Assignment 2

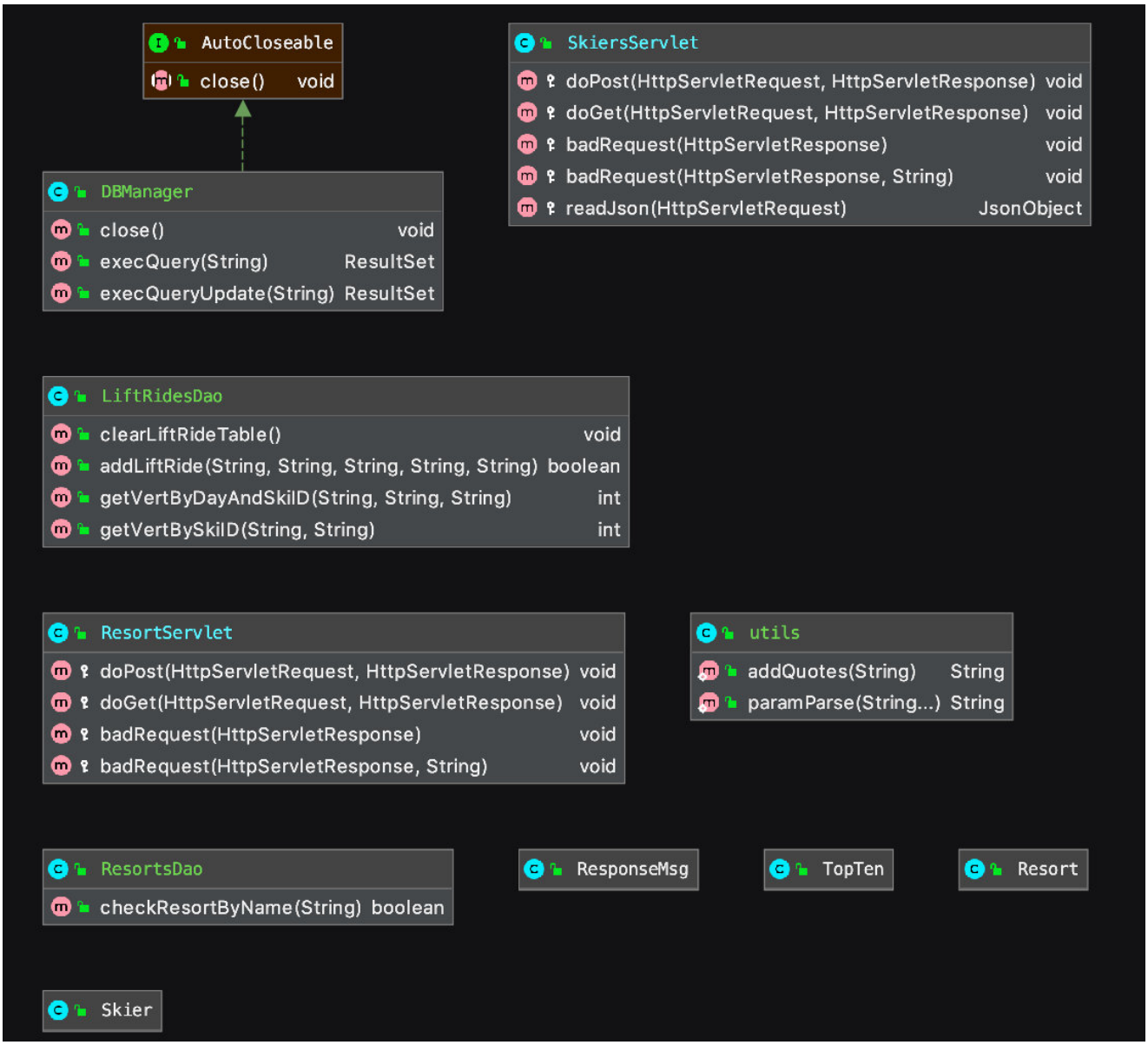
Github Repo

https://github.com/duskcloudxu/bsds2020fall_Assignment

Load Balancer Address

```
http://assignment2-2130851594.us-east-1.elb.amazonaws.com//remoteServer_war/
```

Project Structure



Overview

- **DBManager**
  - DB use JDBC to create databse connection and return query result as `ResultSet`
- **ResortDao, LiftRideDao**
  - Class interact with database using database manager, **ResortDao** implement function for `/resort` API and **LiftRideDao** implement function regarding `/skier` API
- **ResortServlet, SkierServlet**
  - Response corresponding URL based on the service provided by class in package `dao`

Workflow

- Request would receive by the corresponding servlet, and servlet would call the related function in dao layer to insert data into database or query data from database
  - Data would be returned in the form of `ResultSet` class and we use `GSON` to parse the result and transfer it into JSON format, and forward it into response and return.

Update from last assignment

I replaced the while loop with CountdownLatch and add unit in wall time as suggested. It's definitely an improvement in design yet does not help much on the total wall time(accelerated around 5-6 seconds in wall time).

I've Due to the time limit, I have to submit this assignment though I am not satisfied with the current performance. A guess is that it might caused by my network and it could help if I could run this project on the EC2 instance. *(I asked my friend to query on my server and he got a far better performance using client in his local environment, and I used his client on my server in my local environment but only got the same bad performance as my client.)* The blocker for that idea is that I could not create executable jar file using maven after hours research on that. Maybe I would switch to Gradle in next assignment.

Performance with single server

numThread\metric	Mean Post Latency(ms)	Mean Get Latency(ms)	Median Post Latency(ms)	Median Get Latency(ms)	Wall Time(Sec)	Throughput(per Sec)	P99 of Post	P99 of Get	Max Response of Post	Max Response of Get
32	258	187	252	169	425	113	447	337	5126	383
64	307	287	336	270	726	133	520	628	2644	834
128	564	757	618	519	1230	157	1168	2123	2649	2208
256	1151	1506	1222	961	2341	165	2594	4472	11032	6369

32 threads

```
mean latency for POST
258ms
mean latency for GET
187ms
median latency for POST
252ms
median latency for GET
169ms
Wall Time:
425S
Throughput:
113
P99 for POST
447ms
P99 for GET
377ms
Max Latency for POST
5126ms
Max Latency for GET
383ms

Process finished with exit code 0
```

64threads

```
mean latency for POST
307ms
mean latency for GET
287ms
median latency for POST
336ms
median latency for GET
270ms
Wall Time:
726S
Throughput:
133
P99 for POST
520ms
P99 for GET
628ms
Max Latency for POST
2644ms
Max Latency for GET
834ms

Process finished with exit code 0
```

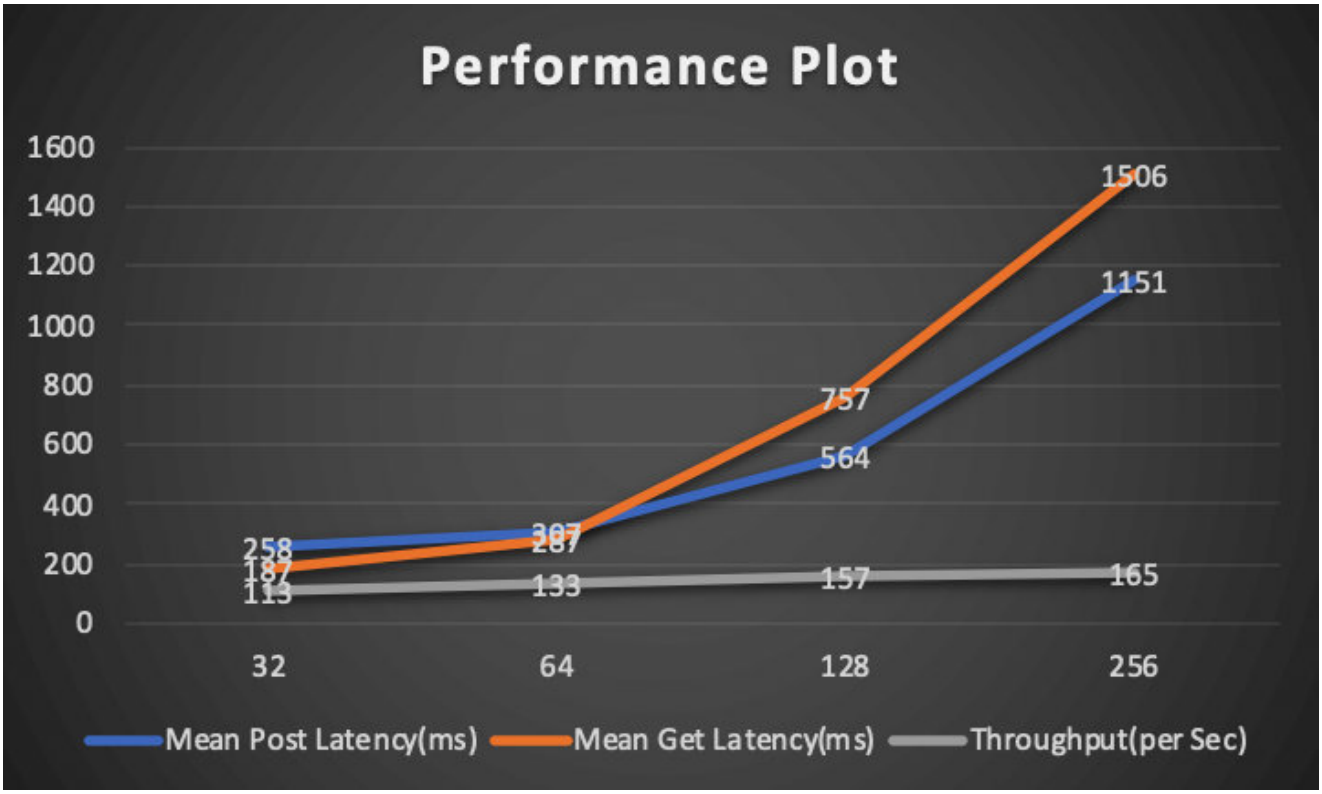
128 threads

```
mean latency for POST
564ms
mean latency for GET
757ms
median latency for POST
618ms
median latency for GET
519ms
Wall Time:
1230S
Throughput:
157
P99 for POST
1168ms
P99 for GET
2123ms
Max Latency for POST
2649ms
Max Latency for GET
2208ms
```

256threads

```
mean latency for POST
1151ms
mean latency for GET
1506ms
median latency for POST
1222ms
median latency for GET
961ms
Wall Time:
2341S
Throughput:
165
P99 for POST
2594ms
P99 for GET
4472ms
Max Latency for POST
11032ms
Max Latency for GET
6369ms
```

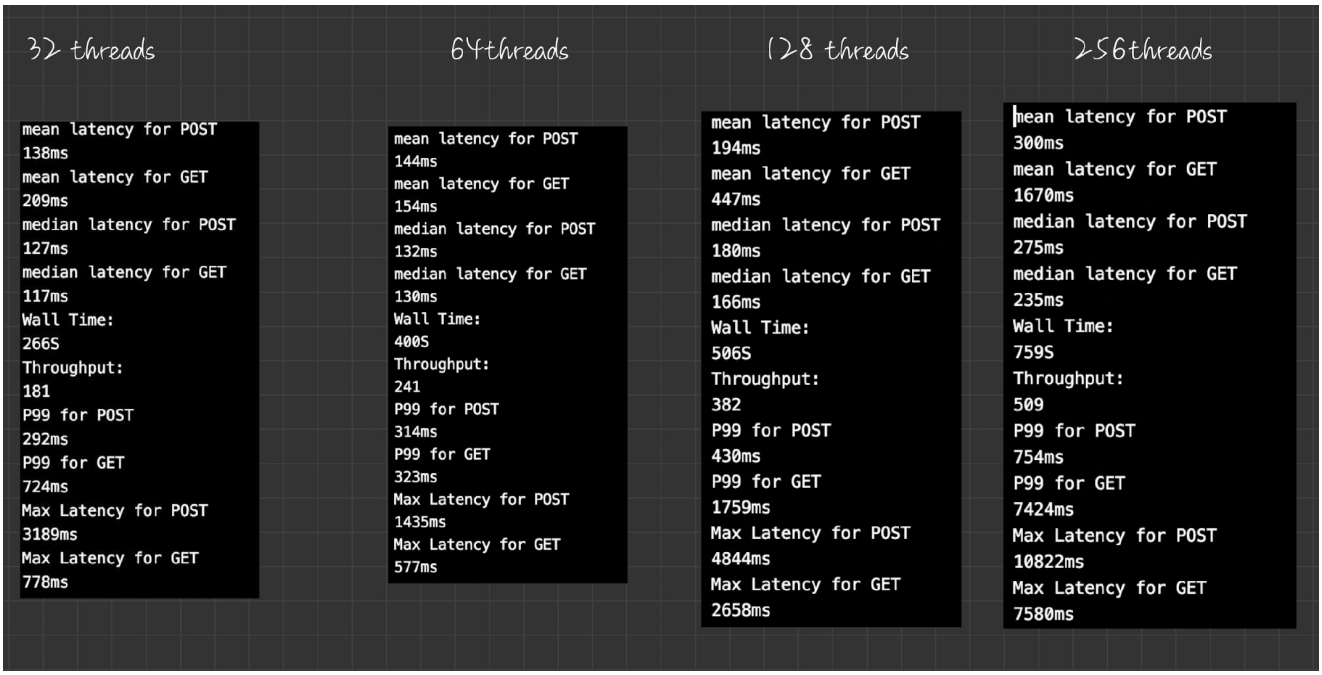
Plot of throughput, meanResponseTimeGet and mean ResponseTimePost against number of threads

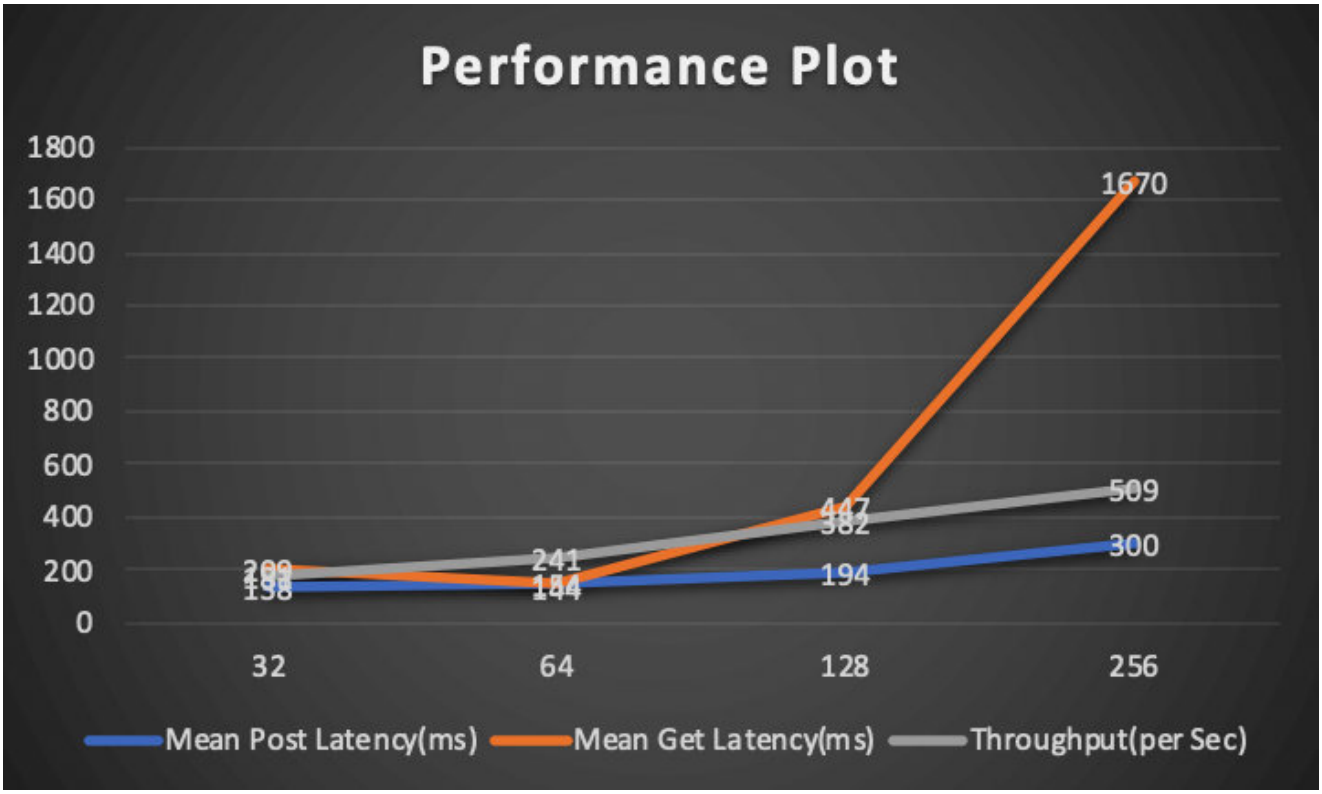


Performance with load balancer

In this section, I added 4 free-tier EC2 instance, and there is significant improvement in server performance.

numThread\metric	Mean Post Latency(ms)	Mean Get Latency(ms)	Median Post Latency(ms)	Median Get Latency(ms)	Wall Time(Sec)	Throughput(per Sec)	P99 of Post	P99 of Get	Max Response of Post	Max Response of Get
32	138	209	127	117	266	181	292	724	3189	778
64	144	154	132	130	400	241	314	323	1435	778
128	194	447	180	166	506	382	430	1759	4844	2658
256	300	1670	275	235	759	509	754	7424	10822	7580





Bonus Point

- I tried to run 512 threads client with 4 EC2-instance-cluster, and the bandwidth would be too large as there are many timeout requests. I made a horizontal scaling by adding another 4 ec2 instance into the cluster, and it works better now, in the end the static for 512 threads client as below:

```
mean latency for POST
356ms
mean latency for GET
3765ms
median latency for POST
329ms
median latency for GET
247ms
Wall Time:
976S
Throughput:
792
P99 for POST
874ms
P99 for GET
30016ms
Max Latency for POST
13182ms
Max Latency for GET
30154ms
```

P.S.

In order to prevent costing too much aws credits, I will shutdown all the extra ec2 instance in the cluster after I submit this report, so please let me know if you want to test my server performance.