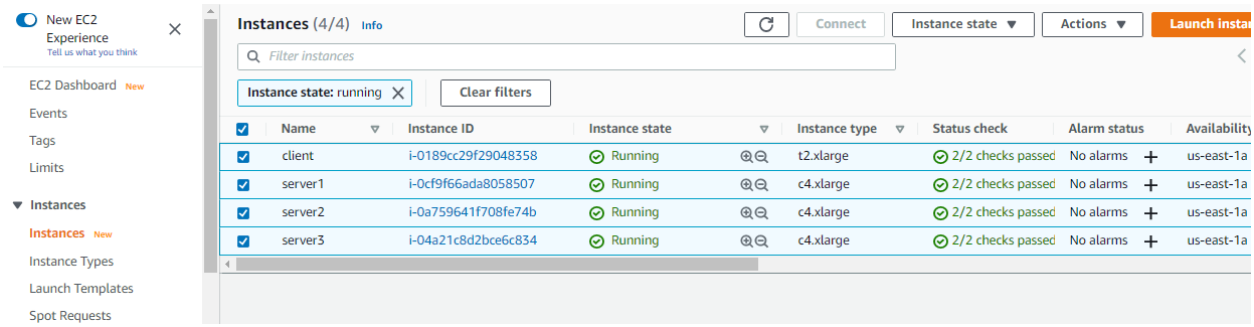
Name : Shweta Mandavgane
Assignment no: 4

INDEX

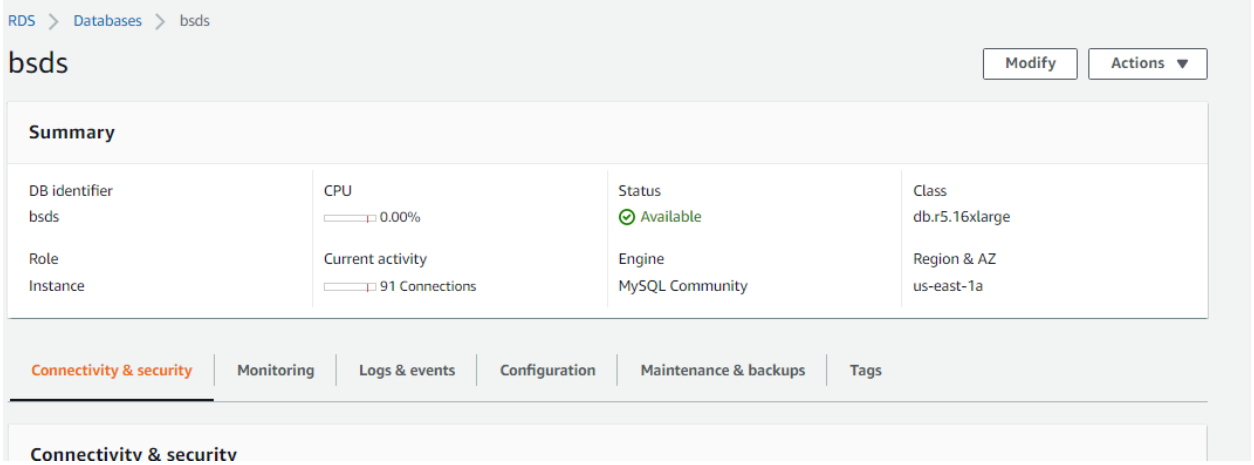
DESCRIPTION

Below is a detailed description of the tasks completed:

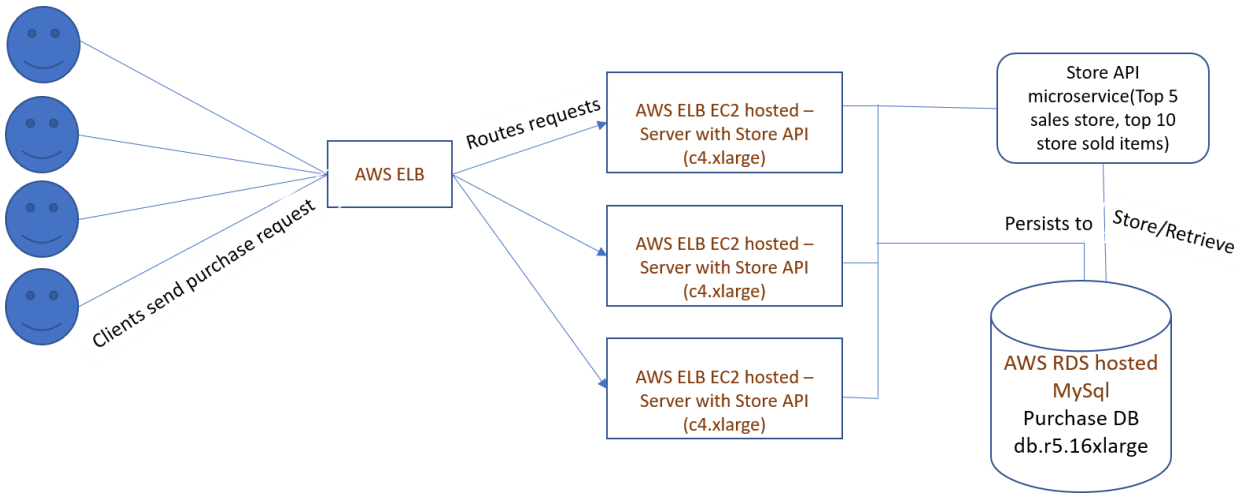
- **URL of Github repository:** https://github.com/shwetamandavgane/bsdsa2.git
- **Server URL:**
  - Loadbalancer URL: **http://bsds-612468346.us-east-1.elb.amazonaws.com/**
  - Used 3 servers with configuration as below:



  - Used RDS with below configuration



New Server design experiment description:



Through this assignment, I was trying to gain more insights into how multiple systems work and how the performance improves when we scale hardware/infrastructure alongwith learning various AWS SDKs/libraries.

In an effort to achieve this, I tried the below experiments

1) Java Web application with Tomcat as server:
   I tried a multiple of options for assignment 4 as below:
   a) Using dynamodb for assignment3 with 1 consumer and rabbitmq
   b) Using dynamodb for assignment3 with 3 consumers and rabbitmq
   c) Using RDS(t3.small/16xlarge) for assignment3 with 3 consumer and rabbitmq

   The above results were not satisfying as database inserts from consumer were too slow(175/s for 3 consumers total). Also, they did not guarantee that records would be persisted in the database. With such low throughput and rabbitmq crash, it seemed like rabbitmq was a huge bottleneck for this design of mine. So rolled back to assignment2 and experimented with dynamodb and RDS.

   Below are a few things I tried experimenting :
   a) Tried working with Java Web application with dynamodb for assignment 2, but did not observe much performance gain with dynamodb. I am not sure if the bottleneck was because of improper use of DynamoDBMapper class or the libraries.
   b) So switched back to **RDS with larger capacity** as mentioned above and it worked all fine. used **HikariCP** for performance improvement.

Also, the Store microservice which initially used H2 is now shifted to Message database table, since now using an ALB for routing requests to appropriate servers.

Finally the design looks like below:

Development side :

Found 2 major development bottlenecks of assignment 2 & 3:

1) On web app side where the connections to Database were exceeding the connection pool. SO used HikariCP which gave a static and optimized connection pool for Java.
2) The client application was also slow because of RandomNumberGenerator was not thread safe. So used ThreadRandomNumberGenerator function of Java.

Infrastructure side:

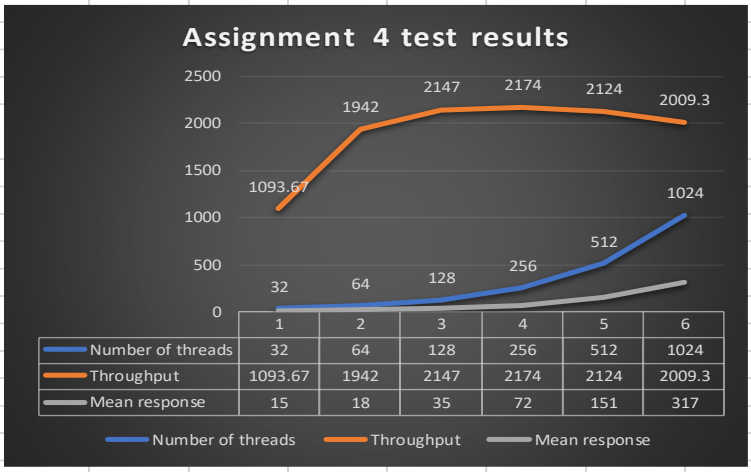
1) The RDS instance was inefficient to handle such huge traffic. Limiting the number of connection through HikariCP as well as using 16xlarge instance, as compared to smaller instance in assignment2 and 3 worked well.
2) Also, for server side, using large instances as against micro ones with "High" network speed worked better.

1) How does the system work:
   On every client POST call, the request and URL is validated -> if invalid a SC_BAD_REQUEST
   If URL is valid => call addPurchase() method were a static DB connection is established(Used HikariCP in assignment4) and the record is saved to the DB. So in total the following number of records will persist in DB:
   Number of threads * Number of hours * Number of purchases/hour * number of items in payload
2) Database design:
   Currently, there is only one table in the schema -> Purchases table with 5 columns and one auto-increment column as primary key.
   I also tried normalizing the tables to 5 tables : stock, store, customer, purchase, store_item. But it was creating bottleneck due to lots of foreign key relations and thus the server/DB used to stop responding. The DB connections used to be exhausted and the "Too many connections". Since the DB is very simple and we only need to insert data without reading, the best way to achive maximum response time is to have a single table according to me.

❖ **Output Graph Summary & Comparison assignment4 vs. assignment2 vs. assignment3 respectively:**

Assignment 4 output:

Number of threads	Throughput	Mean response	Failing requests
32	1093.67	15	0
64	1942	18	0
128	2147	35	0
256	2174	72	0
512	2124	151	0
1024	2009.3	317	16



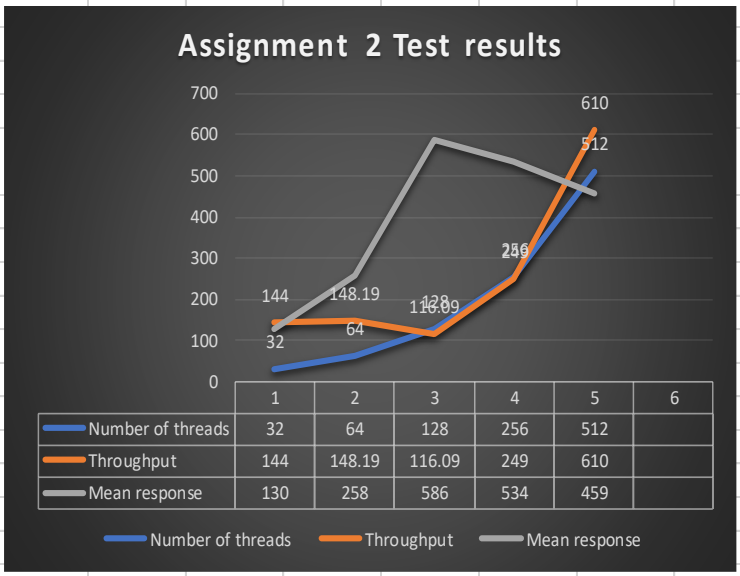
Above is the assignment 4 graph with all results successful.

Assignment4 vs. Assignment2:

In my assignment 2, the throughput was very low(almost below 500). With the above server architecture and improvements:

1) Throughput improved(2174 for 256 threads as against 249 for assignment 2's 256 threads)
2) Success rate is 100%.There were many failing requests(in millions) for assignment2 (shown below)
3) Solution is now scalable(Works for 1024 threads vs failure for 512 assignment2 single server)
4) Mean response time is much lower(72 for 256 threads vs 524 in assignment2

Number of threads	Throughput	Mean response	Failing requests
32	144	130	0
64	148.19	258	778
128	116.09	586	48508
256	249	534	402494
512	610	459	906795

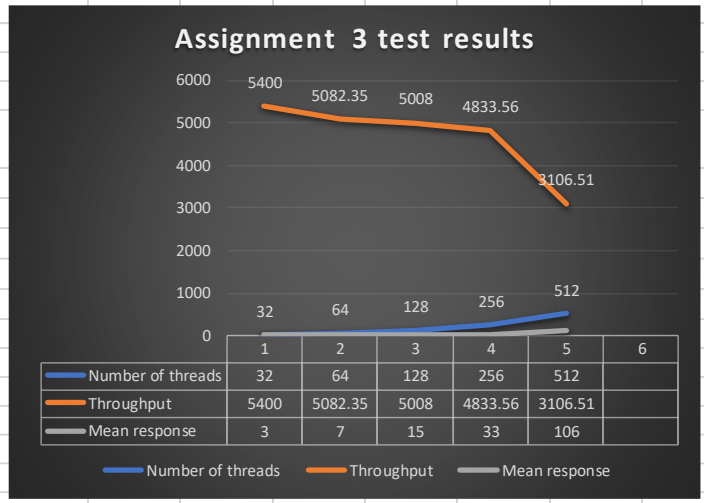


Assignment4 vs. Assignment3:

With the above server architecture and improvements:

1) DB Insert success rate for assignment4 is 100%. There were many failing requests(in millions) for assignment3 (shown below). For queue size of 128 threads, database inserts from rabbitmq used to run overnight and 256 never succeeded completely.
2) Solution is now scalable(Works for 1024 threads vs failure for 256 assignment3 consumers)

Number of threads	Throughput	Mean response	Failing requests
32	5400	3	0
64	5082.35	7	0
128	5008	15	0
256	4833.56	33	0
512	3106.51	106	0

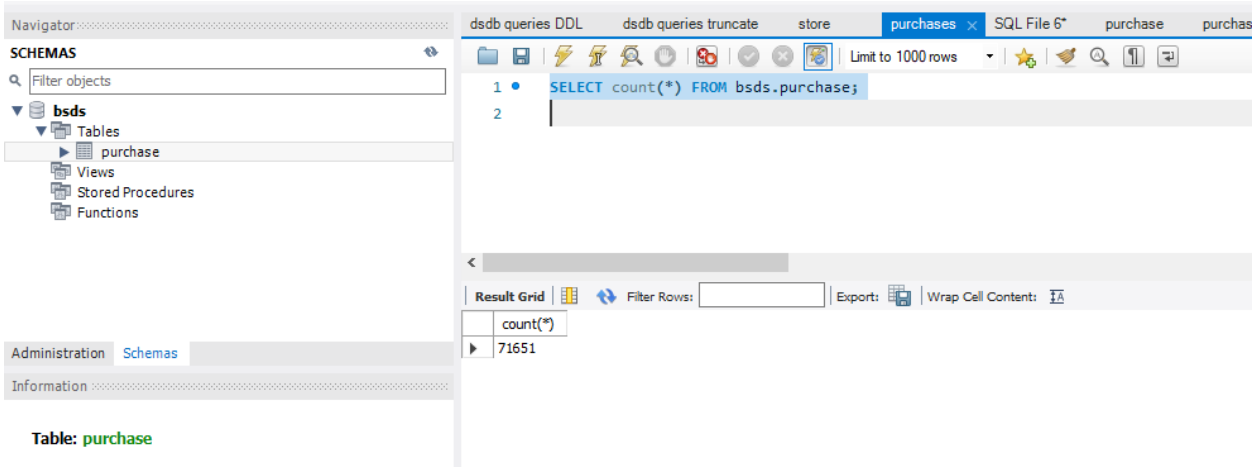


Test runs

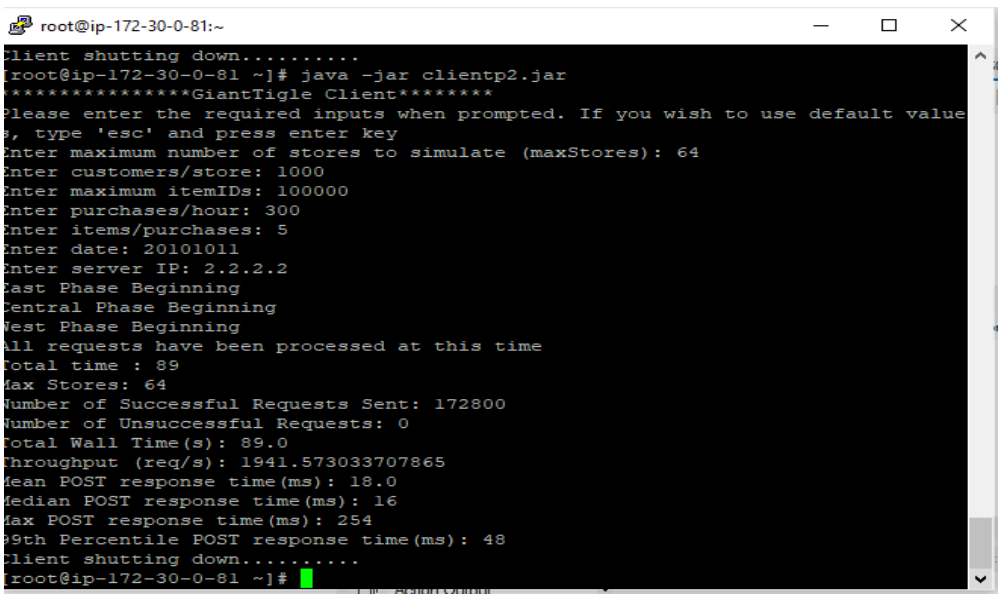
Thread Count: 32

```
root@ip-172-30-0-81:~
Client shutting down..........
[root@ip-172-30-0-81 ~]# java -jar clientp2.jar
****************GiantTigle Client********
Please enter the required inputs when prompted. If you wish to use default value
s, type 'esc' and press enter key
Enter maximum number of stores to simulate (maxStores): 32
Enter customers/store: 1000
Enter maximum itemIDs: 100000
Enter purchases/hour: 300
Enter items/purchases: 5
Enter date: 20210101
Enter server IP: 2.2.2.2
East Phase Beginning
Central Phase Beginning
West Phase Beginning
All requests have been processed at this time
Total time : 79
Max Stores: 32
Number of Successful Requests Sent: 86400
Number of Unsuccessful Requests: 0
Total Wall Time(s): 79.0
Throughput (req/s): 1093.6708860759493
Mean POST response time(ms): 15.0
Median POST response time(ms): 15
Max POST response time(ms): 256
99th Percentile POST response time(ms): 22
Client shutting down..........
[root@ip-172-30-0-81 ~]#
```

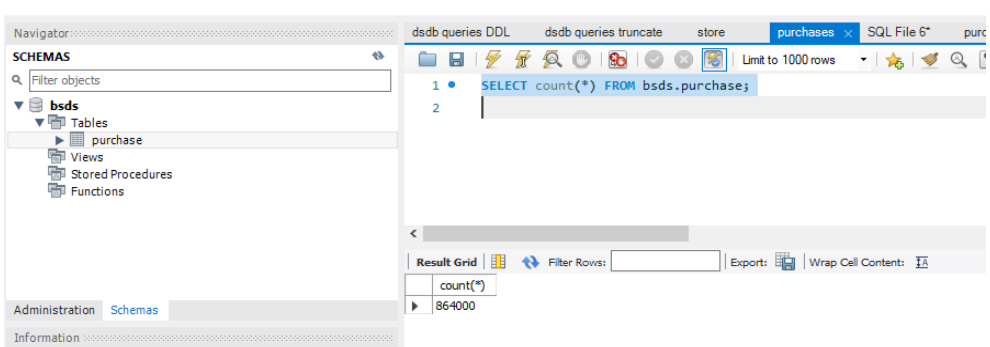
DB Records to be inserted should be 432000 as per calculation in first part. And since all requests are successful everything is sent to DB.



Thread Count: 64



Expected records: 864000 . The below result:



Thread Count: 128

```
root@ip-172-30-0-81:~
Client shutting down..........
[root@ip-172-30-0-81 ~]# java -jar clientp2.jar
****************GiantTigle Client********
Please enter the required inputs when prompted. If you wish to use default value
s, type 'esc' and press enter key
Enter maximum number of stores to simulate (maxStores): 128
Enter customers/store: 1000
Enter maximum itemIDs: 100000
Enter purchases/hour: 300
Enter items/purchases: 5
Enter date: 20210101
Enter server IP: 2.2.2.2
East Phase Beginning
Central Phase Beginning
West Phase Beginning
All requests have been processed at this time
Total time : 161
Max Stores: 128
Number of Successful Requests Sent: 345600
Number of Unsuccessful Requests: 0
Total Wall Time(s): 161.0
Throughput (req/s): 2146.583850931677
Mean POST response time(ms): 35.0
Median POST response time(ms): 19
Max POST response time(ms): 306
99th Percentile POST response time(ms): 144
Client shutting down..........
[root@ip-172-30-0-81 ~]#
```

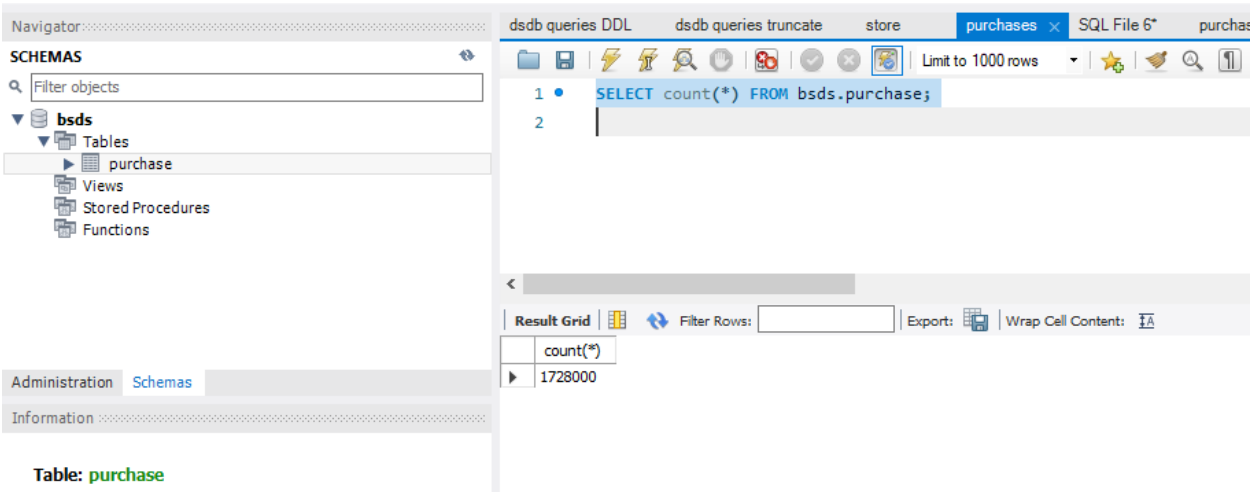
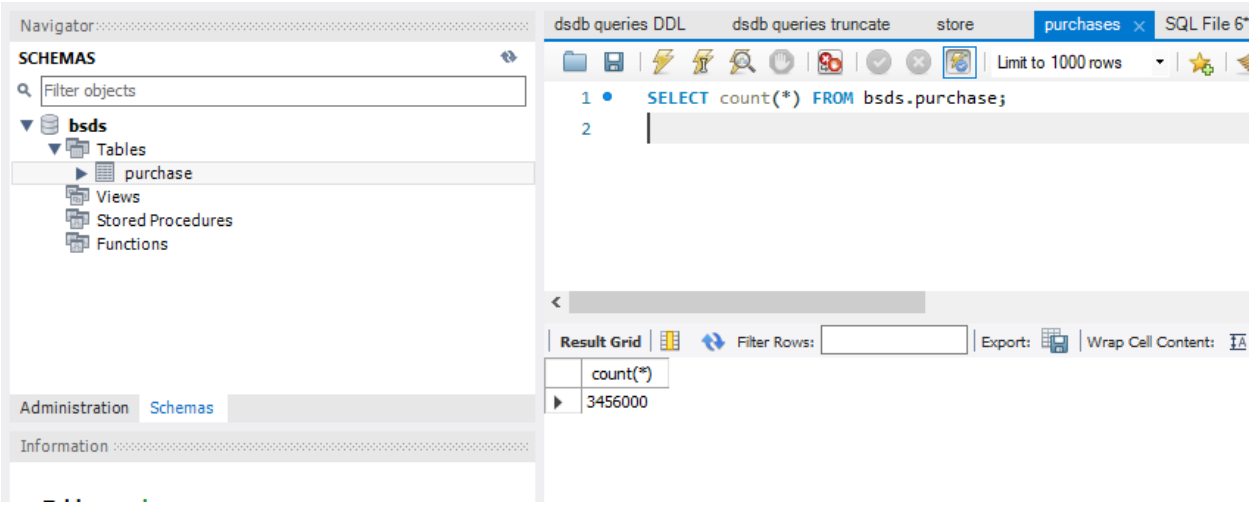


Table: purchase

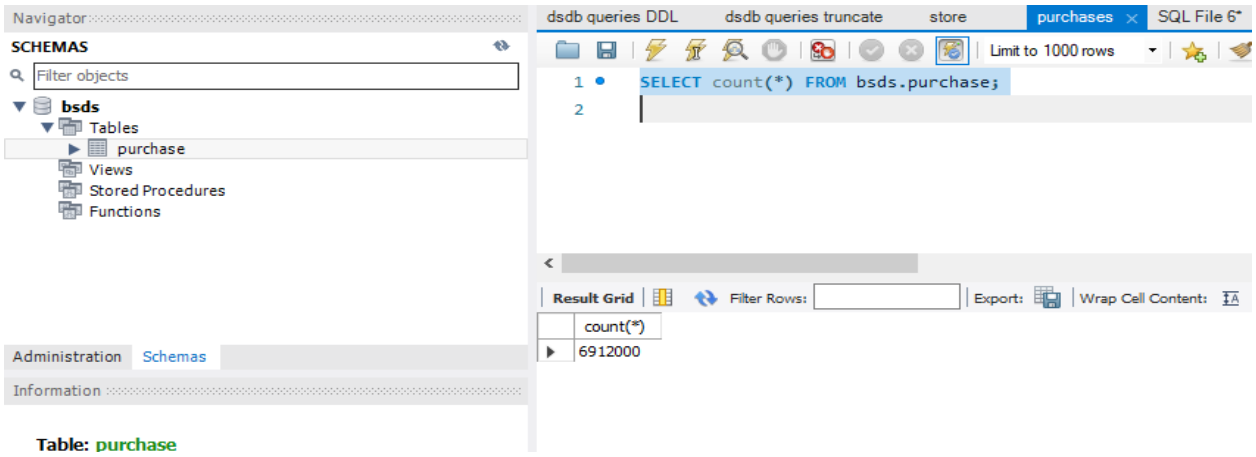
Client Threads: 256

```
root@ip-172-30-0-81:~
[root@ip-172-30-0-81 ~]#
[root@ip-172-30-0-81 ~]# java -jar clientp2.jar
****************GiantTigle Client********
Please enter the required inputs when prompted. If you wish to use default value
s, type 'esc' and press enter key
Enter maximum number of stores to simulate (maxStores): 256
Enter customers/store: 1000
Enter maximum itemIDs: 100000
Enter purchases/hour: 300
Enter items/purchases: 5
Enter date: 20210101
Enter server IP: 2.2.2.2
East Phase Beginning
Central Phase Beginning
West Phase Beginning
All requests have been processed at this time
Total time : 318
Max Stores: 256
Number of Successful Requests Sent: 691200
Number of Unsuccessful Requests: 0
Total Wall Time(s): 318.0
Throughput (req/s): 2173.5849056603774
Mean POST response time(ms): 72.0
Median POST response time(ms): 20
Max POST response time(ms): 382
99th Percentile POST response time(ms): 331
Client shutting down..........
[root@ip-172-30-0-81 ~]#
```



Test run 512 threads:

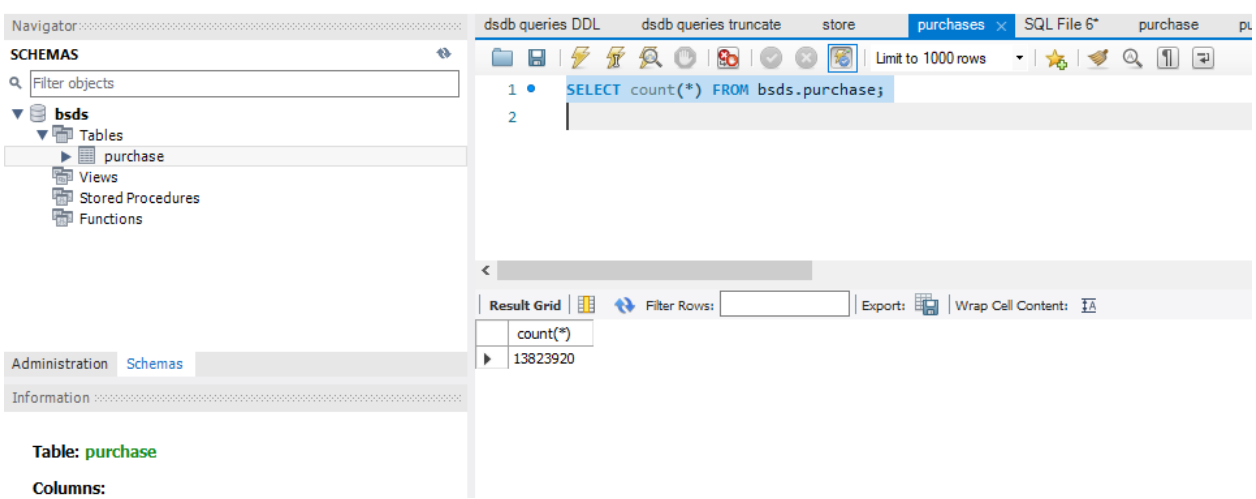
```
root@ip-172-30-0-81:~
Client shutting down..........
[root@ip-172-30-0-81 ~]# java -jar clientp2.jar
****************GiantTigle Client********
Please enter the required inputs when prompted. If you wish to use default value
s, type 'esc' and press enter key
Enter maximum number of stores to simulate (maxStores): 512
Enter customers/store: 1000
Enter maximum itemIDs: 100000
Enter purchases/hour: 300
Enter items/purchases: 5
Enter date: 20210101
Enter server IP: 2.2.2.2
East Phase Beginning
Central Phase Beginning
West Phase Beginning
All requests have been processed at this time
Total time : 651
Max Stores: 512
Number of Successful Requests Sent: 1382400
Number of Unsuccessful Requests: 0
Total Wall Time(s): 651.0
Throughput (req/s): 2123.5023041474656
Mean POST response time(ms): 151.0
Median POST response time(ms): 22
Max POST response time(ms): 861
99th Percentile POST response time(ms): 644
Client shutting down..........
[root@ip-172-30-0-81 ~]#
```



5. Successful test with 1024 clients that shows the solution is still scalable:

Below are screenshots of 2 consecutive successful runs with 1024 clients where the throughput is almost similar to the other test runs:

```
root@ip-172-30-0-81:~
Client shutting down..........
[root@ip-172-30-0-81 ~]# java -jar clientp2.jar
****************GiantTigle Client********
Please enter the required inputs when prompted. If you wish to use default value
s, type 'esc' and press enter key
Enter maximum number of stores to simulate (maxStores): 1024
Enter customers/store: 1000
Enter maximum itemIDs: 100000
Enter purchases/hour: 300
Enter items/purchases: 5
Enter date: 20210101
Enter server IP: 2.1.1.1
East Phase Beginning
Central Phase Beginning
West Phase Beginning
All requests have been processed at this time
Total time : 1376
Max Stores: 1024
Number of Successful Requests Sent: 2764784
Number of Unsuccessful Requests: 16
Total Wall Time(s): 1376.0
Throughput (req/s): 2009.3023255813953
Mean POST response time(ms): 317.0
Median POST response time(ms): 54
Max POST response time(ms): 1514
99th Percentile POST response time(ms): 1346
Client shutting down..........
[root@ip-172-30-0-81 ~]#
```



root@ip-172-30-0-81:~

```
Client shutting down..........
[root@ip-172-30-0-81 ~]# java -jar clientp2.jar
****************GiantTigle Client********
Please enter the required inputs when prompted. If you wish to use default value
s, type 'esc' and press enter key
Enter maximum number of stores to simulate (maxStores): 1024
Enter customers/store: 1000
Enter maximum itemIDs: 100000
Enter purchases/hour: 300
Enter items/purchases: 5
Enter date: 20210101
Enter server IP: 2.2.2.2
East Phase Beginning
Central Phase Beginning
West Phase Beginning
All requests have been processed at this time
Total time : 3732
Max Stores: 1024
Number of Successful Requests Sent: 2764771
Number of Unsuccessful Requests: 29
Total Wall Time(s): 3732.0
Throughput (req/s): 740.8360128617363
Mean POST response time(ms): 748.0
Median POST response time(ms): 27
Max POST response time(ms): 9221
99th Percentile POST response time(ms): 8638
Client shutting down..........
[root@ip-172-30-0-81 ~]#
```

Navigator

SCHEMAS

Filter objects

- bsds
  - Tables
    - purchase
  - Views
  - Stored Procedures
  - Functions

Administration Schemas

Information

dsdb queries DDL | dsdb queries truncate | store | purchases | SQL File 6* | purchase

Limit to 1000 rows

```
1 SELECT count(*) FROM bsds.purchase;
2
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

count(*)
13823855

❖ **Client Implementation:**

There are two classes for this part:

1) Store -> Main Program to be run
2) StoreThread -> The class that implements Runnable. All the threads use the run() here

This program accepts the required inputs from command line, validates them against the given conditions like only integer values needed, range etc and then calls the POST method as mentioned in above URL.

```
un:  Store
"C:\Program Files\JetBrains\IntelliJ IDEA 2019.2.2\jbr\bin\java.exe" ...
****************GiantTigle Client********
Please enter the required inputs when prompted. If you wish to use default values, type 'esc' and press enter key
Enter maximum number of stores to simulate (maxStores): 256
Enter customers/store: 1000
Enter maximum itemIDs: 10000
Enter purchases/hour: 60
Enter items/purchases: 5
Enter date: 20210101
Enter server IP: 128.01.01.01
East Phase Beginning
Central Phase Beginning
```

If any input is entered wrong, InputMismatchException exception gets displayed as follows:

```
Store
"C:\Program Files\JetBrains\IntelliJ IDEA 2019.2.2\jbr\bin\java.exe" ...
****************GiantTigle Client********
Please enter the required inputs when prompted. If you wish to use default values, type 'esc' and press enter key
Enter maximum number of stores to simulate (maxStores): ss
Exception in thread "main" java.util.InputMismatchException
  at java.base/java.util.Scanner.throwFor(Scanner.java:939)
  at java.base/java.util.Scanner.next(Scanner.java:1594)
  at java.base/java.util.Scanner.nextInt(Scanner.java:2258)
  at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
  at com.bsds.assignment1.part1.clientp1.Store.acceptInputs(Store.java:40)
  at com.bsds.assignment1.part1.clientp1.Store.main(Store.java:111)

Process finished with exit code 1
```

The main() program calls the given number of threads, for three phases east(threads/4), central(threads/4) and west(threads/2) when the threshold is achieved. For this synchoronization, I have used CountDownLatch. CountDownLatch is used for 3 phases, once its 0 for every phase then the next set of threads will execute and the totalThreads latch takes care of completion of execution of all threads.

The run() in StoreThread runs for every Store and has a loop for 9 hours, which calls the numPurchases times(input by user), and calls the POST URL multiple times. Every POST URL has a request body as given in the assignment, waits for 200/201 response and then immediately calls next request. If there is a 500/401 response, then its logged to stderr(Refer images below for 256 thread count).

One new enhancement for this **assignment2** is the **retries:** If the client receives a response other than 201, the client again tries to send the request till upto 3 times. If after 3 times the response is not received, then the client stops trying to save the record.

cURL Outputs from calling GET on the Store microservice:

Top 5 items:

```
[ec2-user@ip-172-30-0-93 ~]$ curl -i -H "Accept: application/json" http://ec2-3-219-218-124.compute-1.amazonaws.com/Server_6650_war/store/72  | grep }| python -mjson.to
ol
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   202    0   202    0     0    547      0 --:--:-- --:--:-- --:--:--   545
{
    "stores": [
        {
            "itemId": 51,
            "numberOfItems": 3709051
        },
        {
            "itemId": 47,
            "numberOfItems": 3707571
        },
        {
            "itemId": 91,
            "numberOfItems": 3690768
        },
        {
            "itemId": 85,
            "numberOfItems": 3689667
        },
        {
            "itemId": 59,
            "numberOfItems": 3675224
        }
    ]
}
[ec2-user@ip-172-30-0-93 ~]$
```

Top 10 stores:

```
ec2-user@ip-172-30-0-93:~
[ec2-user@ip-172-30-0-93 ~]$ curl -i -H "Accept: application/json" http://ec2-3-
219-218-124.compute-1.amazonaws.com/Server_6650_war/item/35  | grep }| python -m
json.tool
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   522    0   522    0     0   1572      0 --:--:-- --:--:-- --:--:--  1567
{
    "stores": [
        {
            "itemId": 35,
            "numberOfItems": 7143207,
            "storeid": 20
        },
        {
            "itemId": 35,
            "numberOfItems": 6925902,
            "storeid": 30
        },
        {
            "itemId": 35,
            "numberOfItems": 6613822,
            "storeid": 12
        },
        {
            "itemId": 35,
            "numberOfItems": 6403767,
            "storeid": 24
        },
        {
            "itemId": 35,
            "numberOfItems": 3947254,
            "storeid": 60
        },
        {
            "itemId": 35,
            "numberOfItems": 3899873,
            "storeid": 36
        },
        {
            "itemId": 35,
            "numberOfItems": 3754550,
            "storeid": 40
        },
```

```
        {
            "itemId": 35,
            "numberOfItems": 3947254,
            "storeid": 60
        },
        {
            "itemId": 35,
            "numberOfItems": 3899873,
            "storeid": 36
        },
        {
            "itemId": 35,
            "numberOfItems": 3754550,
            "storeid": 40
        },
        {
            "itemId": 35,
            "numberOfItems": 3683334,
            "storeid": 48
        },
        {
            "itemId": 35,
            "numberOfItems": 3673713,
            "storeid": 35
        },
        {
            "itemId": 35,
            "numberOfItems": 3635500,
            "storeid": 72
        }
    ]
}
[ec2-user@ip-172-30-0-93 ~]$
```