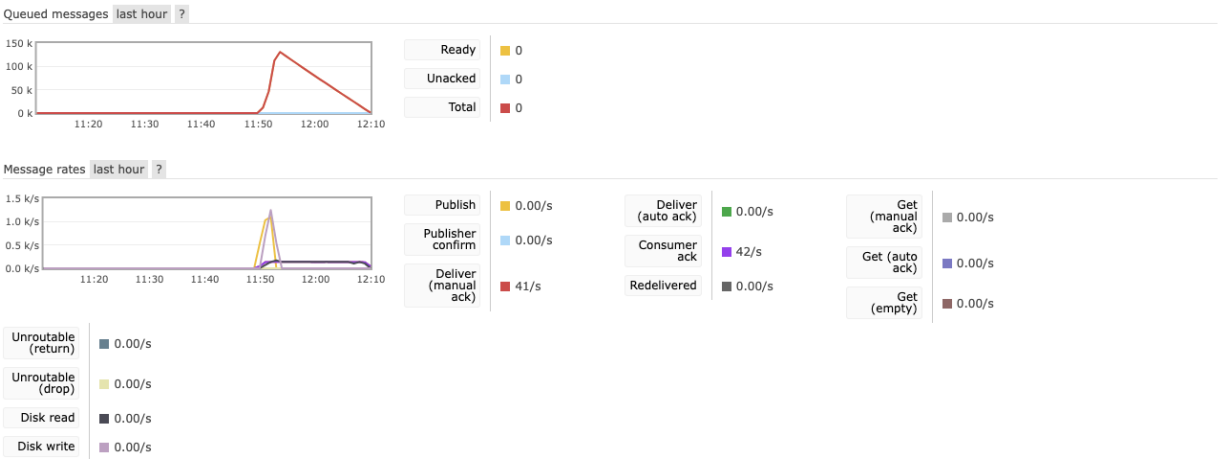
RMQ management windows for model2



We figure out that model1 has greater performance over model2, so I just pick up model1 as my final data model.

Results for step2

Before I apply any optimization, when I run 128 client threads the rabbitmq server message queue has heavy backlog.



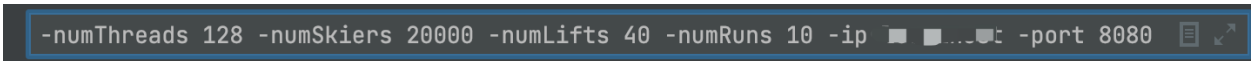
As we can see from the above graph, the consumer ack rate is below 300/s. I know writing data into RDS is the bottleneck for the consumer. Here I just use JDBC connecting MySQL with my application and each time receiving a message from message queue, the program will write the object down to database.

I know we can batch insert data into MySQL to significantly speed up our insertion. But the data comes sequentially, we never know the specific number of messages. Later, I use blockingqueue to store those messages got from message queue, and I start a new thread to constantly check whether the blockingqueue has any messages. Each time we get a batch_size of messages, I will use a separate thread to write those messages into database.

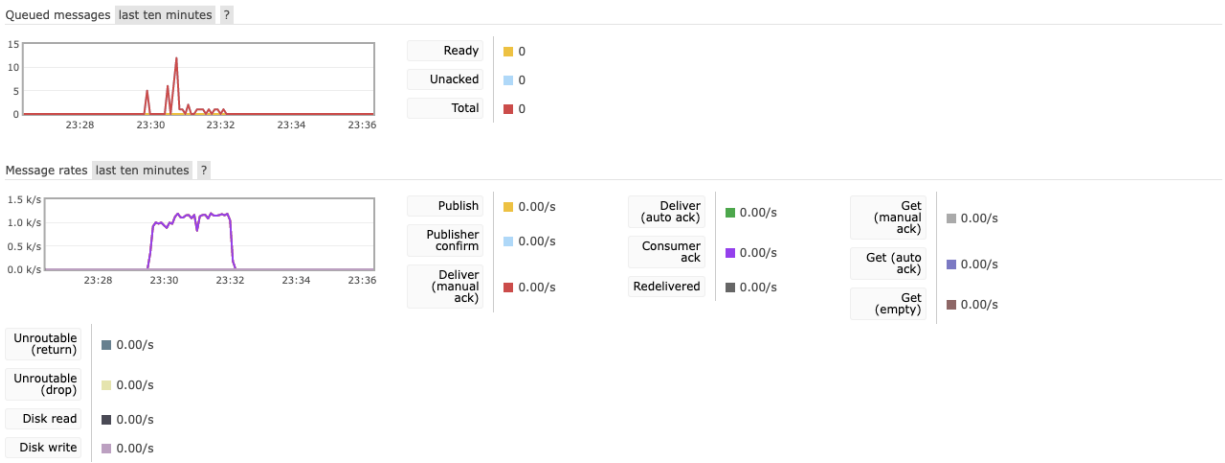
After applying the new solution, the length of queue is largely reduced.

128 client threads

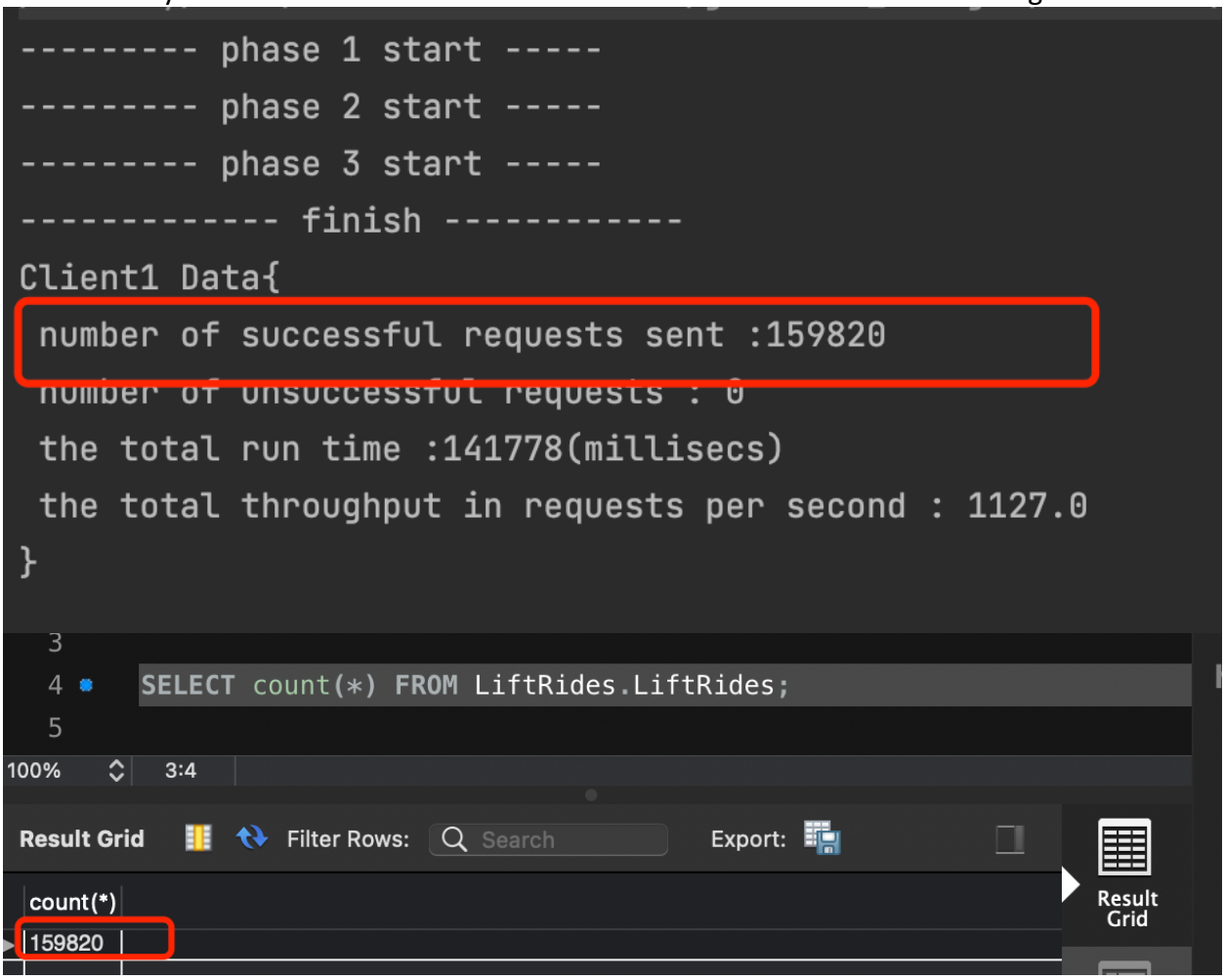
Command window:



RMQ management windows



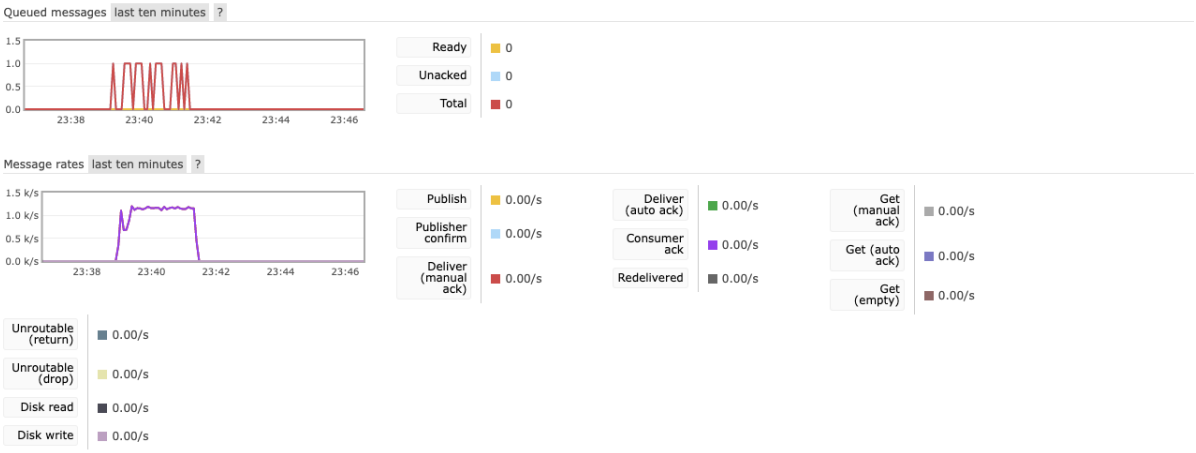
We can verify the number of records in RDS to check whether we write all messages into RDS.



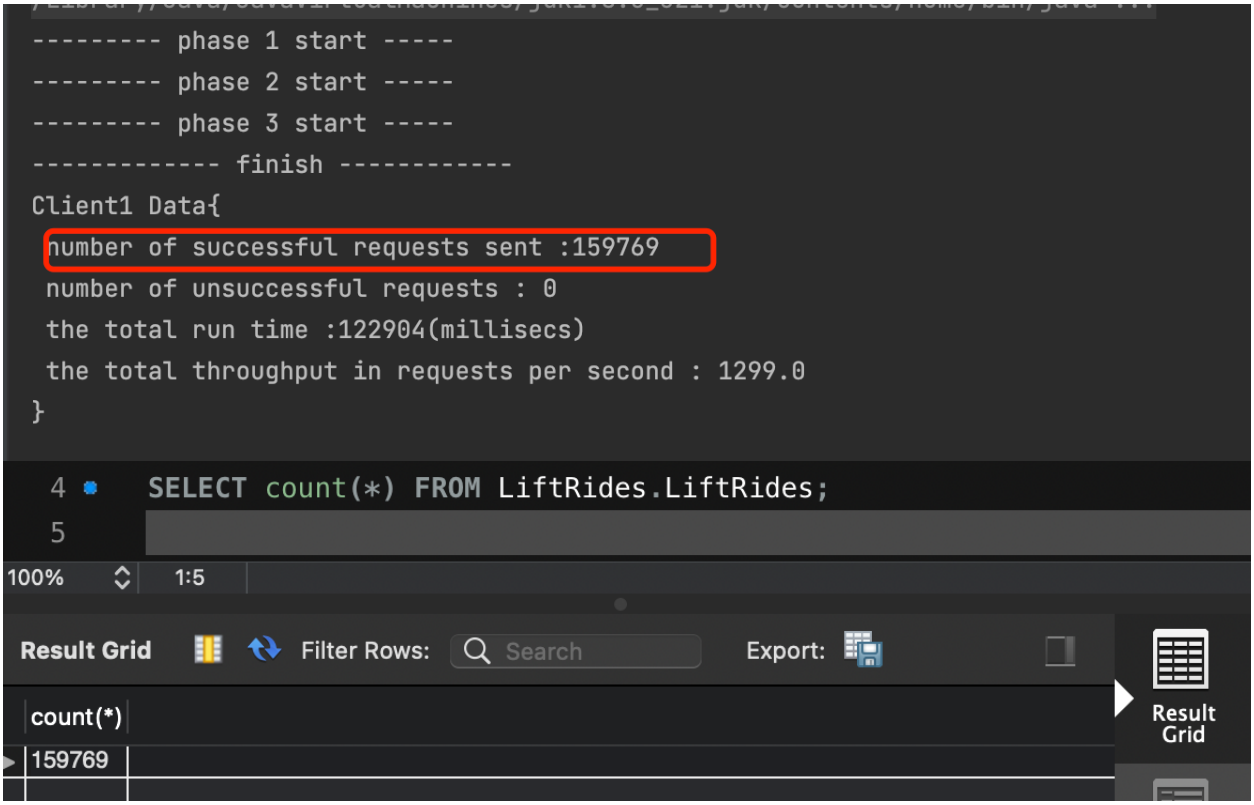
256 client threads

Command window:

RMQ management windows



We can verify the number of records in RDS to check whether we write all messages into RDS.

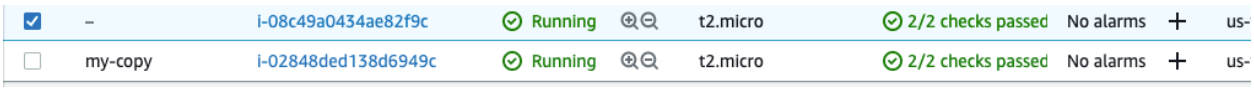


Results for step3

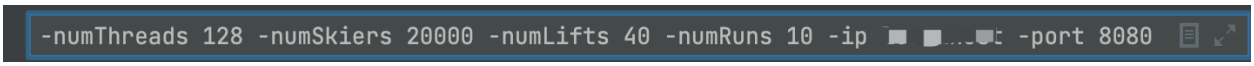
In step3, we need to produce a message, and two consumers should receive that. That is what exchange of type fanout can do for this scenario. I create a fanout exchange in server, and binding each queue with the exchange from different consumers.

The initial problem:
At the beginning, I ran the client with 128 threads. It worked fine for the first few seconds, but later on the rbbitmq management console just got stuck. It showed “we lost connection with rabbitmq server at xxxxx time” and I hardly connected to my ec2. After a few experientments, I realized that my ec2 CPU utilization reach up to nearly 99% which cause all applications in ec2 got stuck including rabbitmq server.

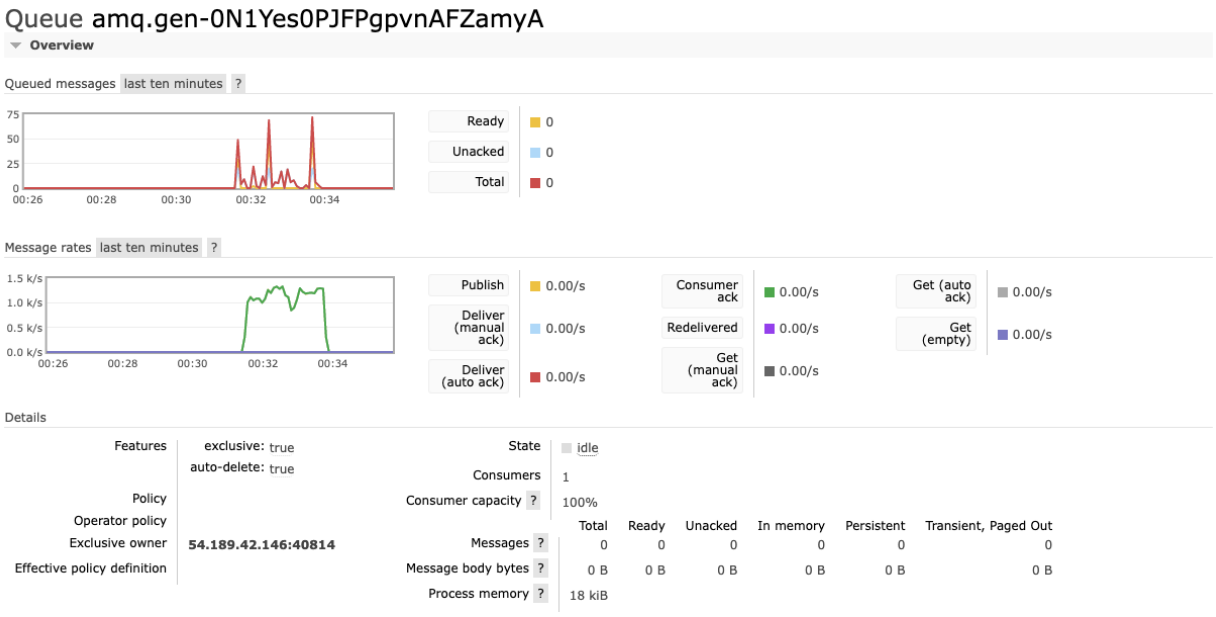
Solution:
My solution is create a new ec2 named my-copy, I deployed my consumer programs in this newly created ec2 so that my applications will not run out of my compute resource.



128 client threads
Command window:



RMQ management windows for queue1(the consumer1)



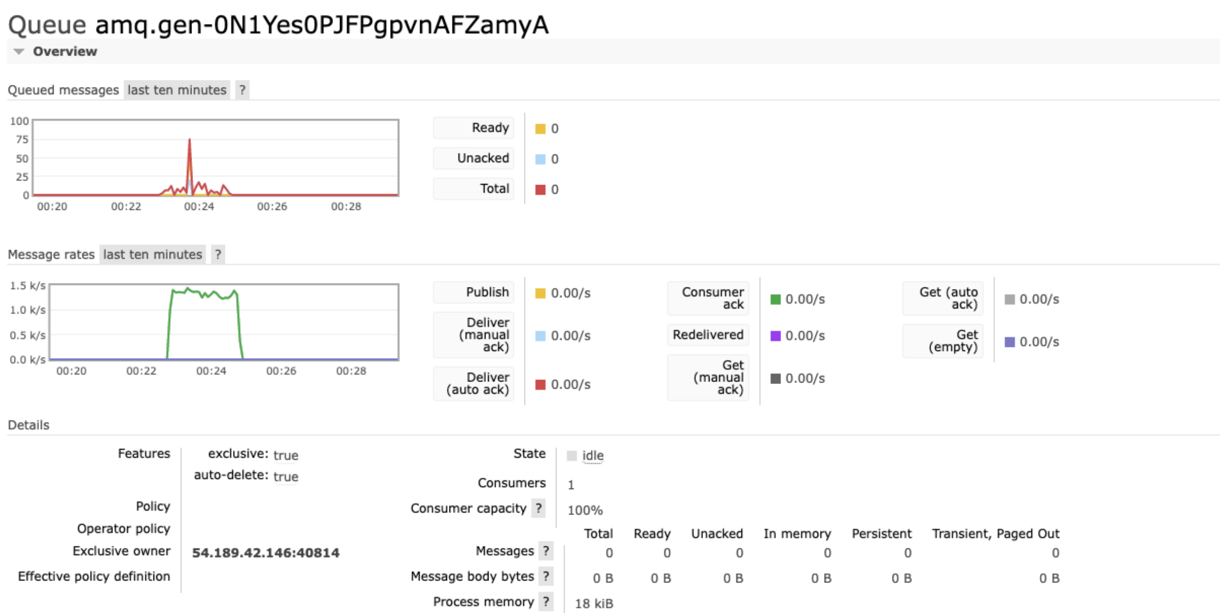
RMQ management windows for queue2(the consumer2)

256 client threads

Command window:

```
-numThreads 256 -numSkiers 20000 -numLifts 40 -numRuns 10 -ip [redacted]: -port 8080
```

RMQ management windows for queue1(the consumer1)



RMQ management windows for queue2(the consumer2)

Queue amq.gen-9T0zciolxsKRtUZpFVcuww

Overview

Queued messages last ten minutes ?

Ready	0
Unacked	0
Total	0

Message rates last ten minutes ?

Publish	0.00/s	Consumer ack	0.00/s	Get (auto ack)	0.00/s
Deliver (manual ack)	0.00/s	Redelivered	0.00/s	Get (empty)	0.00/s
Deliver (auto ack)	0.00/s	Get (manual ack)	0.00/s		

Details

Features	exclusive: true auto-delete: true
Policy	
Operator policy	
Exclusive owner	**54.189.42.146:40866**
Effective policy definition	

State	idle
Consumers	1
Consumer capacity ?	100%

	Total	Ready	Unacked	In memory	Persistent	Transient, Paged Out
Messages ?	0	0	0	0	0	0
Message body bytes ?	0 B	0 B	0 B	0 B	0 B	0 B
Process memory ?	15 kiB					