**Assignment 3**

CS6650 Fall 2022

Xiao Lan

**Github Repository URL**:

<https://github.com/elioxiaolan/Assignment03>

**Database Design**:

For data design, my original plan was to set the skier ID and resort ID as the key, and different values which contain the resort ID, season ID, day ID, skier ID, time, lift ID, and vertical. However, the difficulty with this schema was that I would have to create several Maps in Redis, which was expensive and inefficient for querying data from the database.

A better solution I came up with was to use simple keys with the values being a set of strings, and the strings elements in the set are serialized JSON strings. One benefit of this database design is that we do not have to know the exact design of how the keys and values are formed, which makes the queries ad hoc, and thus this design is more scalable than other alternatives. The JSON structures of values also reduce the cost of queries.

For **Consumer\_Resort**:

**Key**: resort ID

**Value**: JSON String which contains resort ID, season ID, day ID, skier ID, time, lift ID, and vertical

For **Consumer\_Skier**:

**Key**: skier ID

**Value**: JSON String which contains resort ID, season ID, day ID, skier ID, time, lift ID, and vertical

**AWS EC2 Specifications**:

**Server**: I used a t2.micro EC2 instance to distribute requests.

**Consumer**: I used three different sizes of EC2 instance, the first one was t2.micro instance, the second one was t2.medium instance, and the last was t3.medium instance.

**RabbitMQ**: I used a t2.micro EC2 instance.

**Redis**: I used a t2.micro EC2 instance.

**RabbitMQ Results**:

At first, I used a t2.micro EC2 instance for my Consumer, and I set the number of thread as 32 for Client, the number of thread as 128 for Consumer. As a result, there are many messages in the Queue during the whole process, the results are as followed (I used Consumer\_Resort as my Consumer):

**32 Threads for Client and 128 Threads for Consumer (t2.micro)**:

Graphical user interface, application

Description automatically generated

Text

Description automatically generated

It seems that my Consumer can’t process the messages and store them in my remote Redis server fast enough, so I considered increasing the size of my Consumer instance and then increasing the number of threads I spawn.

I employed a new size of EC2 instance as my Consumer EC2 instance, and change the number of threads from 128 to 256 in Consumer, the test results are as followed:

**32 Threads for Client and 128 Threads for Consumer (t2.medium)**:

Graphical user interface, chart

Description automatically generated

Text

Description automatically generated

**32 Threads for Client and 256 Threads for Consumer (t2.medium)**:

Graphical user interface, application

Description automatically generated

Text

Description automatically generated

From the results we can conclude that a larger size of EC2 instance can increase the process rate of Consumer but has no effect on the throughput.

The next step I did was to employed a larger EC2 instance for the Consumer, so I used a t3.medium instance for the Consumer, and the number of threads of Consumer remain the same as previous, but I change the number of threads of Client, and I see an increase in throughput with an increase in the number of messages in the queue:

Graphical user interface, chart

Description automatically generated

Text

Description automatically generated

The increase in throughput can’t offset the increase in the number of messages in queue, which means there are some room of optimization for Consumer part.