CS 6650 Assignment 2 Report

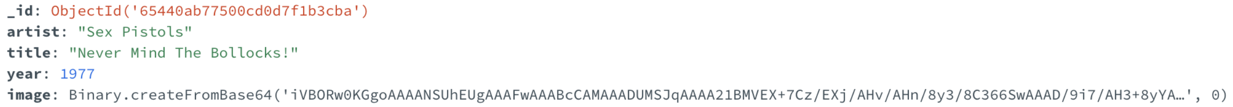
Tzu-Yu Huang

1 - Github Repo

https://github.com/tyhuang06/CS6650-assignments/tree/main/a2

2 - Data Model

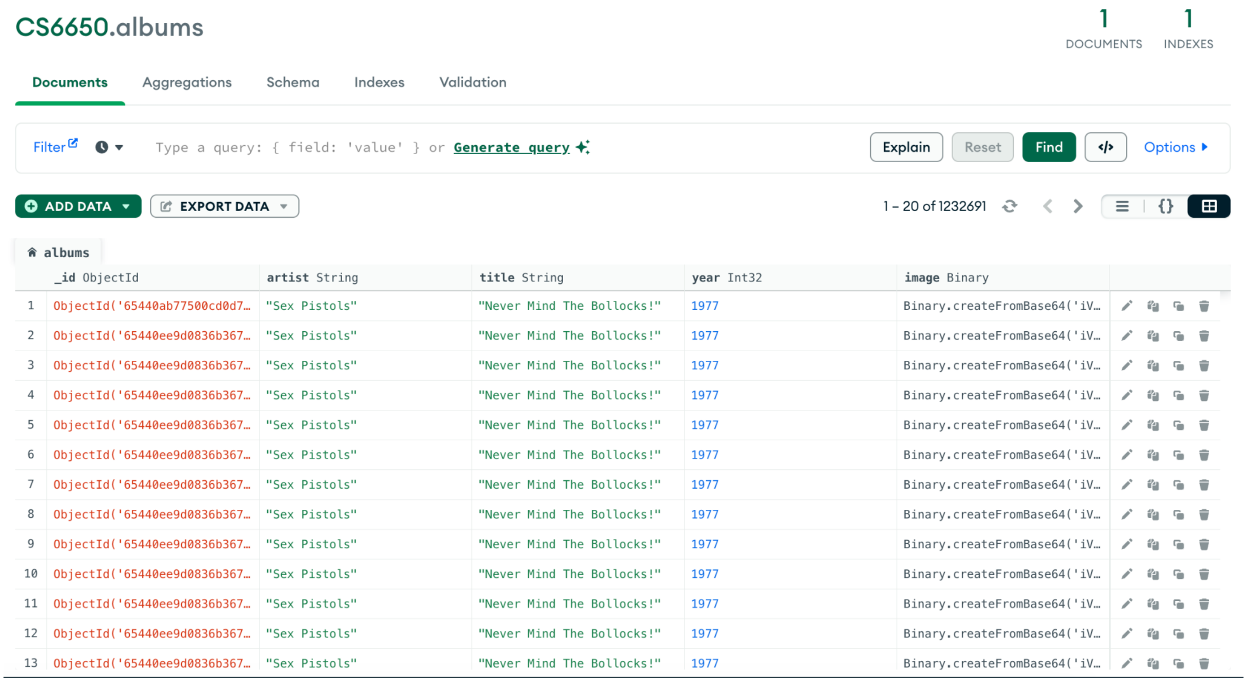
I chose MongoDB as the database for this project, the data model is quite straightforward, each object contains a unique id, profile of the album (title, artist, year) and the album image.



```
_id: ObjectId('65440ab77500cd0d7f1b3cba')
artist: "Sex Pistols"
title: "Never Mind The Bollocks!"
year: 1977
image: Binary.createFromBase64('iVBORw0KGgoAAAANSUhEUgAAAFwAAABcCAMAAADUMSJqAAAA21BMVEX+7Cz/EXj/AHv/AHn/8y3/8C366SwAAAD/9i7/AH3+8yYA…', 0)
```

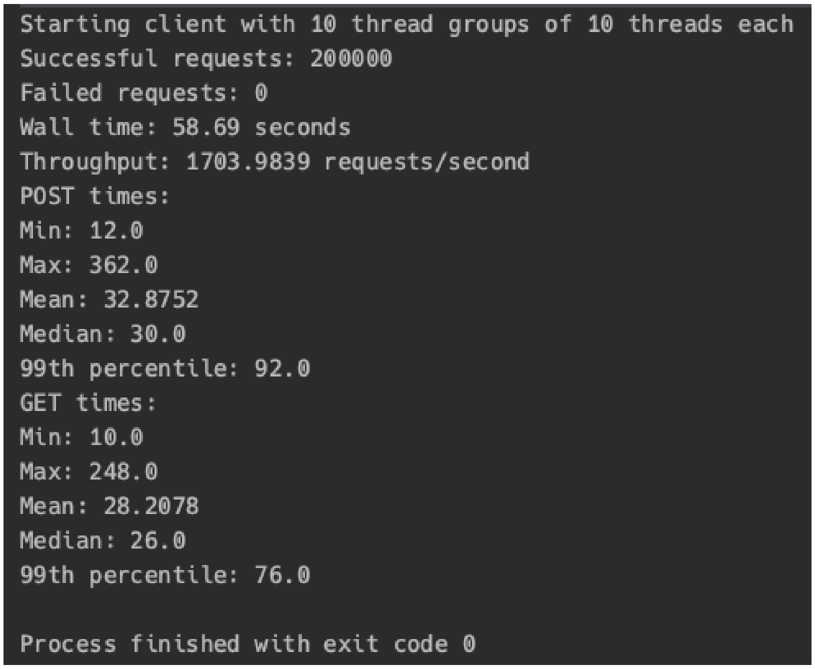
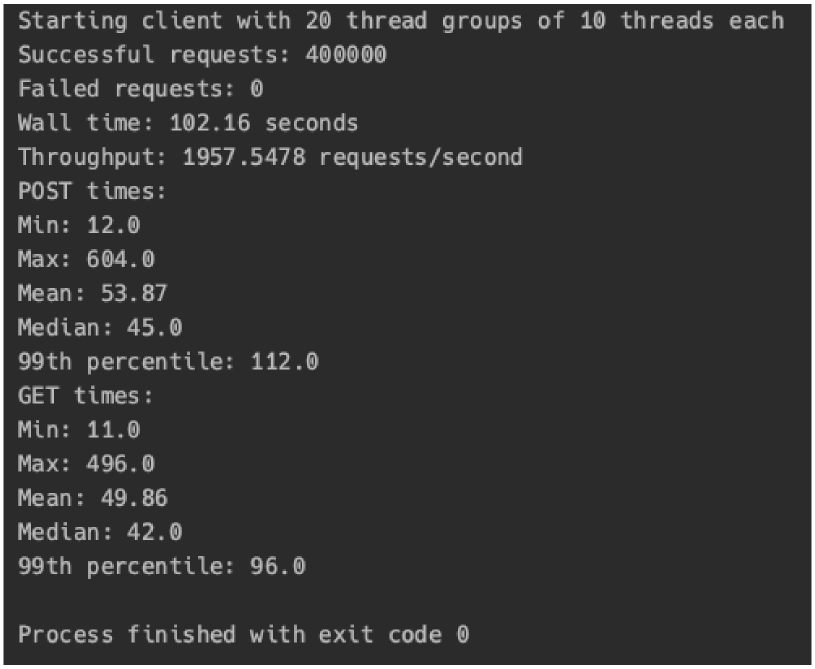
At first I used MongoDB Atlas to host my database, but the connection to Atlas has a high latency, so I was struggling with the performance. To fix this problem, I installed MongoDB “locally” in my EC2 instance instead.

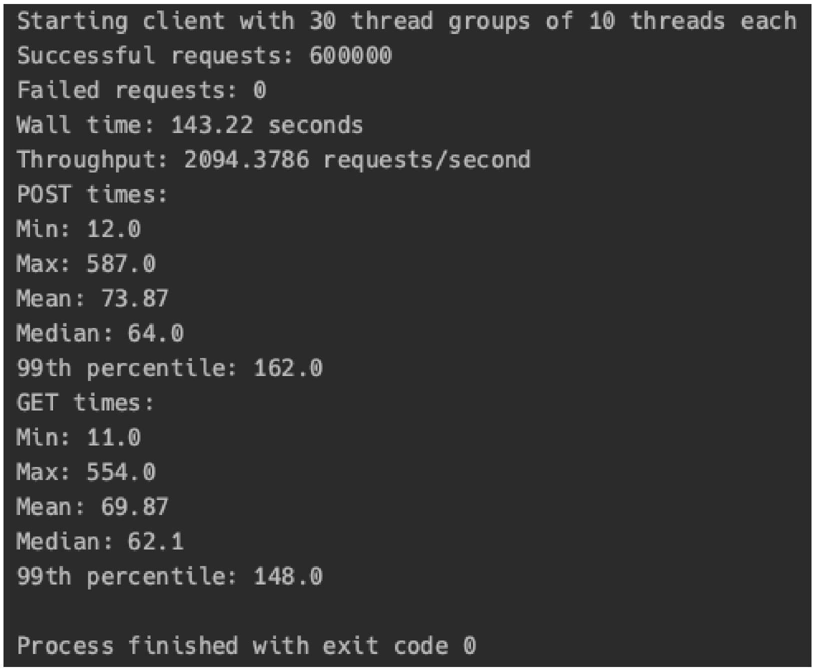
Screenshot of DB after doing several tests:



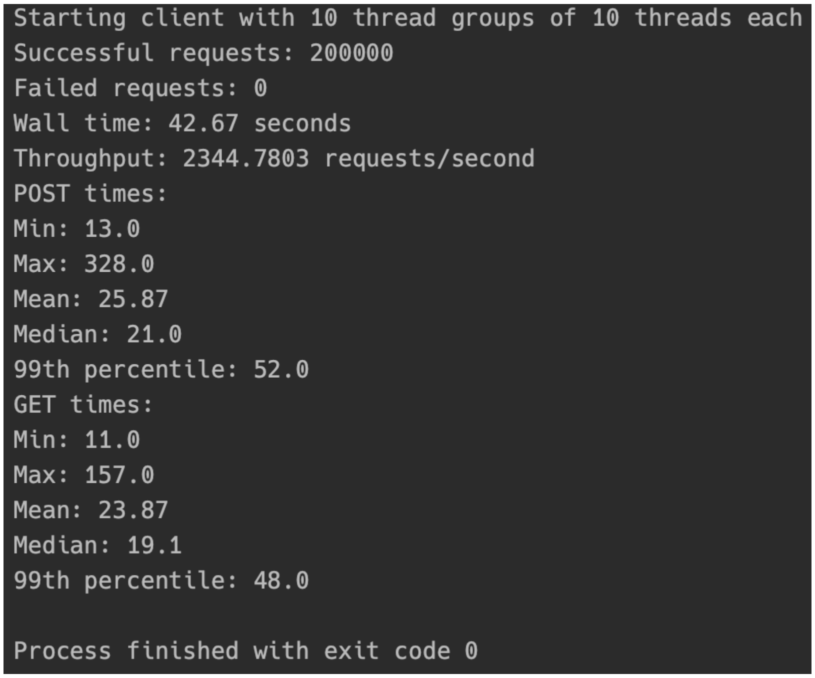
	_id ObjectId	artist String	title String	year Int32	image Binary
1	ObjectId('65440ab77500cd0d7…	"Sex Pistols"	"Never Mind The Bollocks!"	1977	Binary.createFromBase64('iV…
2	ObjectId('65440ee9d0836b367…	"Sex Pistols"	"Never Mind The Bollocks!"	1977	Binary.createFromBase64('iV…
3	ObjectId('65440ee9d0836b367…	"Sex Pistols"	"Never Mind The Bollocks!"	1977	Binary.createFromBase64('iV…
4	ObjectId('65440ee9d0836b367…	"Sex Pistols"	"Never Mind The Bollocks!"	1977	Binary.createFromBase64('iV…
5	ObjectId('65440ee9d0836b367…	"Sex Pistols"	"Never Mind The Bollocks!"	1977	Binary.createFromBase64('iV…
6	ObjectId('65440ee9d0836b367…	"Sex Pistols"	"Never Mind The Bollocks!"	1977	Binary.createFromBase64('iV…
7	ObjectId('65440ee9d0836b367…	"Sex Pistols"	"Never Mind The Bollocks!"	1977	Binary.createFromBase64('iV…
8	ObjectId('65440ee9d0836b367…	"Sex Pistols"	"Never Mind The Bollocks!"	1977	Binary.createFromBase64('iV…
9	ObjectId('65440ee9d0836b367…	"Sex Pistols"	"Never Mind The Bollocks!"	1977	Binary.createFromBase64('iV…
10	ObjectId('65440ee9d0836b367…	"Sex Pistols"	"Never Mind The Bollocks!"	1977	Binary.createFromBase64('iV…
11	ObjectId('65440ee9d0836b367…	"Sex Pistols"	"Never Mind The Bollocks!"	1977	Binary.createFromBase64('iV…
12	ObjectId('65440ee9d0836b367…	"Sex Pistols"	"Never Mind The Bollocks!"	1977	Binary.createFromBase64('iV…
13	ObjectId('65440ee9d0836b367…	"Sex Pistols"	"Never Mind The Bollocks!"	1977	Binary.createFromBase64('iV…

3 - Output: Single Server/DB

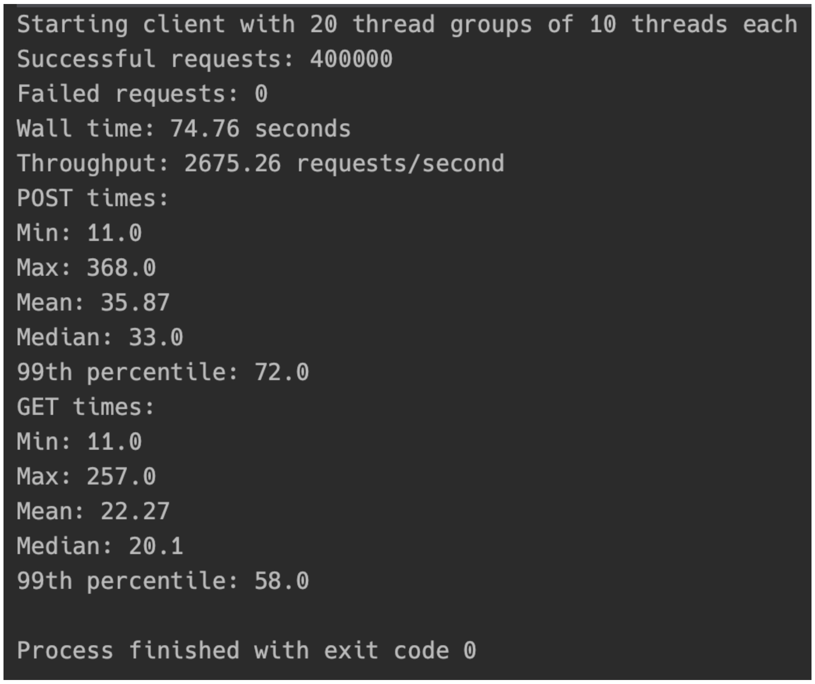
Thread Groups	Output Window
10	 Starting client with 10 thread groups of 10 threads each Successful requests: 200000 Failed requests: 0 Wall time: 58.69 seconds Throughput: 1703.9839 requests/second POST times: Min: 12.0 Max: 362.0 Mean: 32.8752 Median: 30.0 99th percentile: 92.0 GET times: Min: 10.0 Max: 248.0 Mean: 28.2078 Median: 26.0 99th percentile: 76.0 Process finished with exit code 0
20	 Starting client with 20 thread groups of 10 threads each Successful requests: 400000 Failed requests: 0 Wall time: 102.16 seconds Throughput: 1957.5478 requests/second POST times: Min: 12.0 Max: 604.0 Mean: 53.87 Median: 45.0 99th percentile: 112.0 GET times: Min: 11.0 Max: 496.0 Mean: 49.86 Median: 42.0 99th percentile: 96.0 Process finished with exit code 0

30	 Starting client with 30 thread groups of 10 threads each Successful requests: 600000 Failed requests: 0 Wall time: 143.22 seconds Throughput: 2094.3786 requests/second POST times: Min: 12.0 Max: 587.0 Mean: 73.87 Median: 64.0 99th percentile: 162.0 GET times: Min: 11.0 Max: 554.0 Mean: 69.87 Median: 62.1 99th percentile: 148.0 Process finished with exit code 0

4 - Output: Two Load Balanced Server/DB

Thread Groups	Output Window
10	 Starting client with 10 thread groups of 10 threads each Successful requests: 200000 Failed requests: 0 Wall time: 42.67 seconds Throughput: 2344.7803 requests/second POST times: Min: 13.0 Max: 328.0 Mean: 25.87 Median: 21.0 99th percentile: 52.0 GET times: Min: 11.0 Max: 157.0 Mean: 23.87 Median: 19.1 99th percentile: 48.0 Process finished with exit code 0

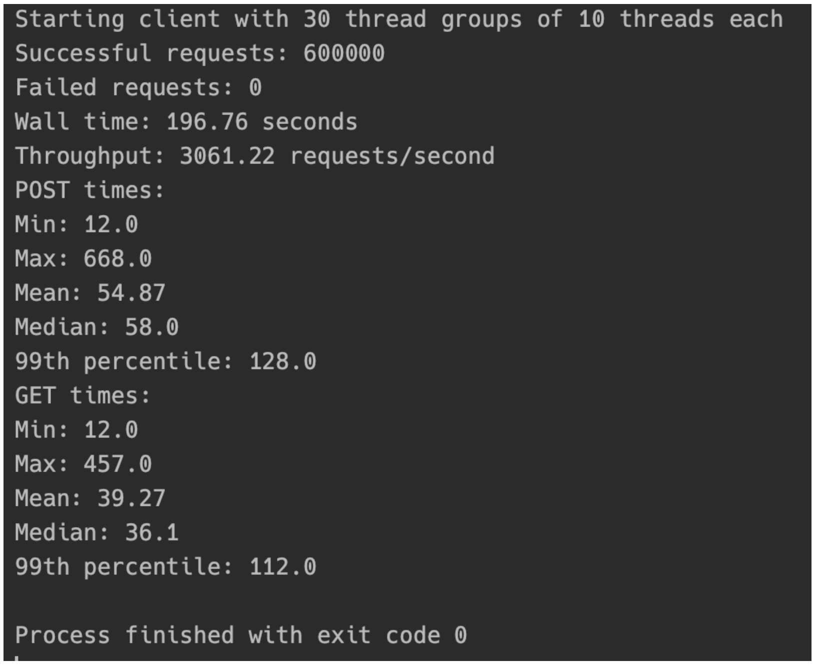
<table>
<tr>
<td>20</td>
<td>



```
Starting client with 20 thread groups of 10 threads each
Successful requests: 400000
Failed requests: 0
Wall time: 74.76 seconds
Throughput: 2675.26 requests/second
POST times:
Min: 11.0
Max: 368.0
Mean: 35.87
Median: 33.0
99th percentile: 72.0
GET times:
Min: 11.0
Max: 257.0
Mean: 22.27
Median: 20.1
99th percentile: 58.0

Process finished with exit code 0
```

</td>
</tr>
<tr>
<td>30</td>
<td>



```
Starting client with 30 thread groups of 10 threads each
Successful requests: 600000
Failed requests: 0
Wall time: 196.76 seconds
Throughput: 3061.22 requests/second
POST times:
Min: 12.0
Max: 668.0
Mean: 54.87
Median: 58.0
99th percentile: 128.0
GET times:
Min: 12.0
Max: 457.0
Mean: 39.27
Median: 36.1
99th percentile: 112.0

Process finished with exit code 0
```

</td>
</tr>
</table>

5 - Output: 30 Thread Groups Optimized

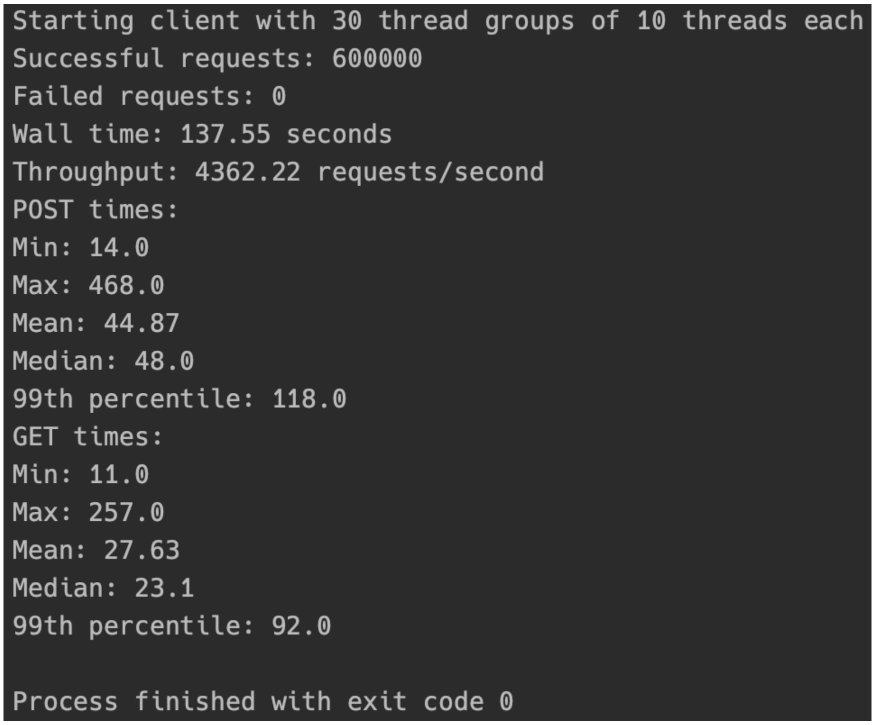
Solution:
I chose MongoDB as my database, so I didn't do a lot of scaling modification to the DB part. I decided to scale up and scale out the servlet to test out how the throughput would improve.

The two changes I made:

1. Scale out servlet: 2 servlets -> 3 servlets
2. Scale up servlet: t2.micro -> t2.medium

After scaling the servlet, I was able to achieve a throughput of about ~4000 requests per second compared to the previously ~3000 requests per second, which is a nearly 40% increase.

Result:
30 Thread groups with 3 servlets of t2.medium



```
Starting client with 30 thread groups of 10 threads each
Successful requests: 600000
Failed requests: 0
Wall time: 137.55 seconds
Throughput: 4362.22 requests/second
POST times:
Min: 14.0
Max: 468.0
Mean: 44.87
Median: 48.0
99th percentile: 118.0
GET times:
Min: 11.0
Max: 257.0
Mean: 27.63
Median: 23.1
99th percentile: 92.0

Process finished with exit code 0
```

Overall Table Throughput Comparison:

Thread Groups	Single Server/DB	Two Load Balanced Server/DB	30 Thread Group Optimized
10	1703	2344	N/A
20	1957	2675	N/A
30	2094	3061	4362