A. GitHub: https://github.com/jiehuizhou66/cs6650

B. Considerations & Overall Design:

In this assignment, my approach is to deploy multiple MySQL instances and shard the lift rides data across instances.

In the previous assignment, I implemented the RabbitMQ(producer consumer based queueing solution) which significantly increased the throughput. However, the improvement in A3 is because that the servlet only pass the request to the producer and send back a success response. And the producer and consumer will handle the rest of the process (which is writing request to DB). By implementing this asynchronous server I cannot see how much does the rest of the process (from producer to database) being improved?

Based on what I observed when building and testing database in A2, the failed requests were due to database exception. The exception was complaining that the connection to the database was rejected because there were too many connections. Thus I decided to build the database improvement (shard table and run multiple MySQL instance) on top of A2's implementation in order to see how much improvement does this approach can provide.

C. Classes:

1. SkierServlet.java:

It is the main class which receive and respond to the request from Skiers API. It performs the necessary validation(ie. URL, parameters). If all the validation passed, It will call LiftRideDao to write/read from database. The corresponding response will then be returned to the client.

2. LiftRideDao.java

The main functionalities of this class is to write/read from database. The Biggest change of LiftRideDao in the A4 is to shard the data into 4 different database tables based on the skierID for both POST and GET request. Because all the types of Skier GET requests have the skierID parameter, so we can determine which database table it should read based on the skierID passed in.

1. createLiftRide() check the skierID and write the LiftRide entry in the designated database table. It write one row in database every time this method gets invoked.
2. queryLiftRideVerticalPerDay() finds the corresponded LiftRide table based on skierID. Quey the table using index which is the combination of skierID and dayID.
3. queryLiftRideVerticalPerResort() finds the corresponded LiftRide table based on skierID. Query the table using index which is the combination of skierID and resortID.

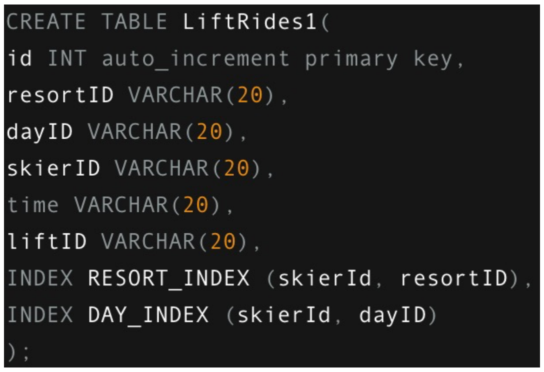
3. DBCPDataSource.java

It is the connection manager class between my program and MySQL database. I created 2 MySQL databases in this class.

4. LiftRide.java

It is a POJO class which stores ResortID, DayID, SkierID, Time, LiftID.

5. Schema:



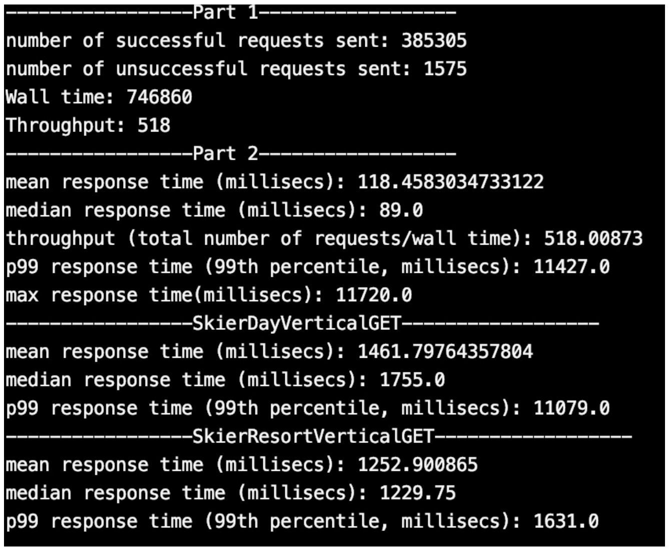
```sql
CREATE TABLE LiftRides1(
id INT auto_increment primary key,
resortID VARCHAR(20),
dayID VARCHAR(20),
skierID VARCHAR(20),
time VARCHAR(20),
liftID VARCHAR(20),
INDEX RESORT_INDEX (skierId, resortID),
INDEX DAY_INDEX (skierId, dayID)
);
```

RESORT_INDEX is used for the GET request to get the total vertical for the skier for the specified resort.

DAY_INDEX is used for the GET request to get the total vertical for the skier for the specified ski day.

D. Result Comparison:

1. Single server 256 threads A2 vs A4.
    1.1. Screen shot of A4:



```
------------------Part 1------------------
number of successful requests sent: 385305
number of unsuccessful requests sent: 1575
Wall time: 746860
Throughput: 518
------------------Part 2------------------
mean response time (millisecs): 118.4583034733122
median response time (millisecs): 89.0
throughput (total number of requests/wall time): 518.00873
p99 response time (99th percentile, millisecs): 11427.0
max response time(millisecs): 11720.0
------------------SkierDayVerticalGET------------------
mean response time (millisecs): 1461.79764357804
median response time (millisecs): 1755.0
p99 response time (99th percentile, millisecs): 11079.0
------------------SkierResortVerticalGET------------------
mean response time (millisecs): 1252.900865
median response time (millisecs): 1229.75
p99 response time (99th percentile, millisecs): 1631.0
```

1.2. Table:

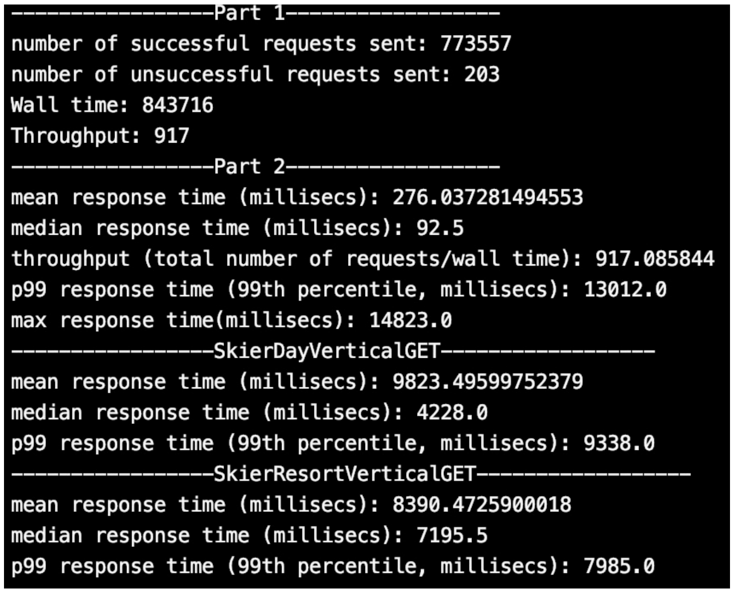
single server	assignment & thread #	wall time(s)	throughput	median response for all request (ms)
	A2 256 thread	1341.607	288	106
	A4 256 thread	746.86	518	89

1.3 Observations:

As we can see in the result of single server 256 threads test run, the throughput is significantly improved after shard the data into multiple database tables.

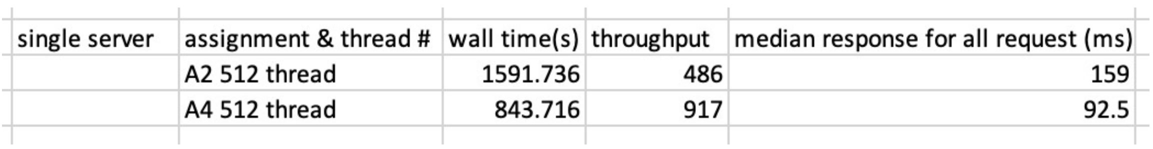
2. Single server 512 threads A2 vs A4.

2.1 Screenshot of A4:



```
------------------Part 1------------------
number of successful requests sent: 773557
number of unsuccessful requests sent: 203
Wall time: 843716
Throughput: 917
------------------Part 2------------------
mean response time (millisecs): 276.037281494553
median response time (millisecs): 92.5
throughput (total number of requests/wall time): 917.085844
p99 response time (99th percentile, millisecs): 13012.0
max response time(millisecs): 14823.0
------------------SkierDayVerticalGET------------------
mean response time (millisecs): 9823.49599752379
median response time (millisecs): 4228.0
p99 response time (99th percentile, millisecs): 9338.0
------------------SkierResortVerticalGET------------------
mean response time (millisecs): 8390.4725900018
median response time (millisecs): 7195.5
p99 response time (99th percentile, millisecs): 7985.0
```

2.2 Table:



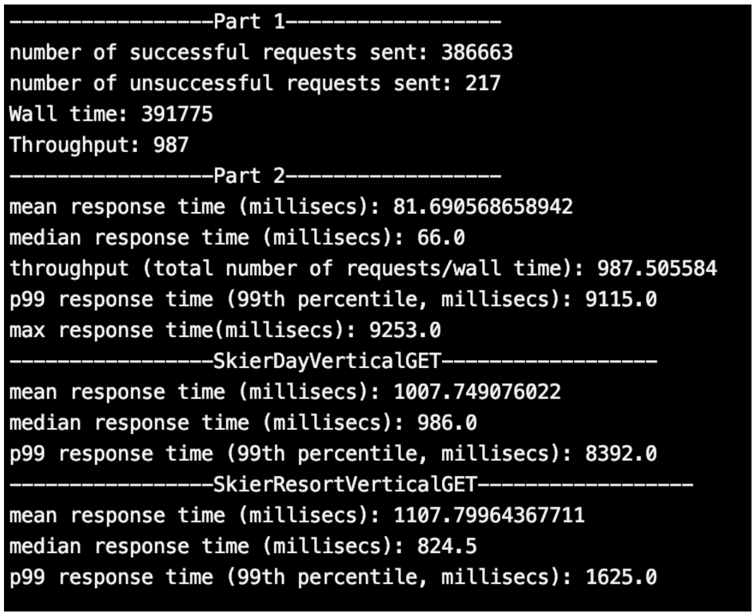
single server	assignment & thread #	wall time(s)	throughput	median response for all request (ms)
	A2 512 thread	1591.736	486	159
	A4 512 thread	843.716	917	92.5

2.3 Observation:

Similar as in the 256 threads test result, the throughput is also increased significantly in the result of 512 thread single server test run.

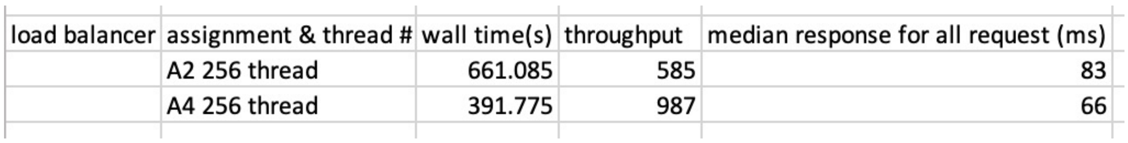
3. Load balancer 256 threads A2 vs A4.

3.1 Screenshot of A4.



```
-----------------Part 1------------------
number of successful requests sent: 386663
number of unsuccessful requests sent: 217
Wall time: 391775
Throughput: 987
-----------------Part 2------------------
mean response time (millisecs): 81.690568658942
median response time (millisecs): 66.0
throughput (total number of requests/wall time): 987.505584
p99 response time (99th percentile, millisecs): 9115.0
max response time(millisecs): 9253.0
-----------------SkierDayVerticalGET------------------
mean response time (millisecs): 1007.749076022
median response time (millisecs): 986.0
p99 response time (99th percentile, millisecs): 8392.0
-----------------SkierResortVerticalGET------------------
mean response time (millisecs): 1107.79964367711
median response time (millisecs): 824.5
p99 response time (99th percentile, millisecs): 1625.0
```

3.2 Table



load balancer	assignment & thread #	wall time(s)	throughput	median response for all request (ms)
	A2 256 thread	661.085	585	83
	A4 256 thread	391.775	987	66

3.3 Observation

The throughput is also increased in the result of 256 threads of load balancer test run compare to the result in A2. Similar pattern happened in table 1.2 and 2.2 as well. It indicates that there was a bottleneck in the database. Shard the lift rides data into multiple database table and run different table on different MySQL instances removed the bottleneck when writing and reading from database.

In order to further remove the bottleneck of the database, adding a replica for GET request or try some other database can be a direction to try and test. Due to the time limitation, I did not do those further improvements.

4. Comparison between A3 and A4

Because A4's main implementation is to shard the database table to check if it can remove the bottleneck of database. And in order to see the result, I did A4's implementation based on A2. So I think comparison between A3 and A4 is not necessary. A3 is to add RabbitMQ to remove the bottleneck between client and server. And A4 is to shard the data to remove the bottleneck of database.

E. Summary

In A3, I see the throughput between client and server is increased. In A4, I see the throughput between server and database is improve. In summary, I think the combination of using producer consumer based queueing solution and shard the database table can provide a better performance for this Skier service distributed system.