**Assignment 2**

CS6650 Fall 2022

Xiao Lan

**Github Repository URL**:

<https://github.com/elioxiaolan/Assignment02>

**Server Design**:

The most functionalities of Server will be implemented in the class **SkierServlet**:

The class **SkierServlet** inherits the most part of one in Assignment 1. When the servlet is initialized, it will initialize a RabbitMQ channel pool. After verifying the URL, the Server will record the parameters in URL, and then use a shared and synchronized RabbitMQ channel pool, which is implemented by **BlockingQueue**, and then threads will publish messages, and send corresponding response back.

**Packages**:

The packages used in the Server include Swagger client, the JavaX Servlet packages, and RabbitMQ AMQP Client packages.

**Consumer Design**:

The most functionalities of Consumer will be implemented in the class **Consumer**:

In the class Consumer, each thread will declare a channel to the message queue, use the push model of message consumption, and provides a callback method to the message broker. Besides, the consumer will record the SkierId and LiftId into **ConcureentHashMap**. Once the Consumer complete all processing, the Consumer will acknowledges the message.

**Packages**:

The packages used in the Server include Swagger client, the JavaX Servlet packages, RabbitMQ AMQP Client packages, and Gson package.

**Test Results**:

**Single Instance Results:**

In single EC2 instance, I use a local client with 32, 64, and 128 threads to send requests to test the performance, the results are as followed:

**32 Threads**:

Chart, line chart

Description automatically generated

Chart, line chart

Description automatically generated

Text

Description automatically generated

**64 Threads**:

Application

Description automatically generated with medium confidence

Chart, line chart

Description automatically generated

Text

Description automatically generated

**128 Threads**:

A picture containing application

Description automatically generated

Chart, line chart

Description automatically generated

Text

Description automatically generated

**Load Balancer Results**:

To build a load balancer, I created an AMI from my initial server instance. When the AMI is created, I launched two new instances from this AMI, and they are exact replicas of my initial instance that hosts your servlet. When all instances have been launched, I placed them into an application load balancer, and the test results are as followed:

**32 Threads**:

Graphical user interface

Description automatically generated

Text

Description automatically generated

**64 Threads**:

Graphical user interface, application

Description automatically generated

Text

Description automatically generated

**128 Threads**:

Graphical user interface, application

Description automatically generated

Text

Description automatically generated