Assignment 2 - submitted by Prakriti Dave

A little explanation about my code:

Client side:

1. **PrakMain:** is where I call the function to execute all my phases and also start the thread that reads from a blocking queue all the metrics and writes to a csv file.
2. **MyRunClass:** is the class that implements runnable and executes each thread as per the iterations.
3. **MyMetrics:** is just a class that stores the metrics recorded for each request response, such as timestamp and latencies
4. **MyFirstClient:** is where all the requests are made to the server using webTarget and Client

Server:

1. **NewConnectionClass: is the class where I configure my DBCP configuration using C3P0 library:**

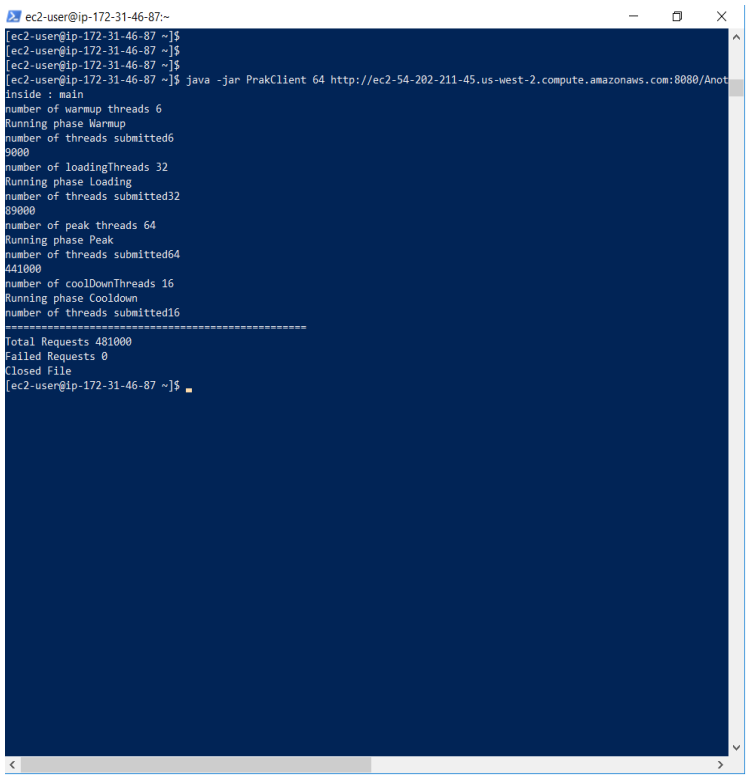
```
public class NewConnectionClass {
  private static ComboPooledDataSource cpds = null;
  static{
    try{
      cpds = new ComboPooledDataSource();
      cpds.setDriverClass("com.mysql.jdbc.Driver");
      cpds.setJdbcUrl("jdbc:mysql://prakschema.ctg53dqnivdw.us-west-2.rds.amazonaws.com:3306/prakschema");
      cpds.setUser("prakriti");
      cpds.setPassword("prakriti");
      // Optional Settings
      cpds.setMinPoolSize(50);
      cpds.setInitialPoolSize(50);
      cpds.setAcquireIncrement(25);
      cpds.setMaxPoolSize(248);
      cpds.setMaxStatementsPerConnection(8);
      cpds.setStatementCacheNumDeferredCloseThreads(1);
//      cpds.setMaxStatements(8);
//      cpds.setNumHelperThreads(10);
//      cpds.setTestConnectionOnCheckin(true);
//      cpds.setTestConnectionOnCheckout(false);
//      cpds.setMaxConnectionAge(28);

    }catch (Exception e){
      e.printStackTrace();
    }
  }

  public static ComboPooledDataSource getDataSource() { return cpds; }
```

2. **UserDao:** creates a connection object interacts with the Database to perform relevant task.
3. **MyFirstApp:** class is where all the REST end points are defined.

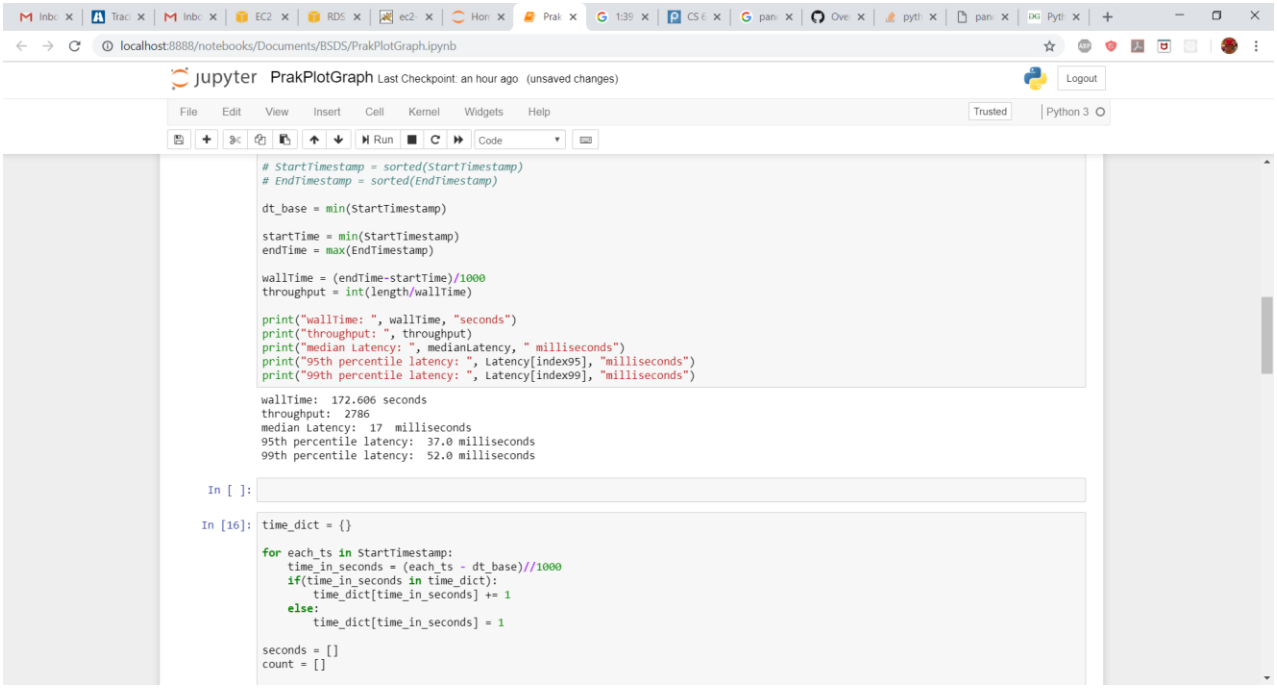
Also, I would like to mention I used a python script to generate all the stats and plot the graph.

Step 1.
Results for a test run with the default client settings for threads/tests, 64 threads, 100 tests/phase

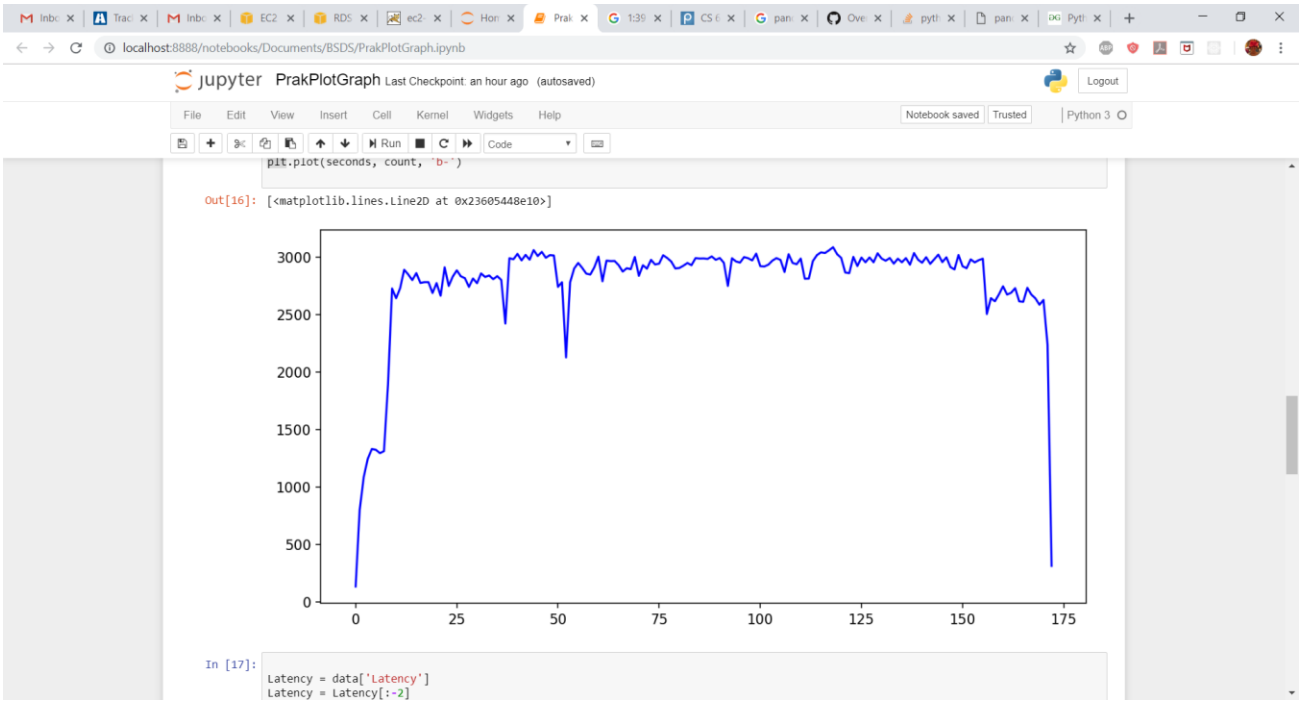
I ran my client on ec2 instance, primarily because I had network issues while running it locally for 256 threads and I would keep getting connection time out. There was quite an improvement on Wall Time After moving my client from local to ec2.



Stats:



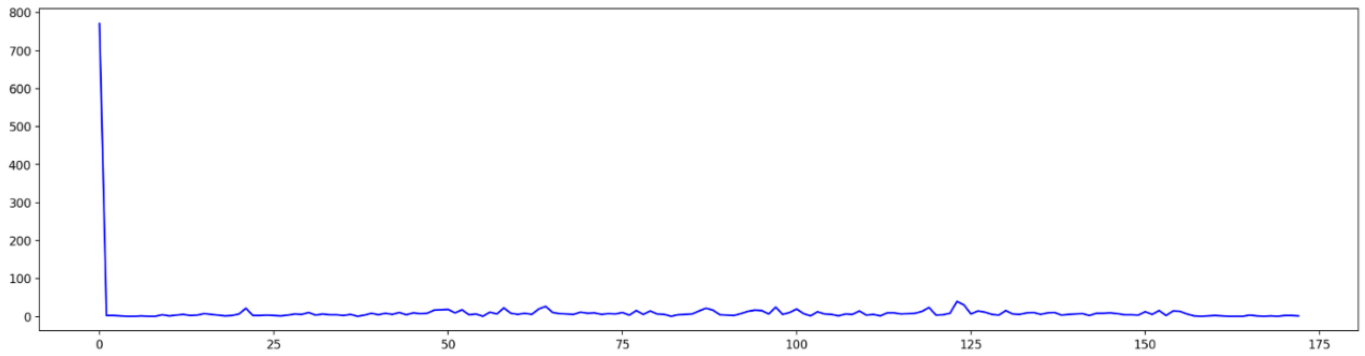
Throughput



Latency

```
max latency:  792.0
min latency:  1.0
```

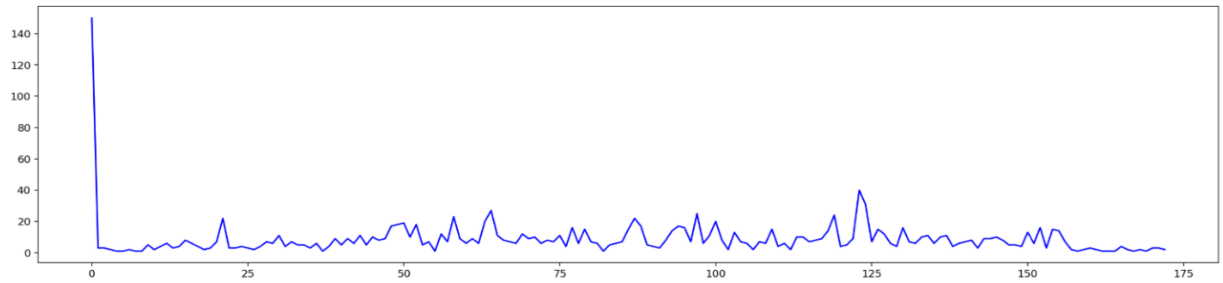
```
[<matplotlib.lines.Line2D at 0x23602245470>]
```



Latency normalized to max 150, for better visualization

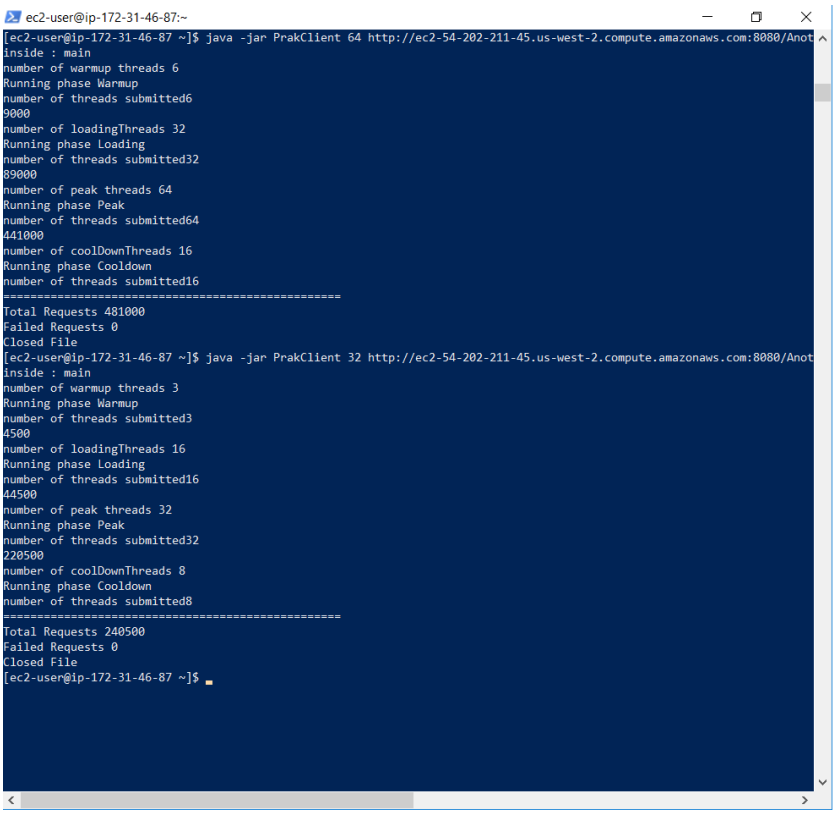
```
fig = plt.figure(figsize=(20,5), dpi=160)

for i in range(len(seconds)):
    if avg_lat[i] > 150:
        avg_lat[i] = 150


plt.plot(seconds, avg_lat, 'b-')
```

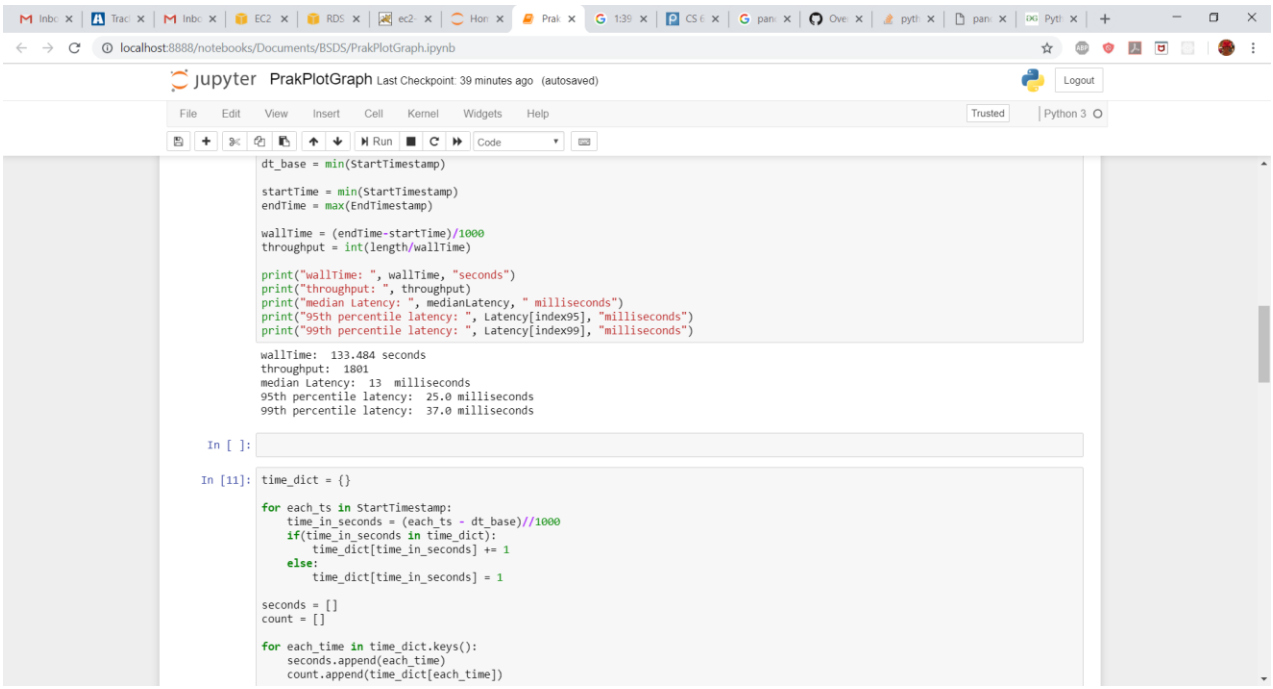
```
[<matplotlib.lines.Line2D at 0x236025e34a8>]
```



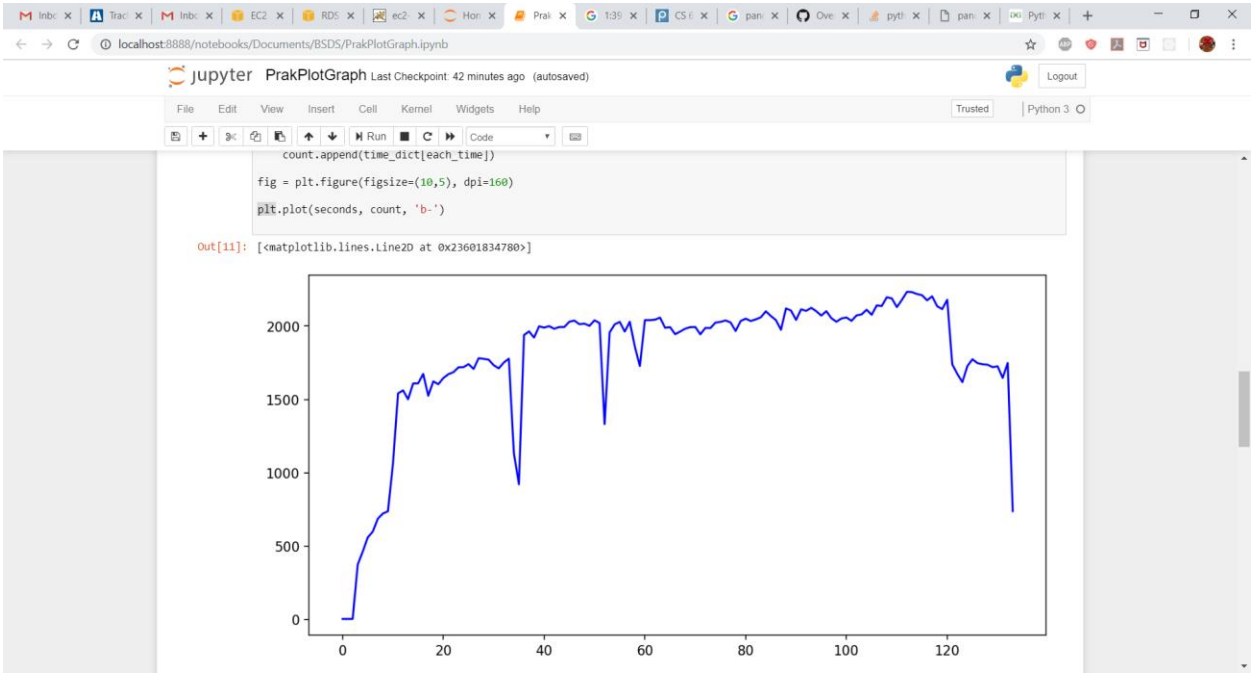
Step 3: Run for thread {32, 64, 128, 256}, all run are for 100 tests/phase

32 Threads

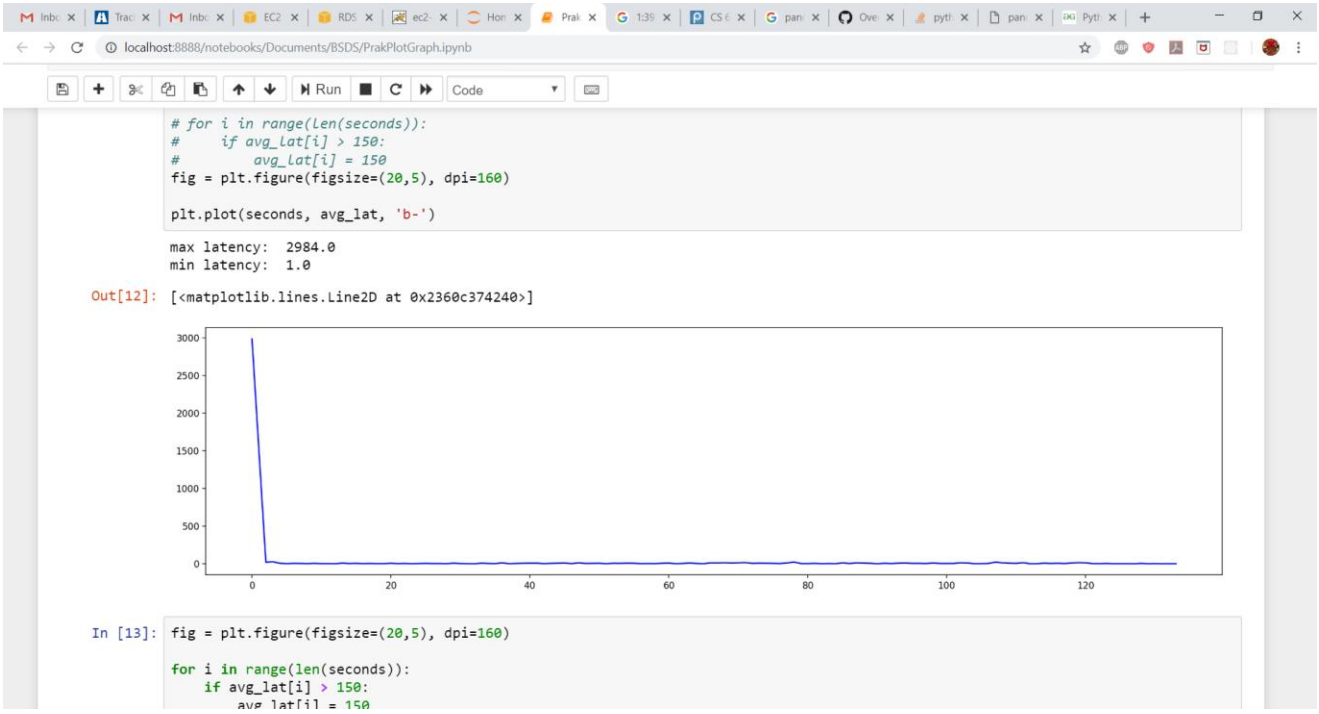




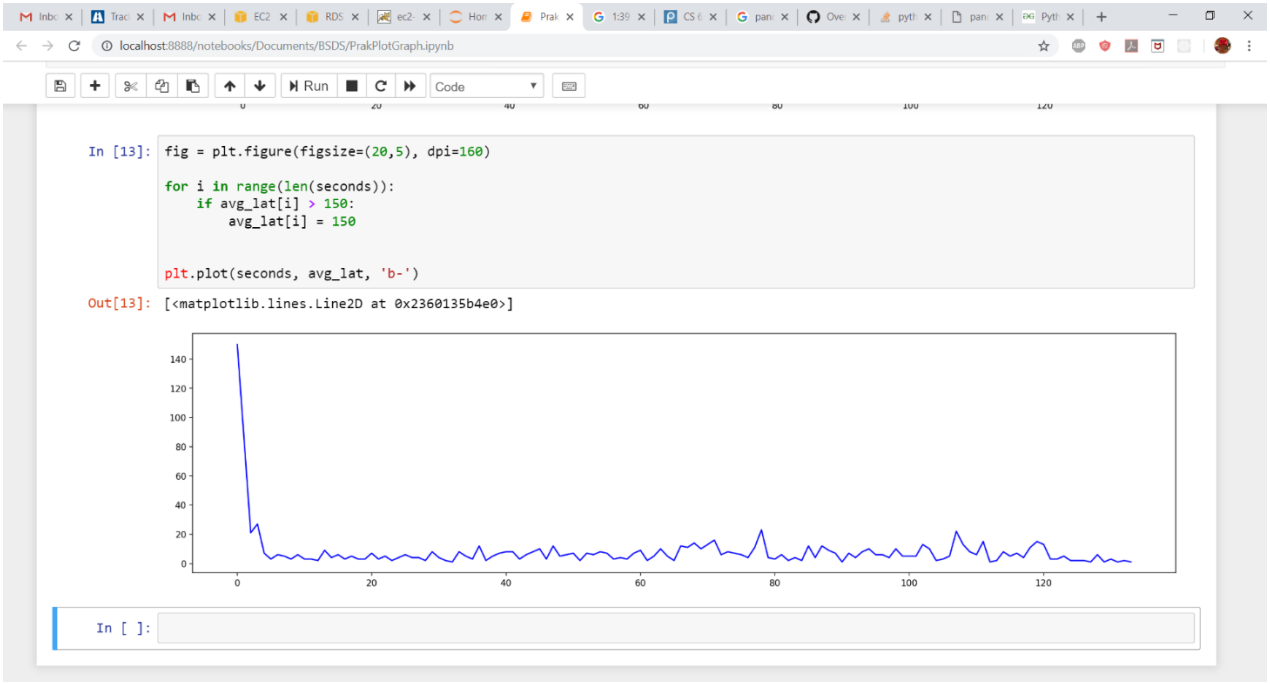
Throughput:



Latency

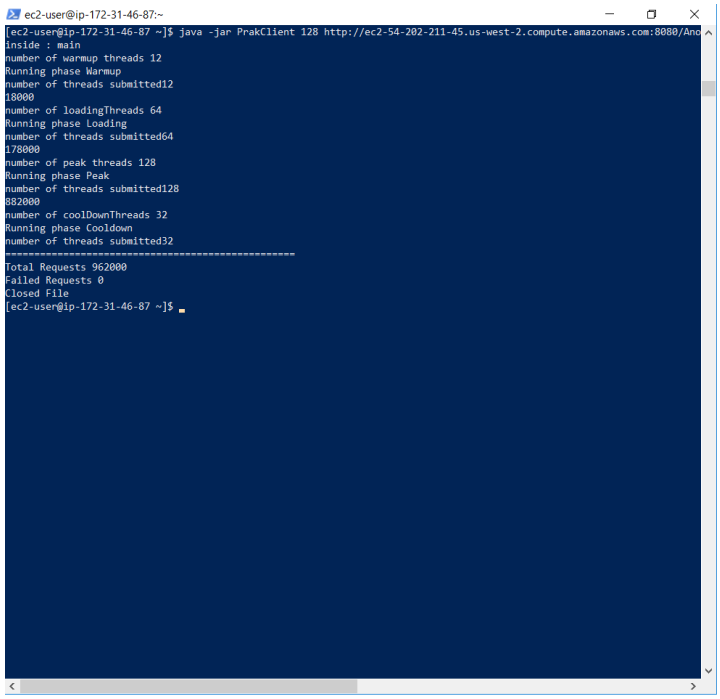


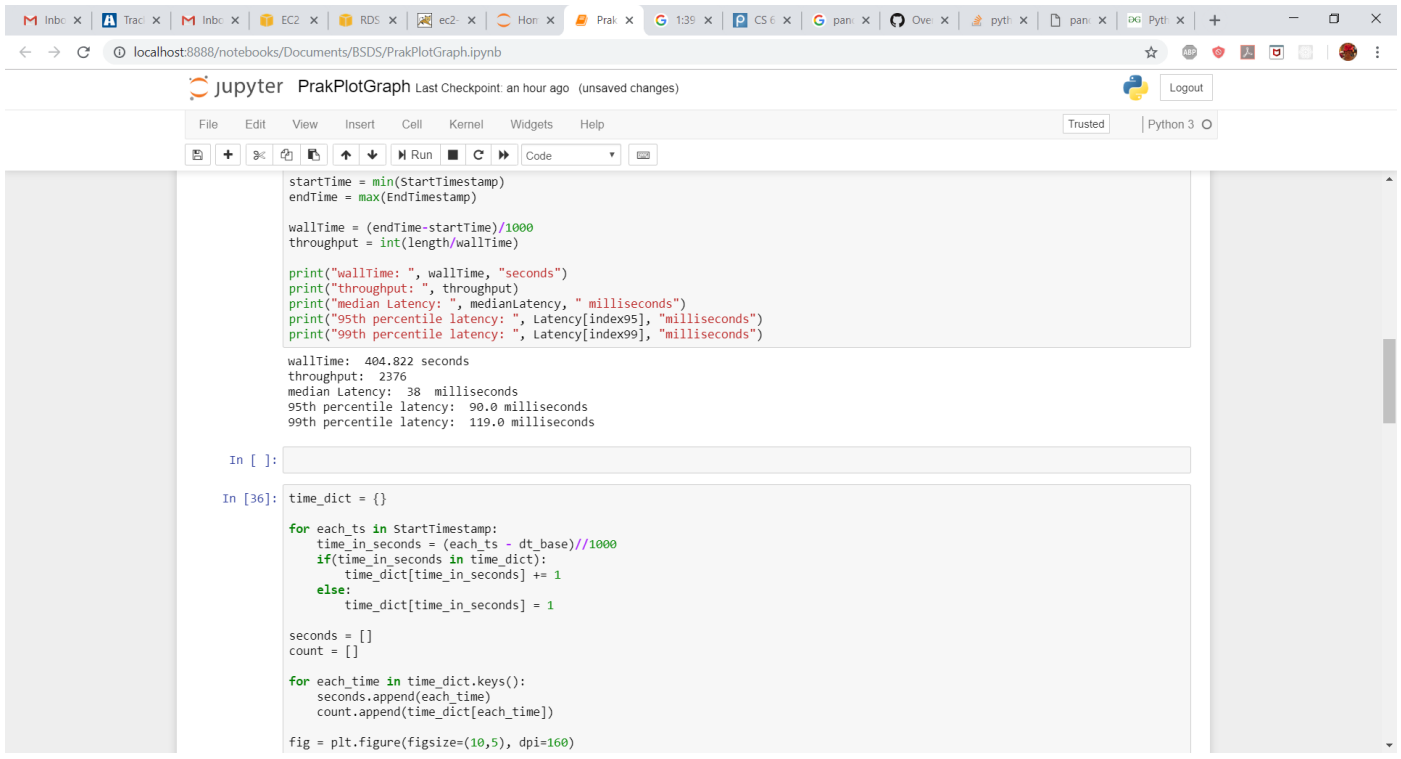
Latency normalized to max 150



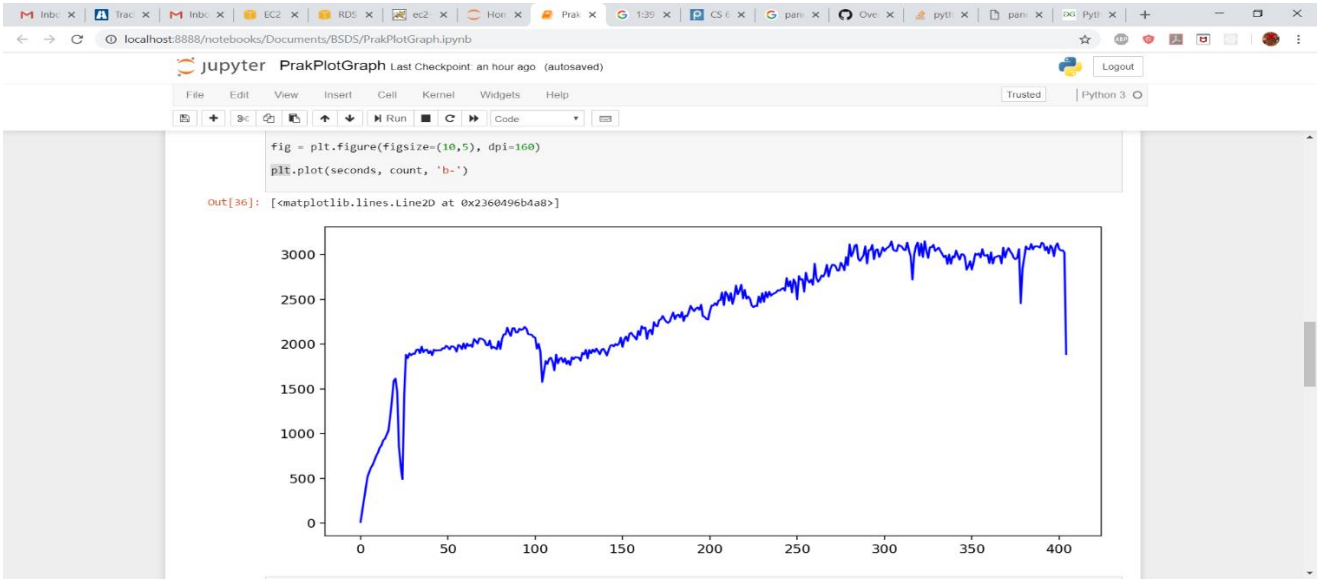
<u>64 Threads are same as for Step1</u>

128 threads:


ec2-user@ip-172-31-46-87:~
[ec2-user@ip-172-31-46-87 ~]$ java -jar PrakClient 128 http://ec2-54-202-211-45.us-west-2.compute.amazonaws.com:8080/Ana
inside : main
number of warmup threads 12
Running phase Warmup
number of threads submitted12
18000
number of loadingThreads 64
Running phase Loading
number of threads submitted64
178000
number of peak threads 128
Running phase Peak
number of threads submitted128
882000
number of coolDownThreads 32
Running phase Cooldown
number of threads submitted32
Total Requests 962000
Failed Requests 0
Closed File
[ec2-user@ip-172-31-46-87 ~]$

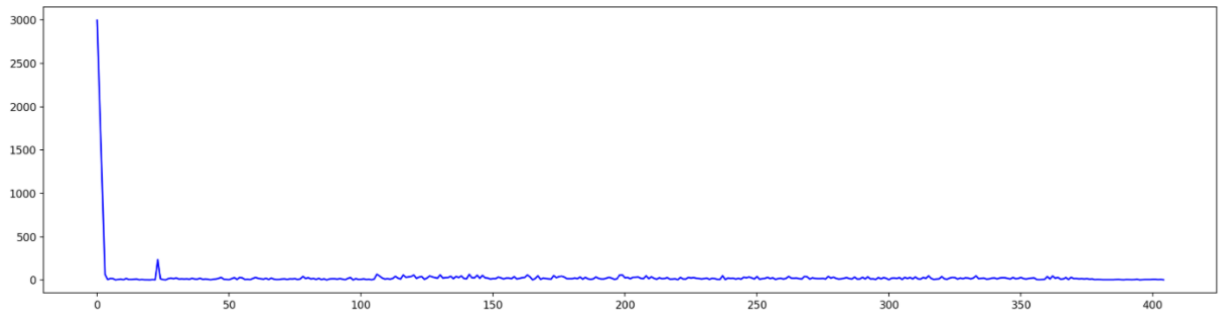

localhost:8888/notebooks/Documents/BSDS/PrakPlotGraph.ipynb
jupyter PrakPlotGraph Last Checkpoint: an hour ago (unsaved changes)
Logout
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
startTime = min(StartTimestamp)
endTime = max(EndTimestamp)
wallTime = (endTime-startTime)/1000
throughput = int(length/wallTime)
print("wallTime: ", wallTime, "seconds")
print("throughput: ", throughput)
print("median Latency: ", medianLatency, " milliseconds")
print("95th percentile latency: ", Latency[index95], "milliseconds")
print("99th percentile latency: ", Latency[index99], "milliseconds")
wallTime: 404.822 seconds
throughput: 2376
median Latency: 38 milliseconds
95th percentile latency: 90.0 milliseconds
99th percentile latency: 119.0 milliseconds
In [ ]:
In [36]:
time_dict = {}
for each_ts in StartTimestamp:
time_in_seconds = (each_ts - dt_base)//1000
if(time_in_seconds in time_dict):
time_dict[time_in_seconds] += 1
else:
time_dict[time_in_seconds] = 1
seconds = []
count = []
for each_time in time_dict.keys():
seconds.append(each_time)
count.append(time_dict[each_time])
fig = plt.figure(figsize=(10,5), dpi=160)

Throughput



Latency

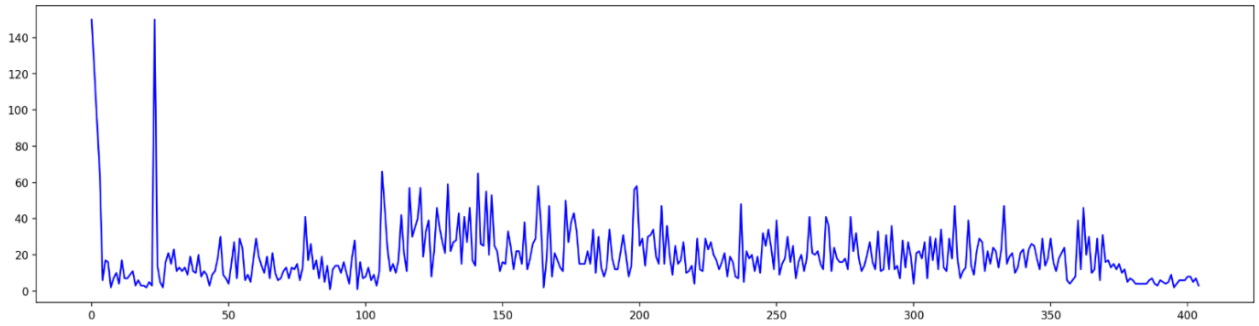
```
max latency:  3002.0
min latency:  1.0

[<matplotlib.lines.Line2D at 0x2360cb3e240>]
```



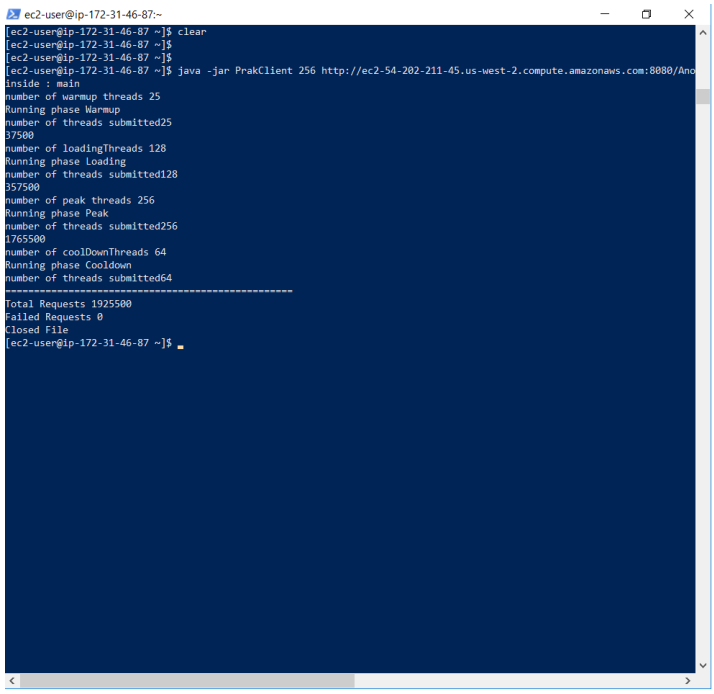
Latency normalized to max 150

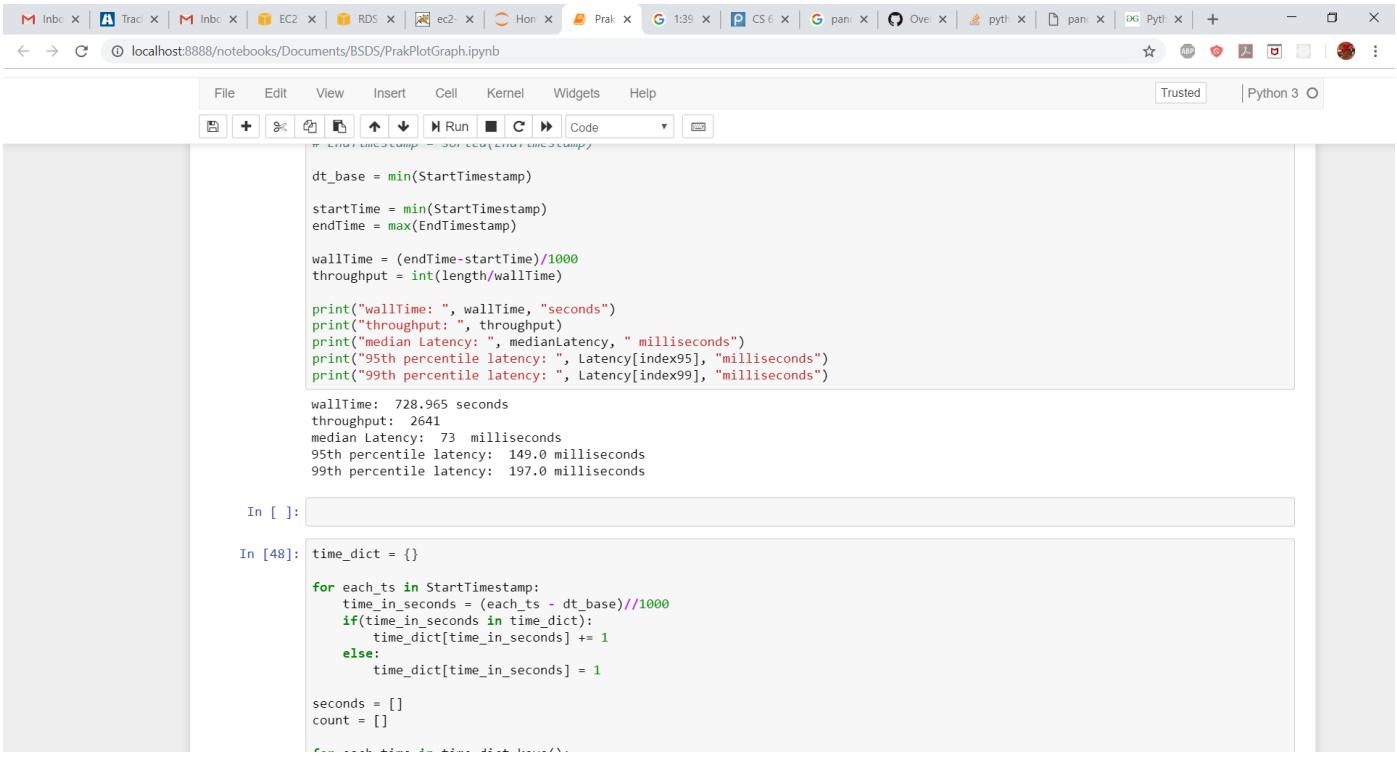
```
fig = plt.figure(figsize=(20,5), dpi=160)

for i in range(len(seconds)):
    if avg_lat[i] > 150:
        avg_lat[i] = 150


plt.plot(seconds, avg_lat, 'b-')
```

```
[<matplotlib.lines.Line2D at 0x236049dfdd8>]
```

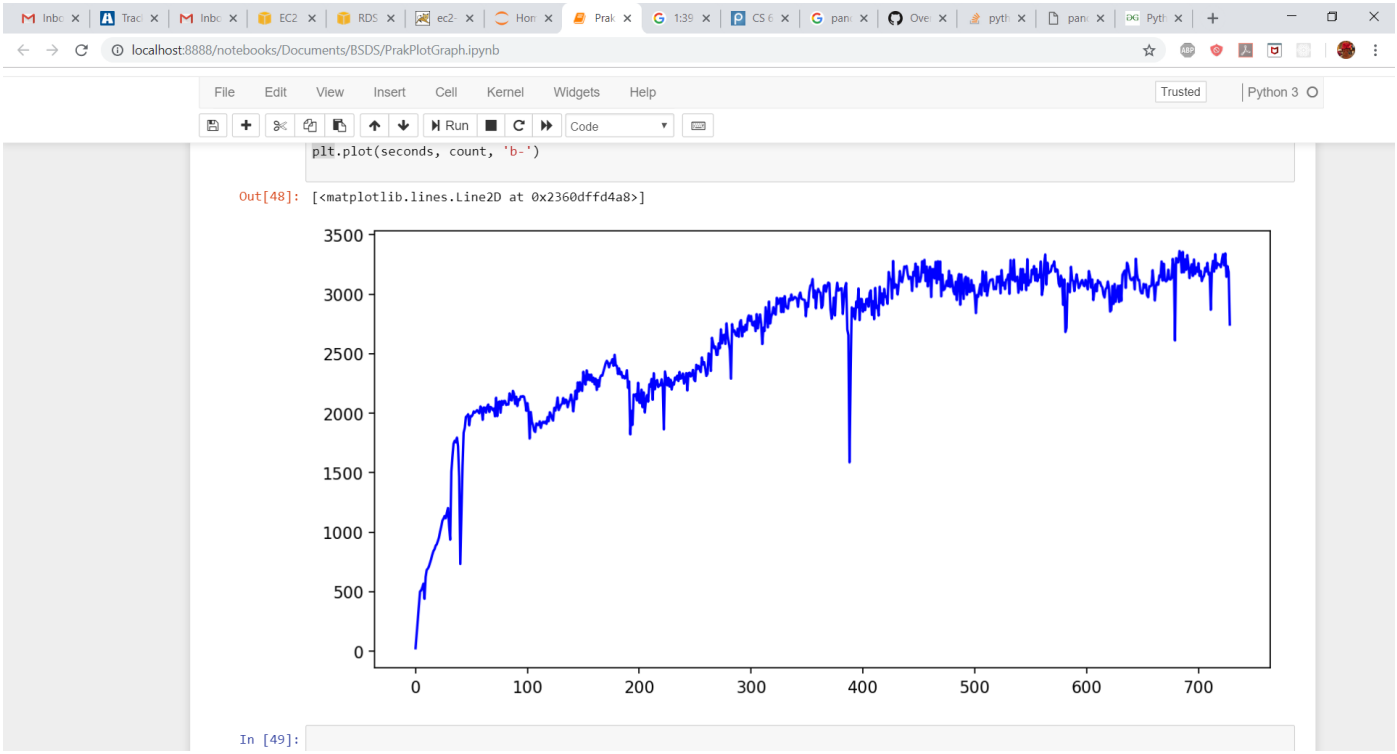


256 threads:

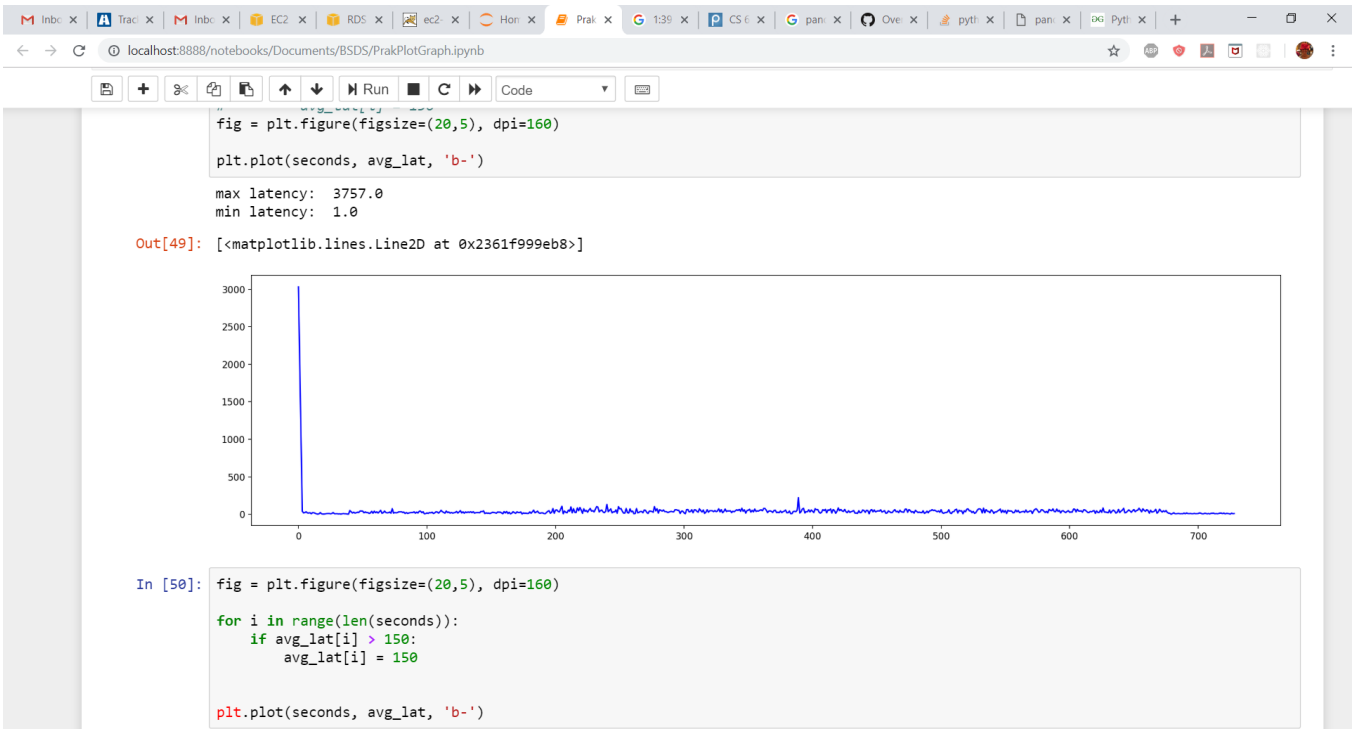

ec2-user@ip-172-31-46-87:~
[ec2-user@ip-172-31-46-87 ~]$ clear
[ec2-user@ip-172-31-46-87 ~]$
[ec2-user@ip-172-31-46-87 ~]$
[ec2-user@ip-172-31-46-87 ~]$ java -jar PrakClient 256 http://ec2-54-202-211-45.us-west-2.compute.amazonaws.com:8080/Ano
inside : main
number of warmup threads 25
Running phase Warmup
number of threads submitted25
37500
number of loadingThreads 128
Running phase Loading
number of threads submitted128
357500
number of peak threads 256
Running phase Peak
number of threads submitted256
1765500
number of coolDownThreads 64
Running phase Cooldown
number of threads submitted64
Total Requests 1925500
Failed Requests 0
Closed File
[ec2-user@ip-172-31-46-87 ~]$

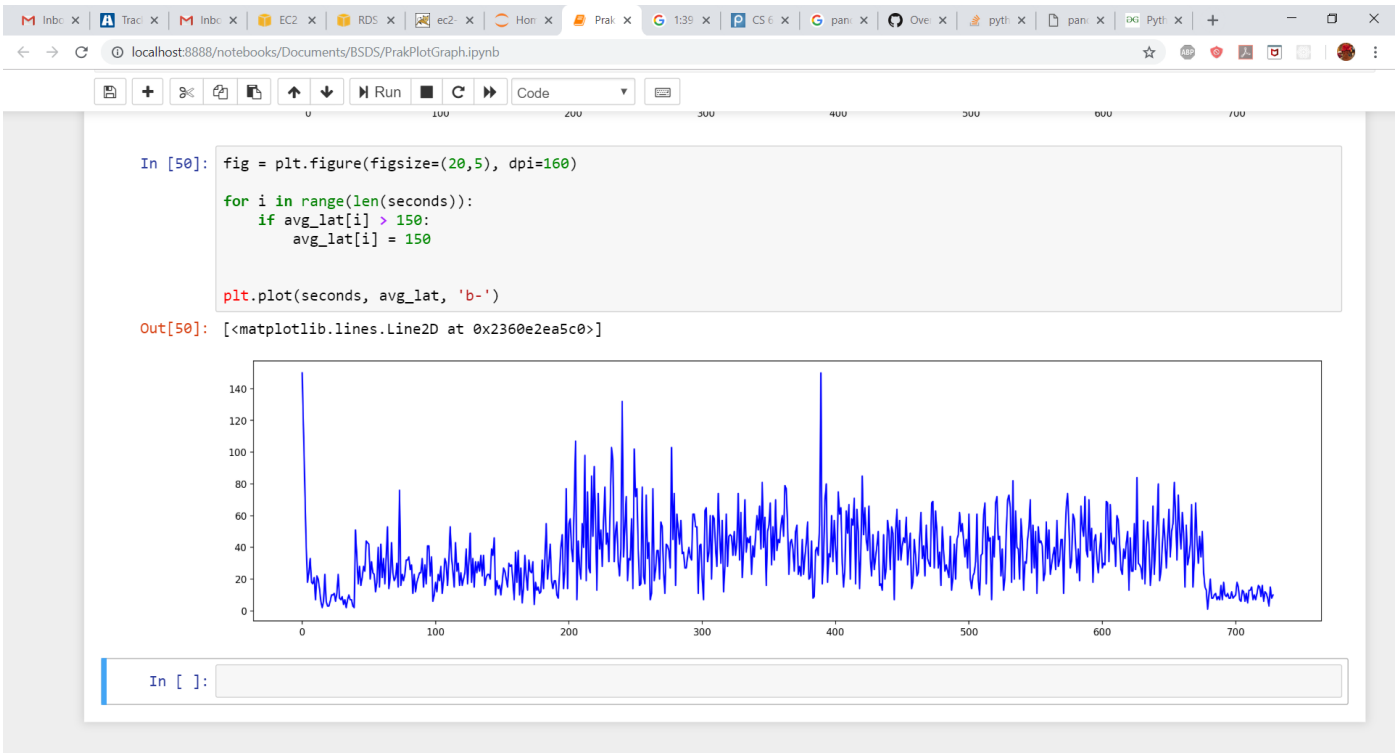

localhost:8888/notebooks/Documents/BSDS/PrakPlotGraph.ipynb
File Edit View Insert Cell Kernel Widgets Help
Trusted
Python 3
Run
Code
dt_base = min(StartTimestamp)
startTime = min(StartTimestamp)
endTime = max(EndTimestamp)
wallTime = (endTime-startTime)/1000
throughput = int(length/wallTime)
print("wallTime: ", wallTime, "seconds")
print("throughput: ", throughput)
print("median Latency: ", medianLatency, " milliseconds")
print("95th percentile latency: ", Latency[index95], "milliseconds")
print("99th percentile latency: ", Latency[index99], "milliseconds")
wallTime: 728.965 seconds
throughput: 2641
median Latency: 73 milliseconds
95th percentile latency: 149.0 milliseconds
99th percentile latency: 197.0 milliseconds
In [ ]:
In [48]:
time_dict = {}
for each_ts in StartTimestamp:
time_in_seconds = (each_ts - dt_base)//1000
if(time_in_seconds in time_dict):
time_dict[time_in_seconds] += 1
else:
time_dict[time_in_seconds] = 1
seconds = []
count = []

Throughput



Latency:

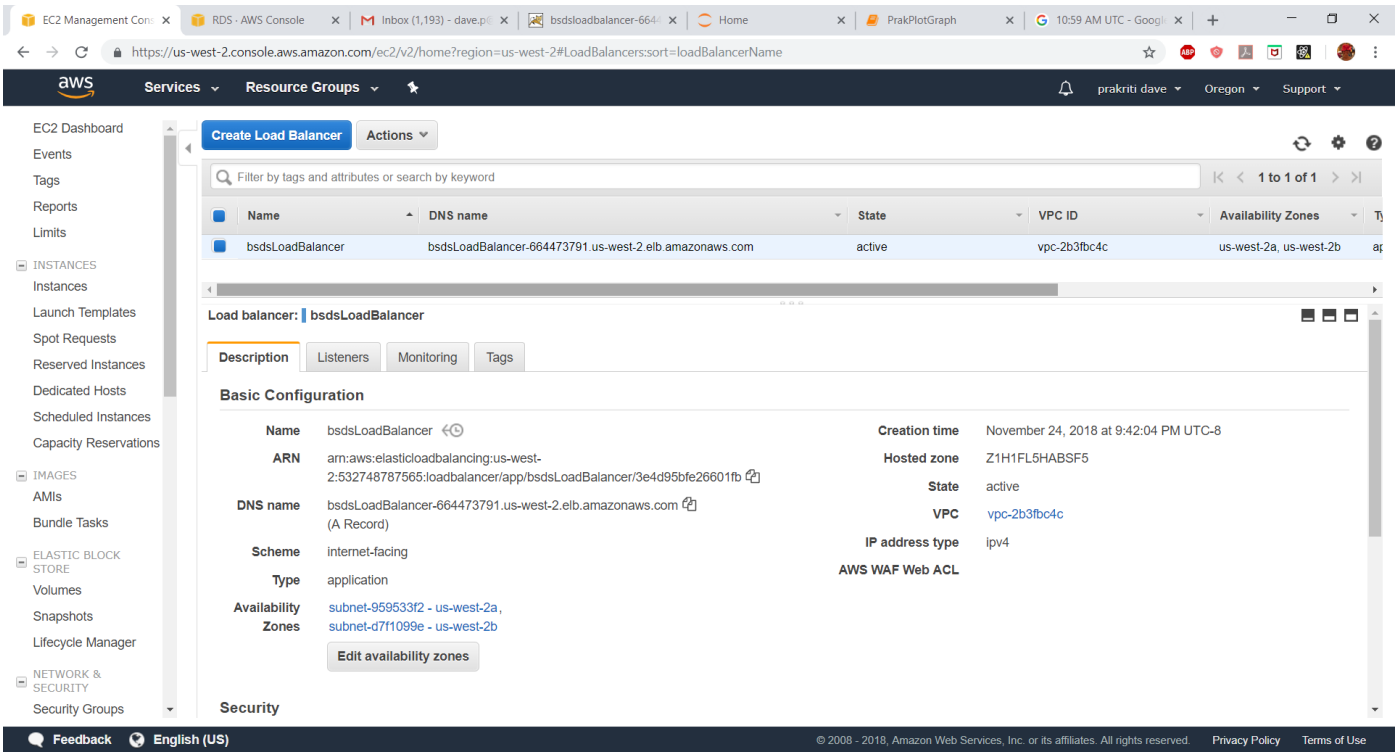




Step 4:

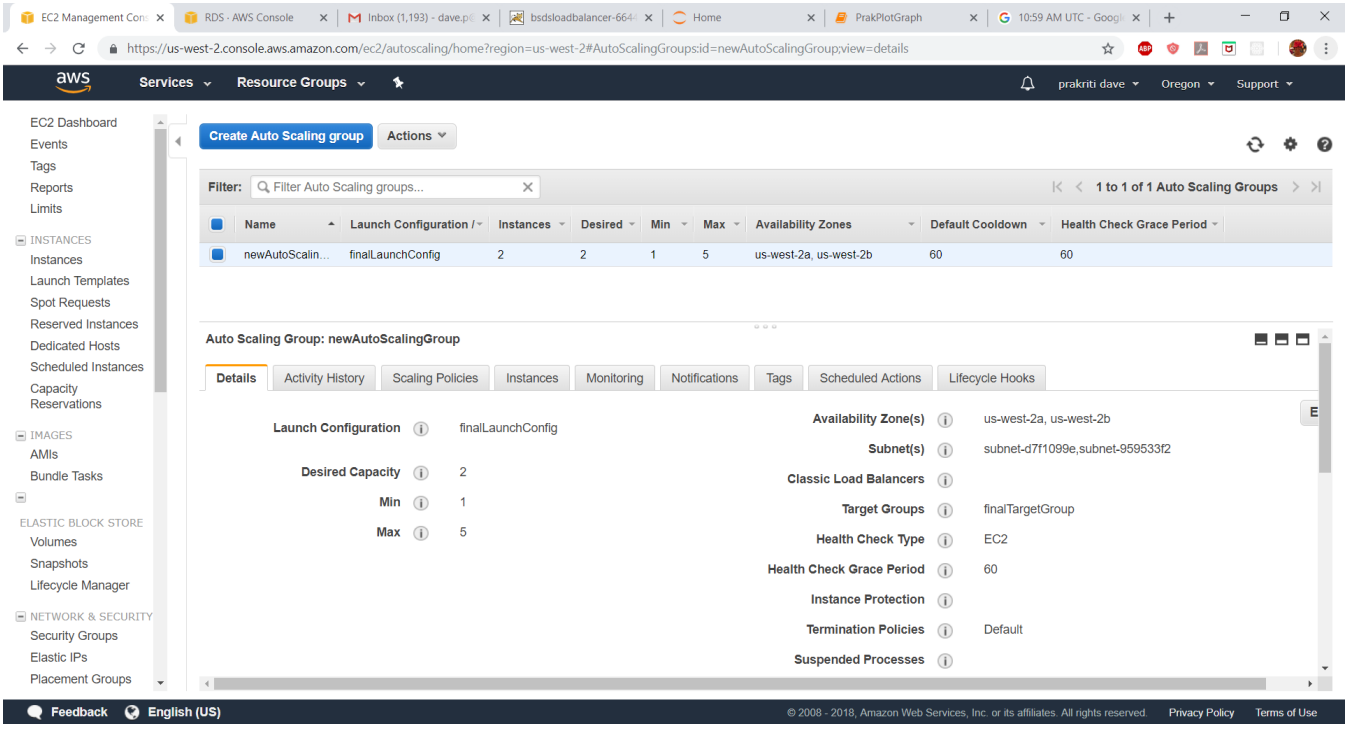
For this step I created an **AWS Load Balancer, Auto Scaling group and a Launch Configuration**

Load Balancer Configuration



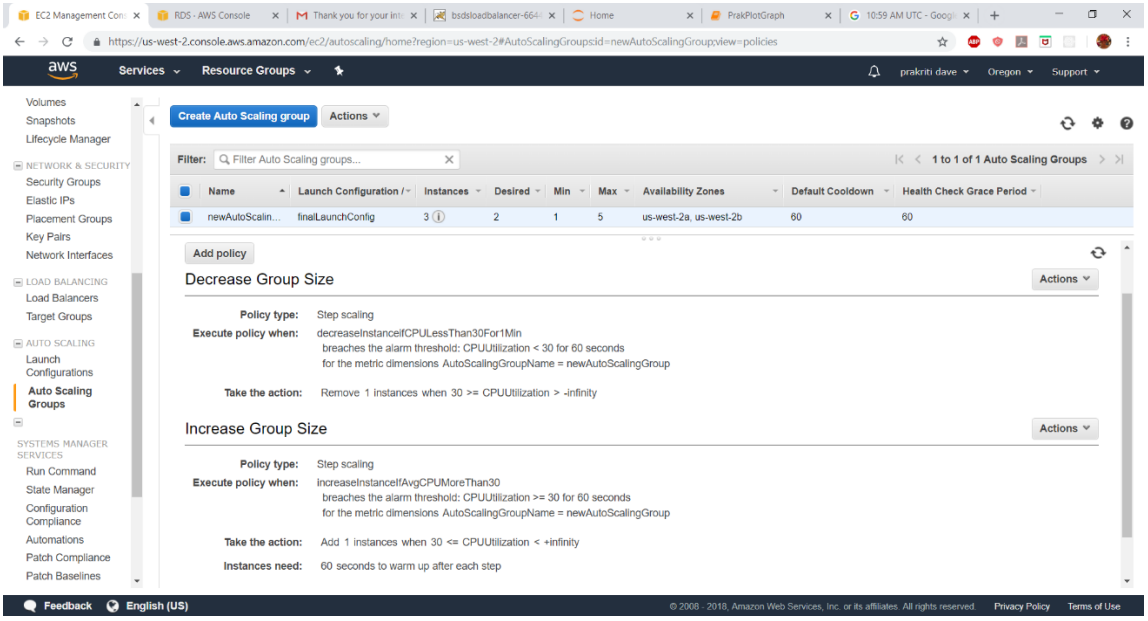
Auto Scaling Group:

I set the **min and max instances to 1 & 5** respectively.

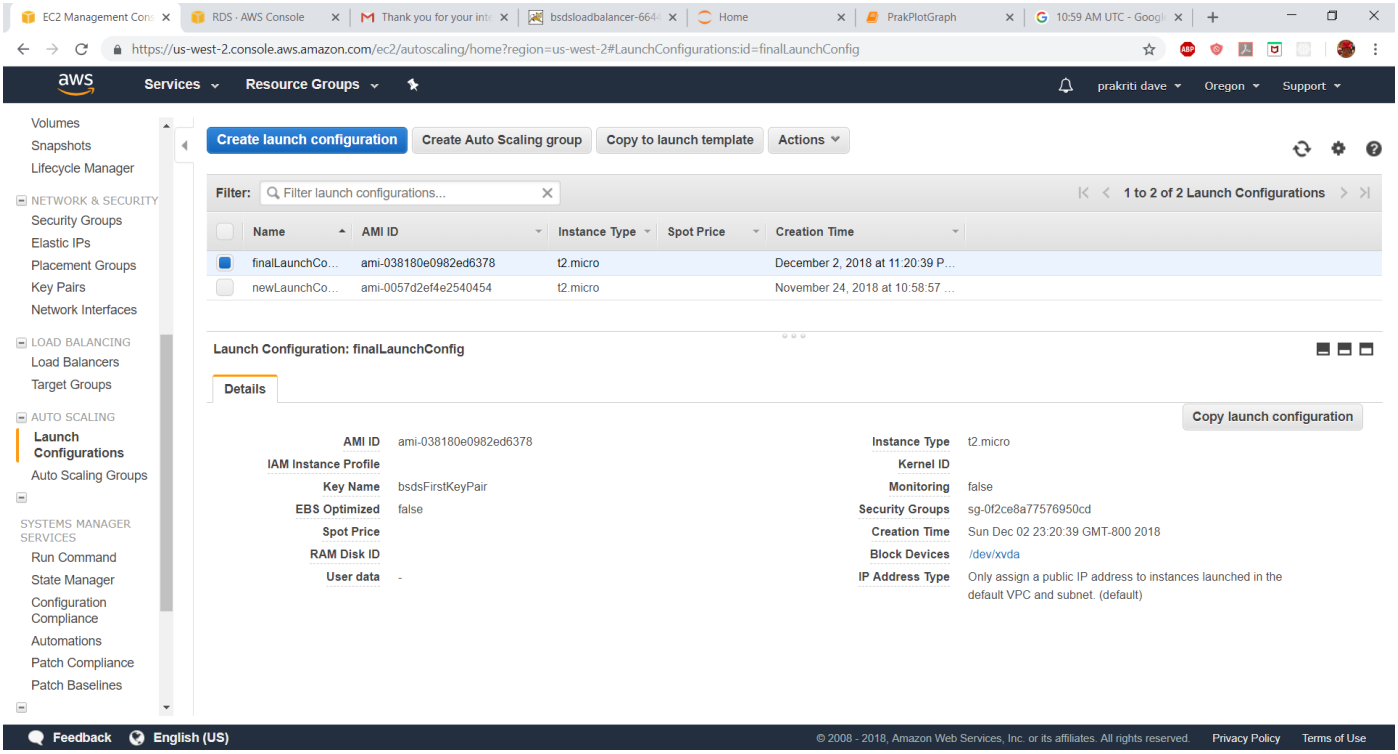


I had set the **Scaling Policies** such that if average CPU usage, for 1min was more than 40 percent, the load balancer would spawn another instance,

And if average CPU usage goes below 30 percent for a min, it would delete an instance.

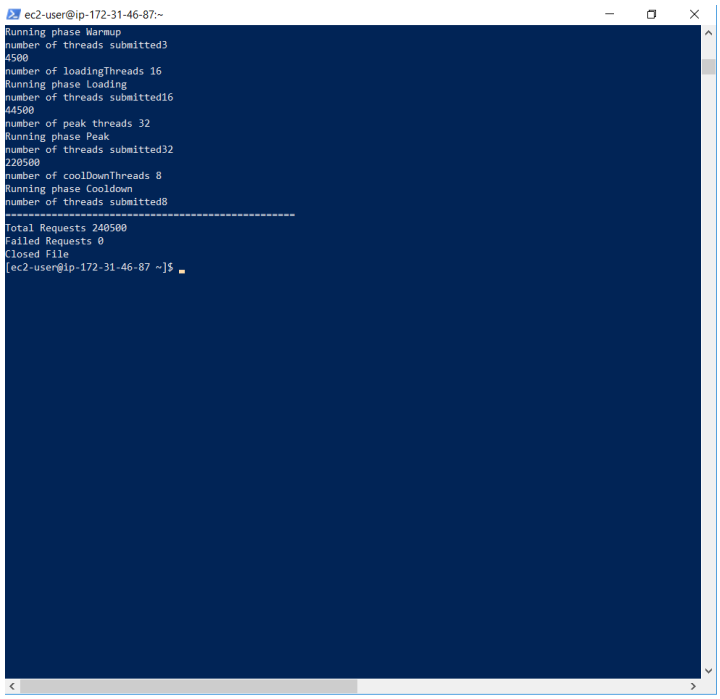


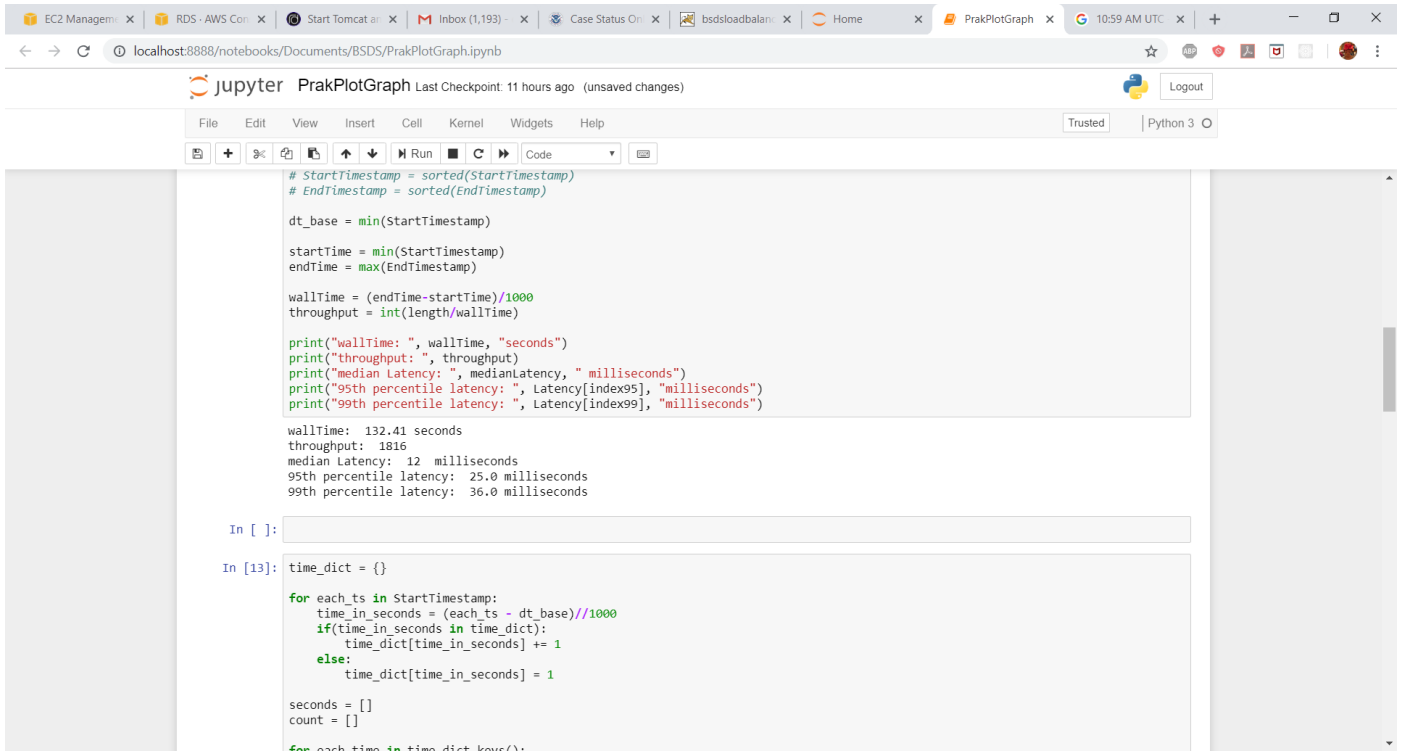
Launch Configuration



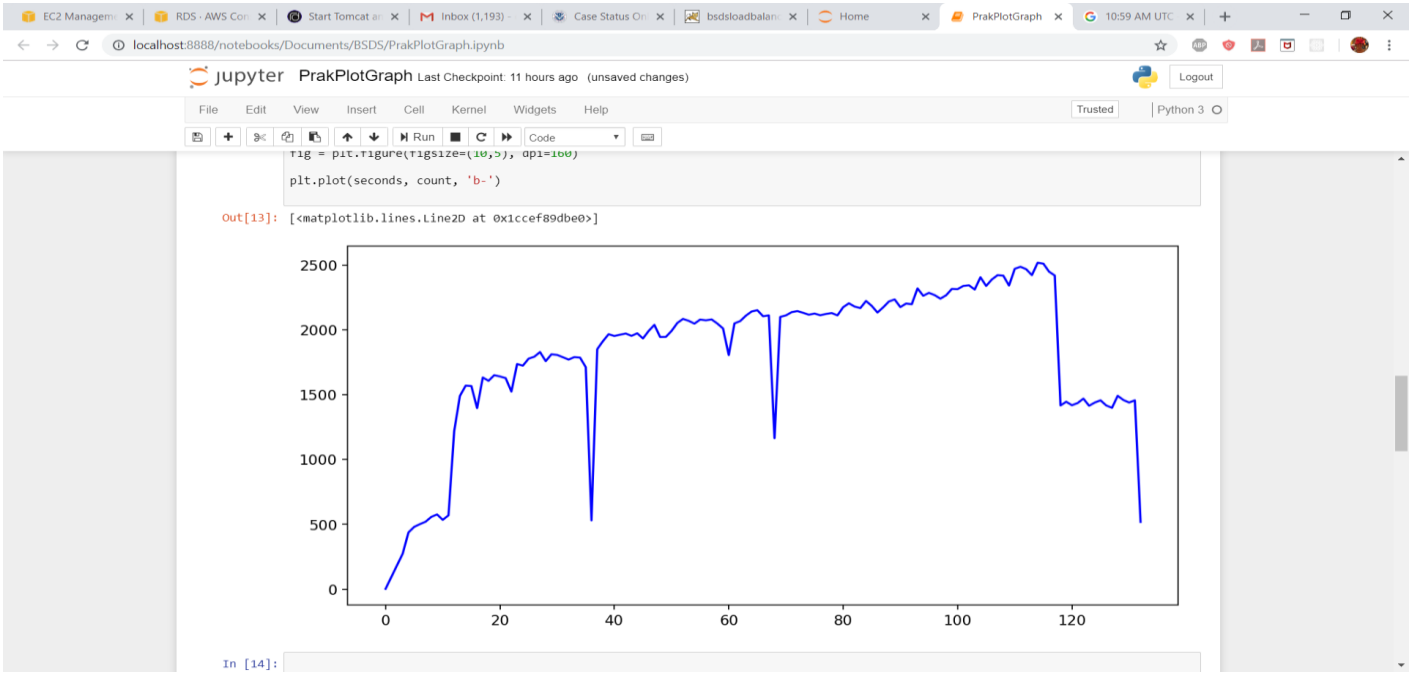
Therads {32, 64, 128, 256} with Load Balancer

32 threads with Load Balancer



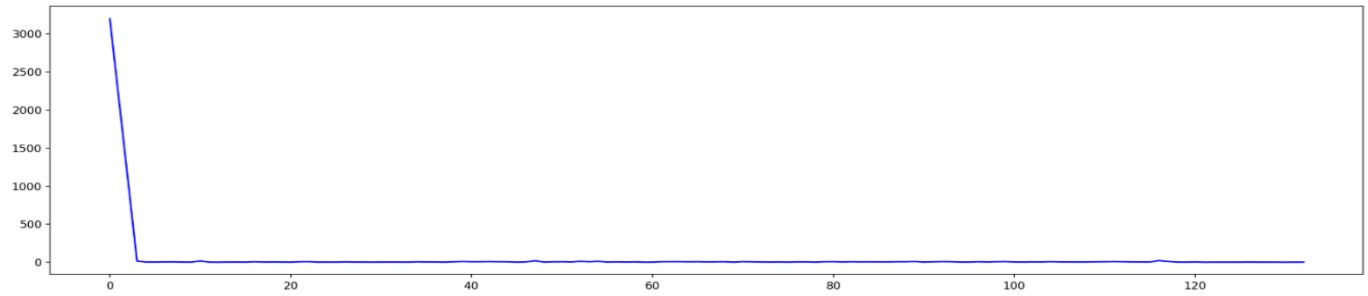


Throughput:

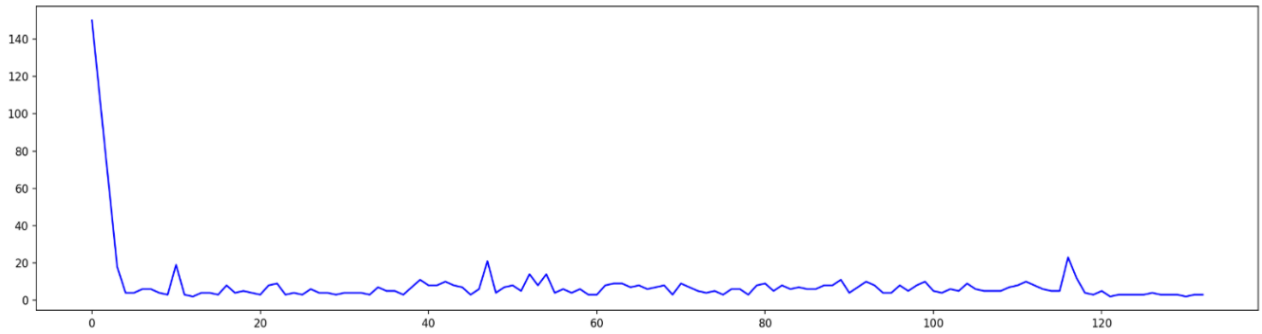


Latency:

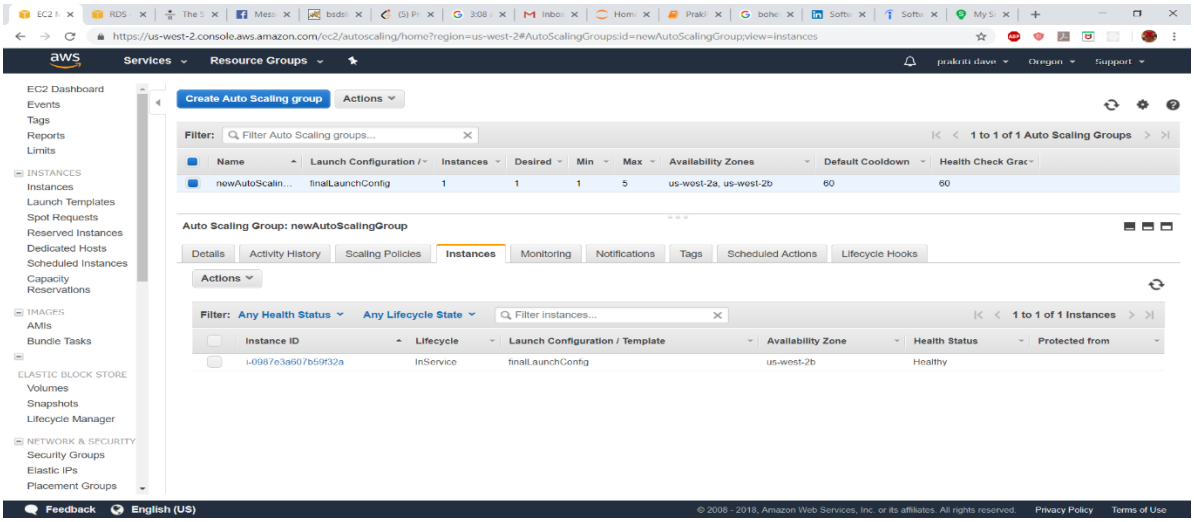
```
max latency:  3199.0
min latency:  2.0
[<matplotlib.lines.Line2D at 0x1cceea7eba8>]
```



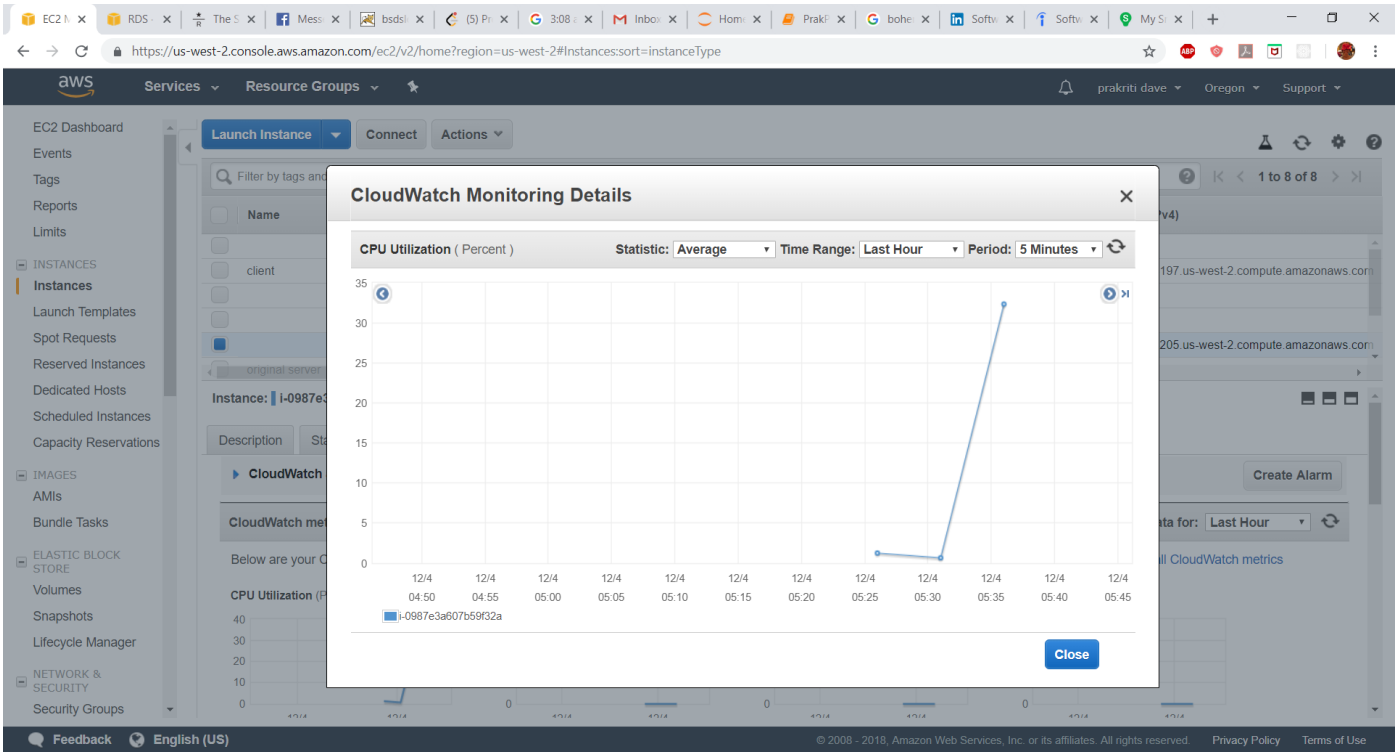

```
Out[15]: [<matplotlib.lines.Line2D at 0x1ccefd7fdd8>]
```



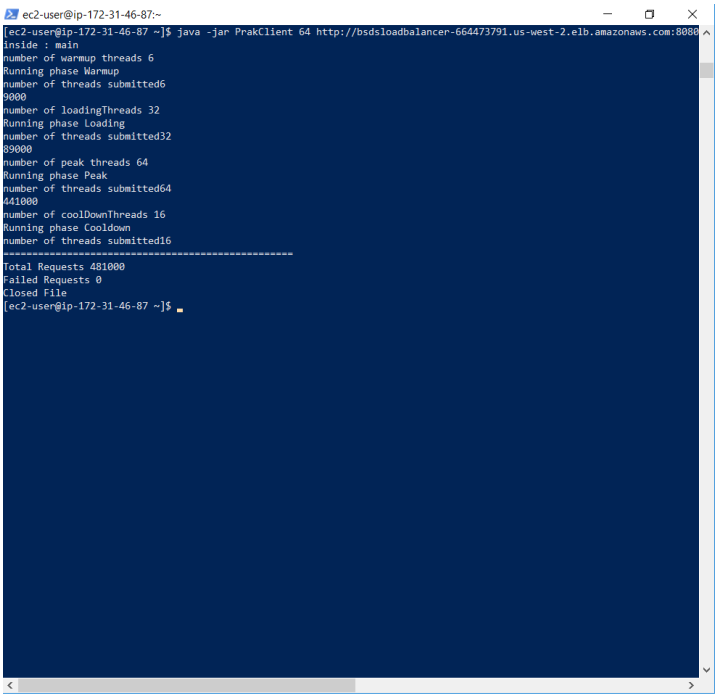
For 32 threads, only 1 instance was used

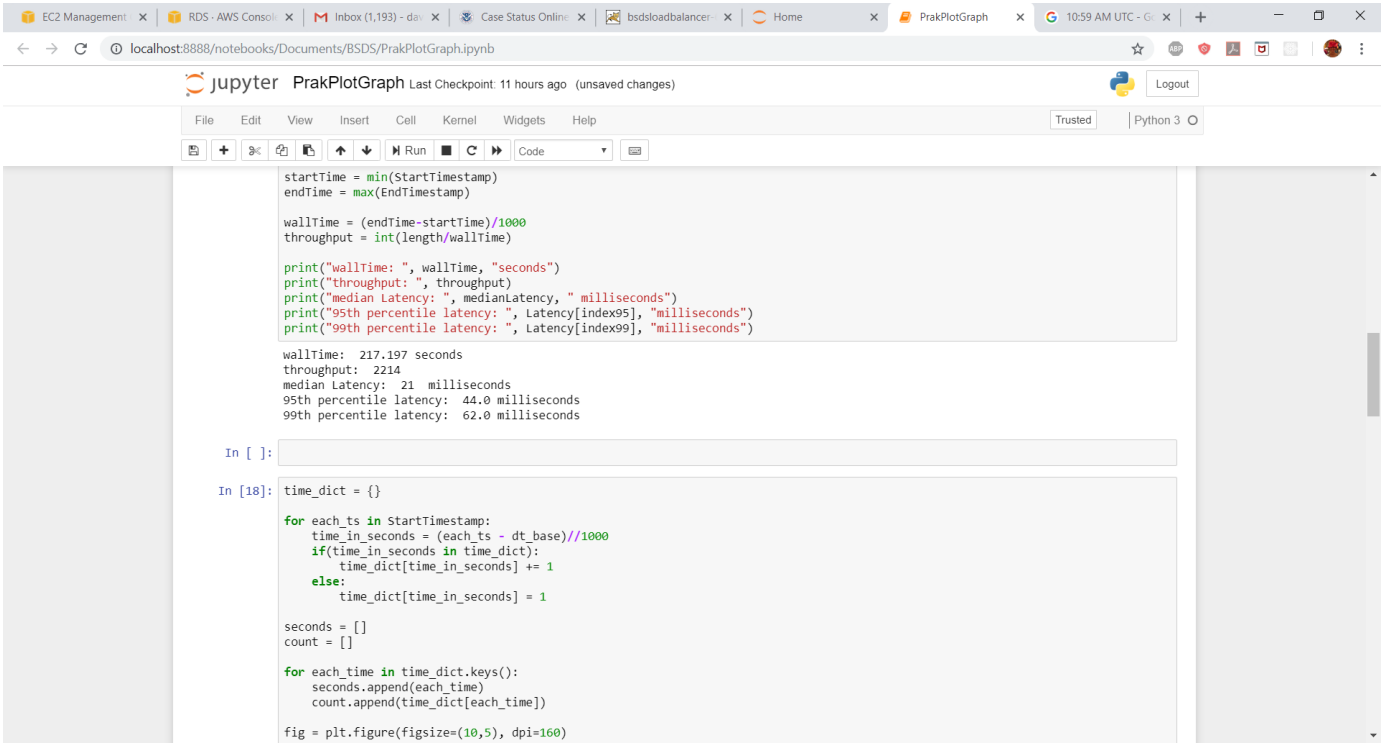


CPU utilization

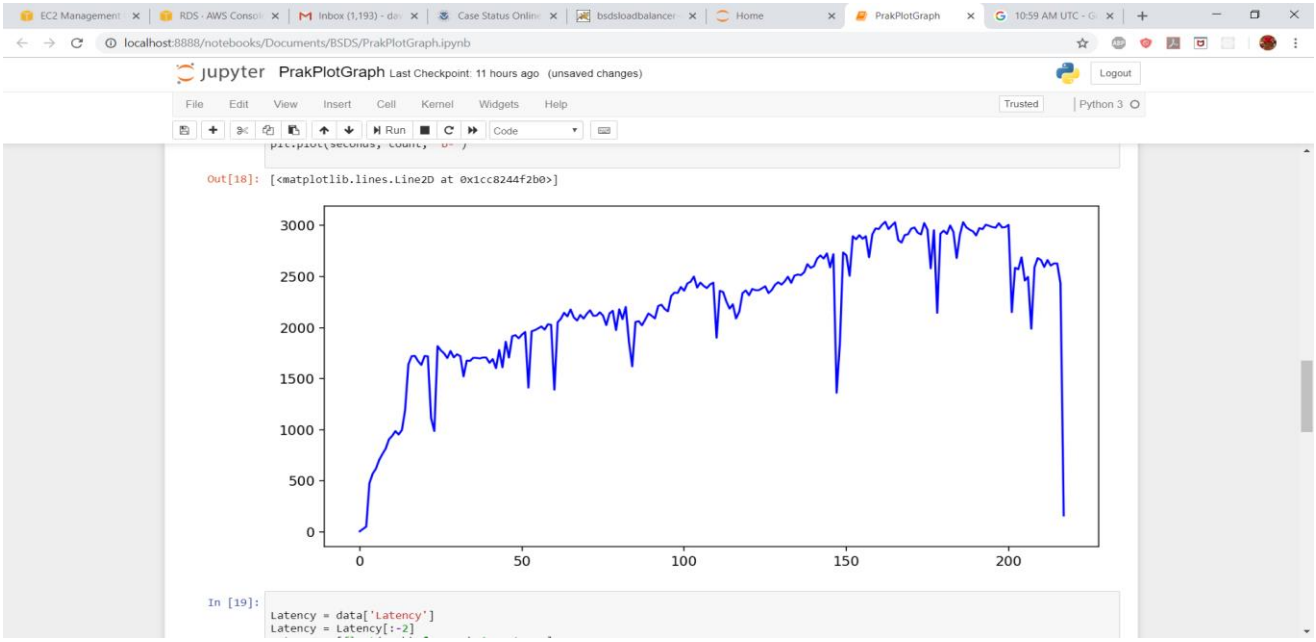


64 threads with a load balancer





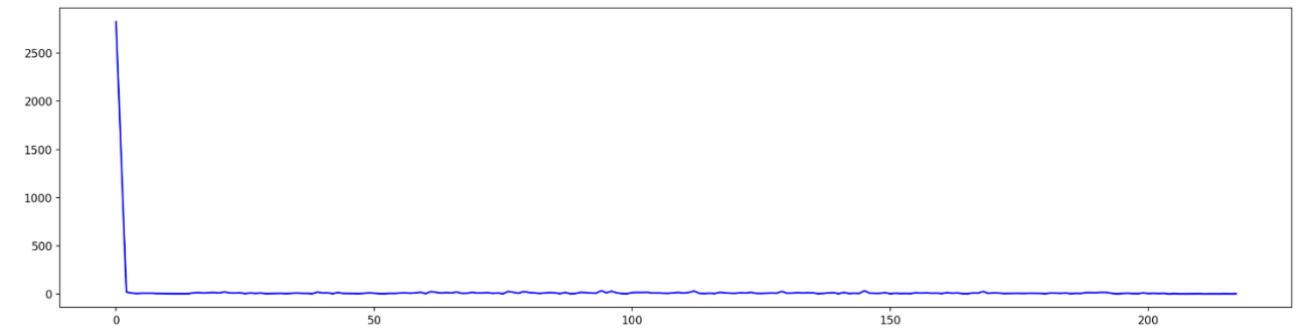
Throughput



Latency

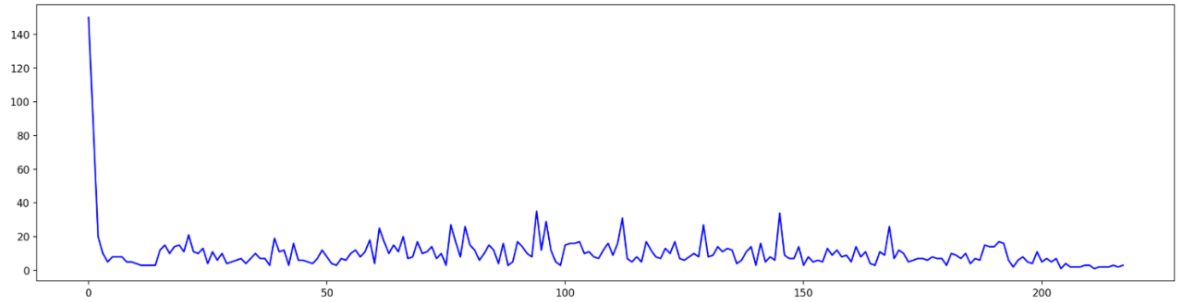
```
max latency:  2827.0
min latency:  1.0
```

```
[<matplotlib.lines.Line2D at 0x1cc80039a90>]
```

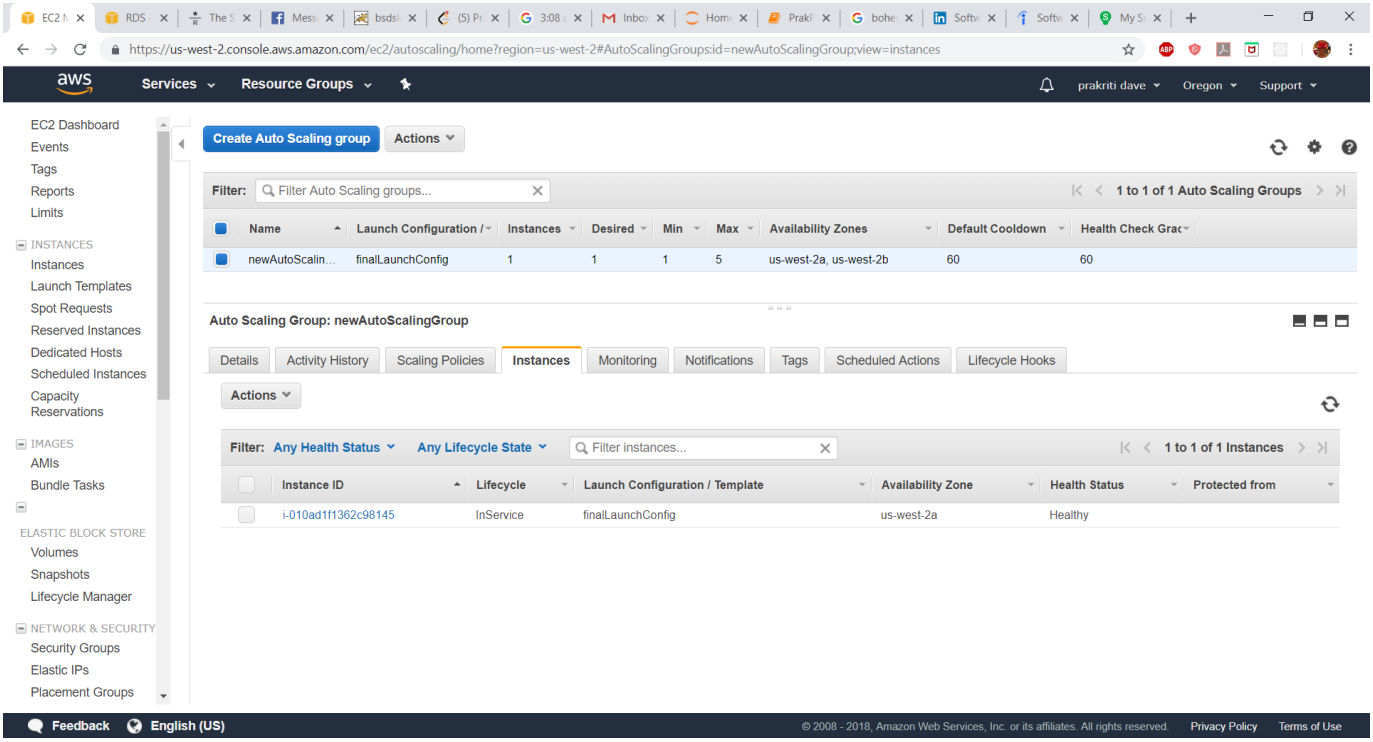



```
In [20]: fig = plt.figure(figsize=(20,5), dpi=160)

for i in range(len(seconds)):
    if avg_lat[i] > 150:
        avg_lat[i] = 150


plt.plot(seconds, avg_lat, 'b-')
```

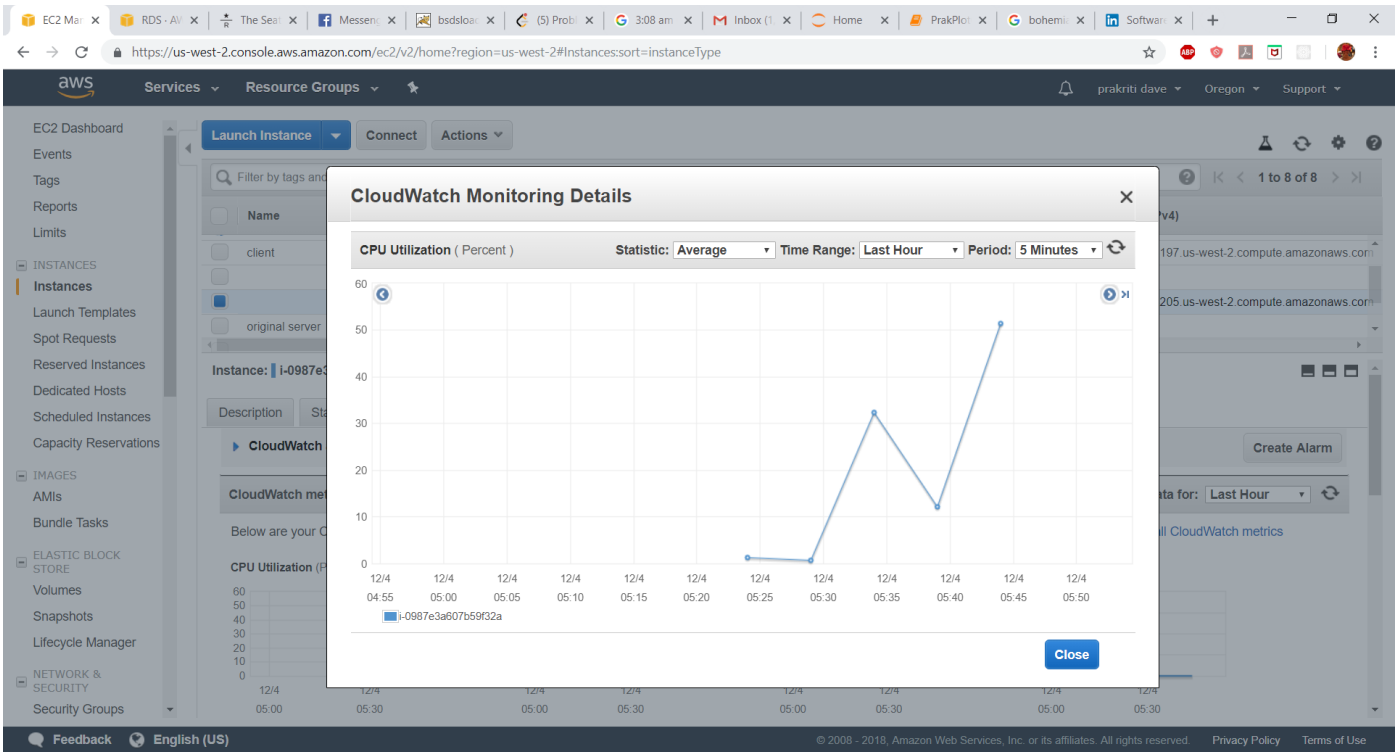
```
Out[20]: [<matplotlib.lines.Line2D at 0x1cc8019ad68>]
```



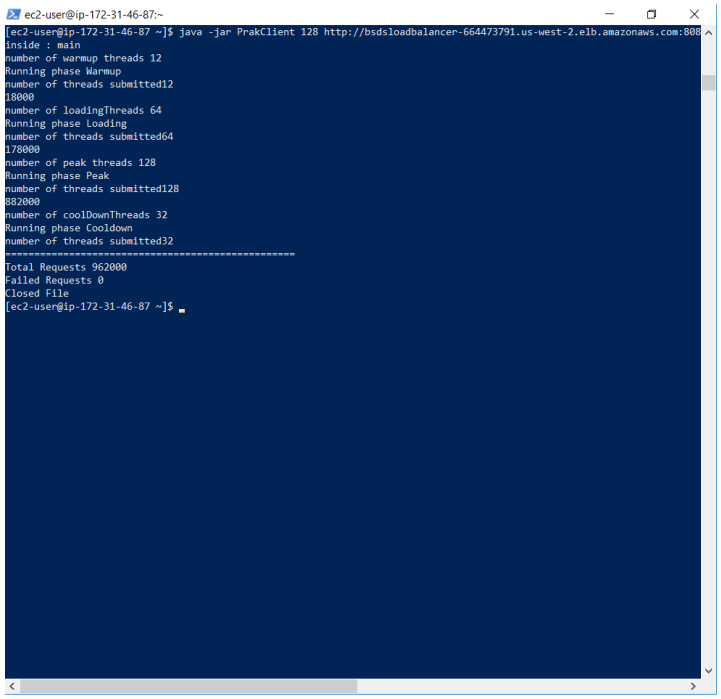
For 64 threads, only 1 CPU was used

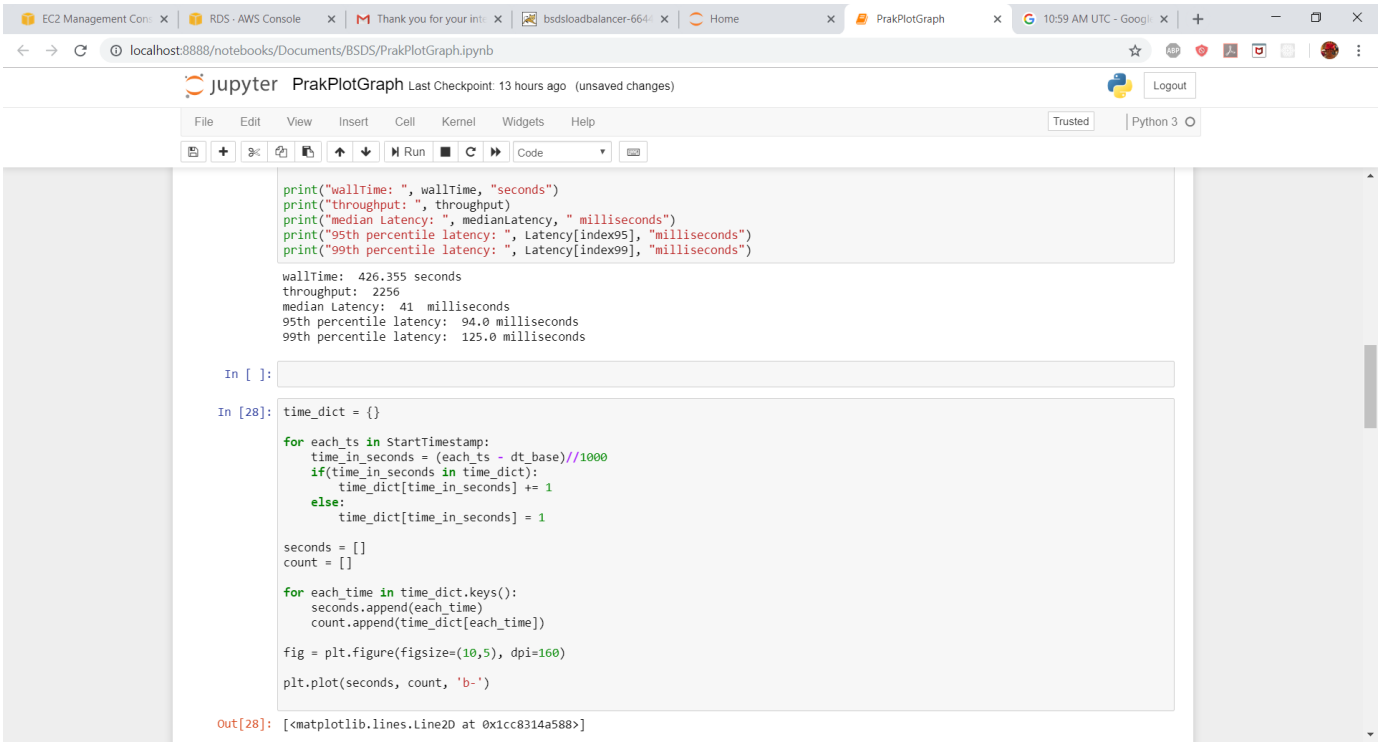


CPU instances:

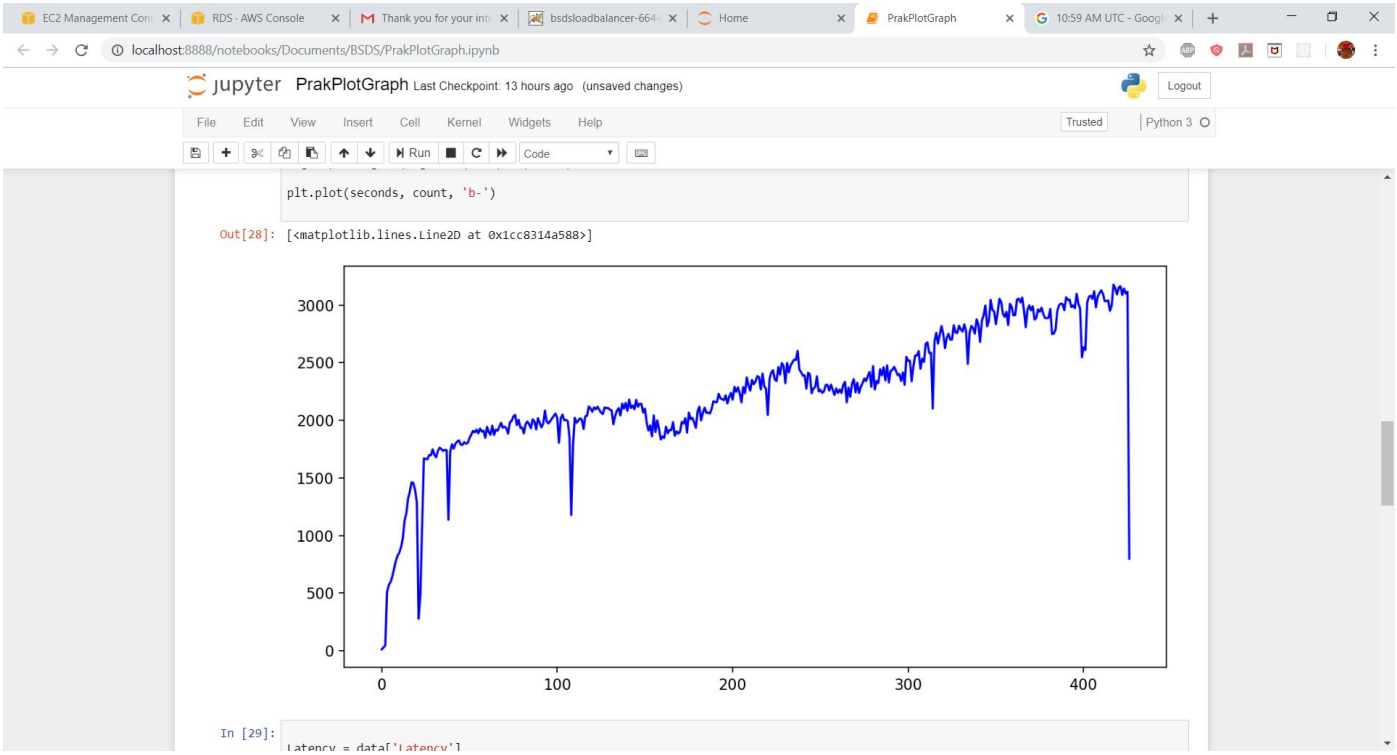


128 threads with Load Balancer:

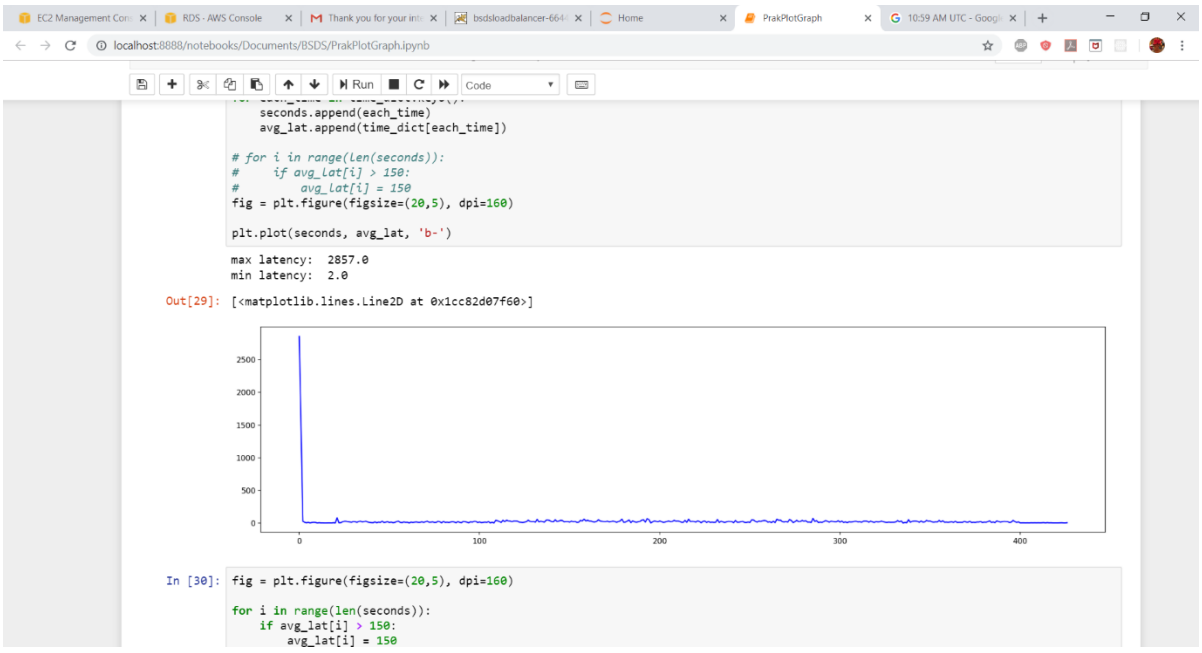




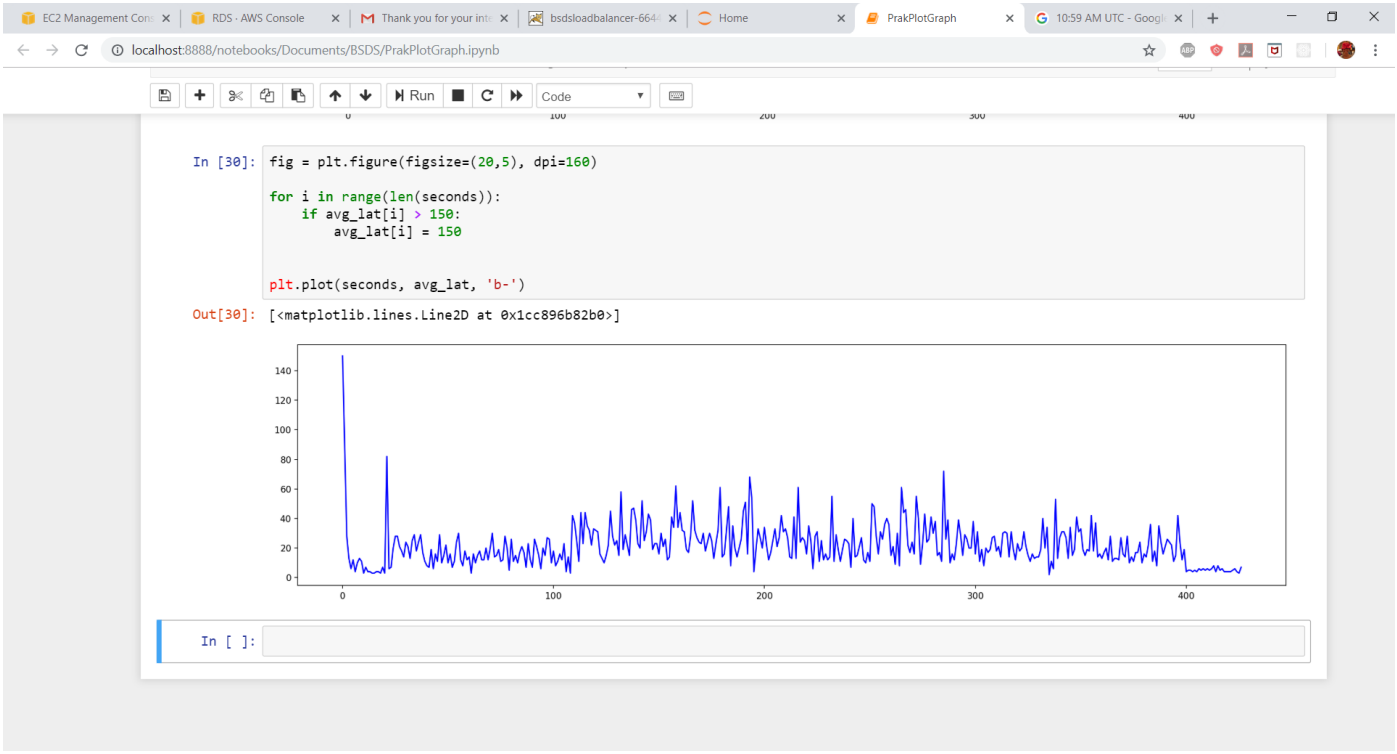
Throughput:



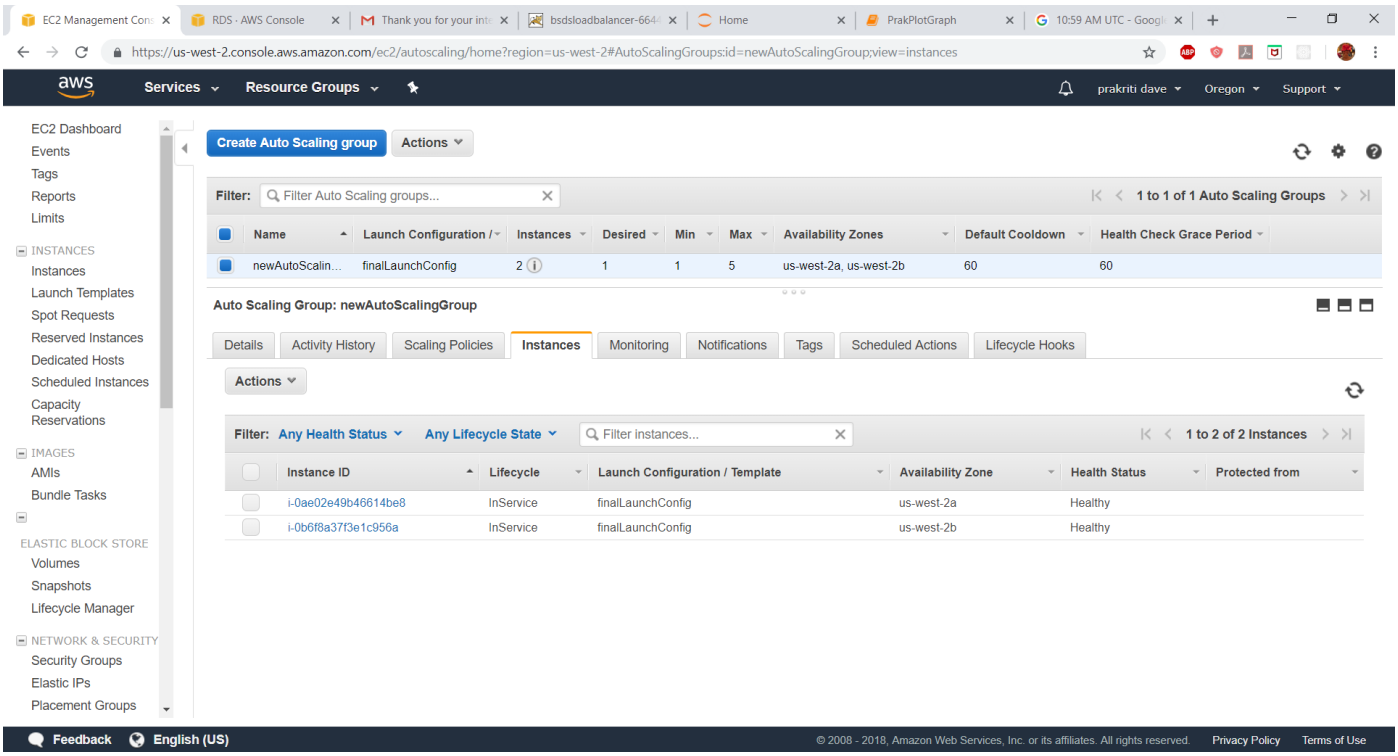
Latency plot:



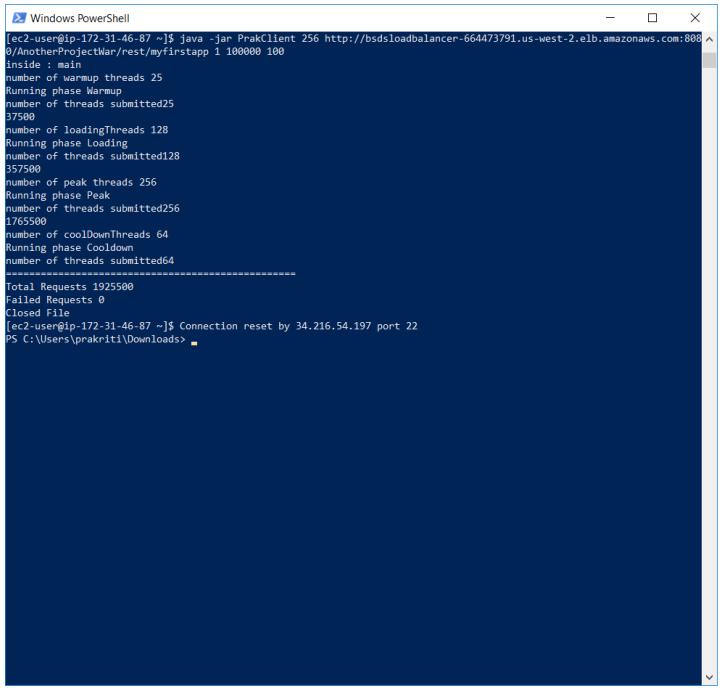
Latencies averaged to 150

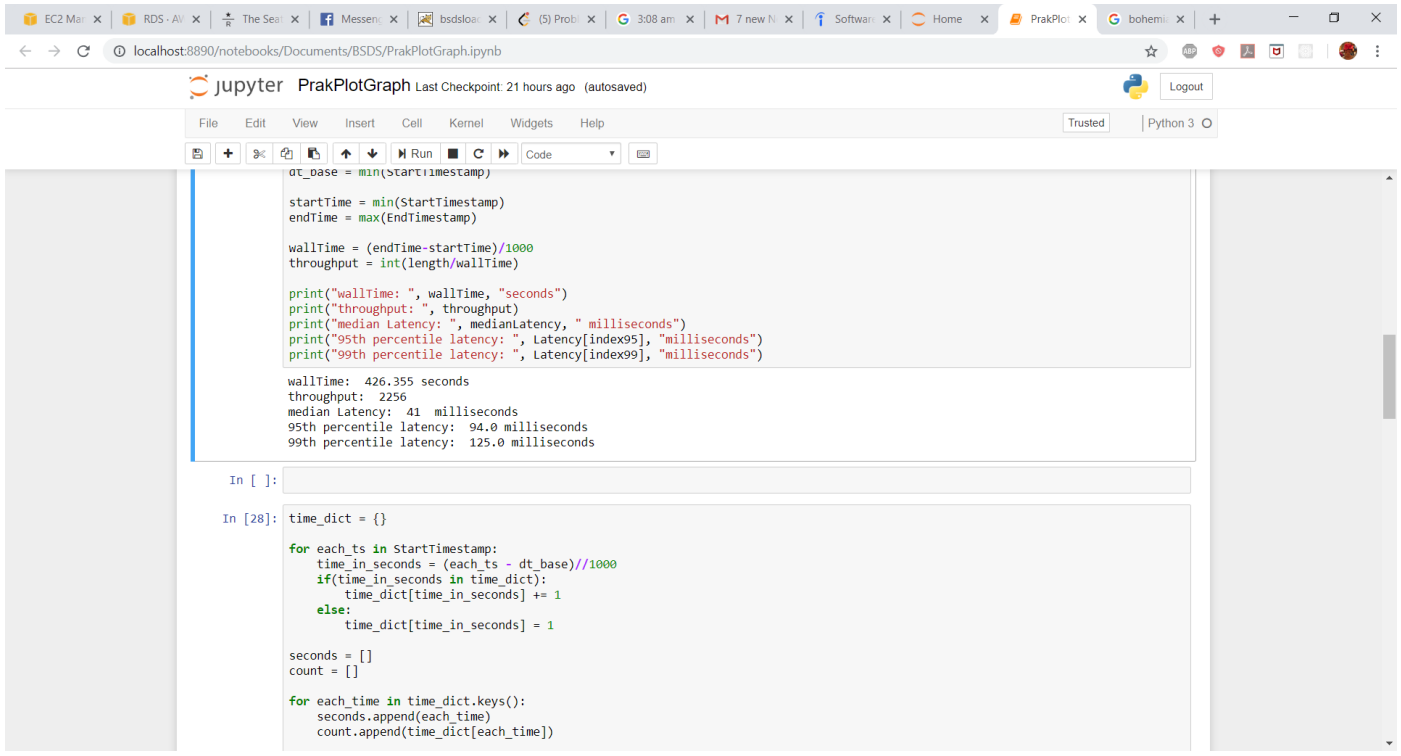


For 128 threads, 2 instances were spawned

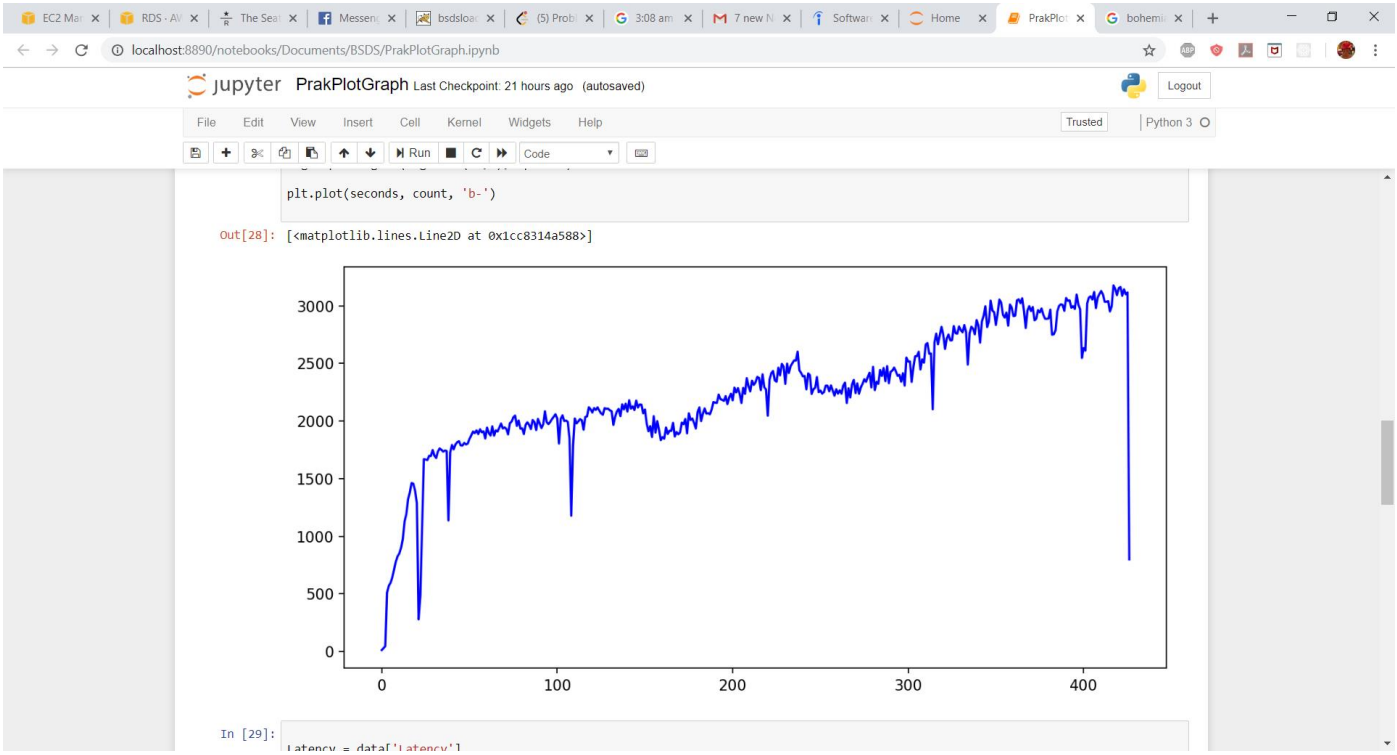


256 Threads with Load Balancer:

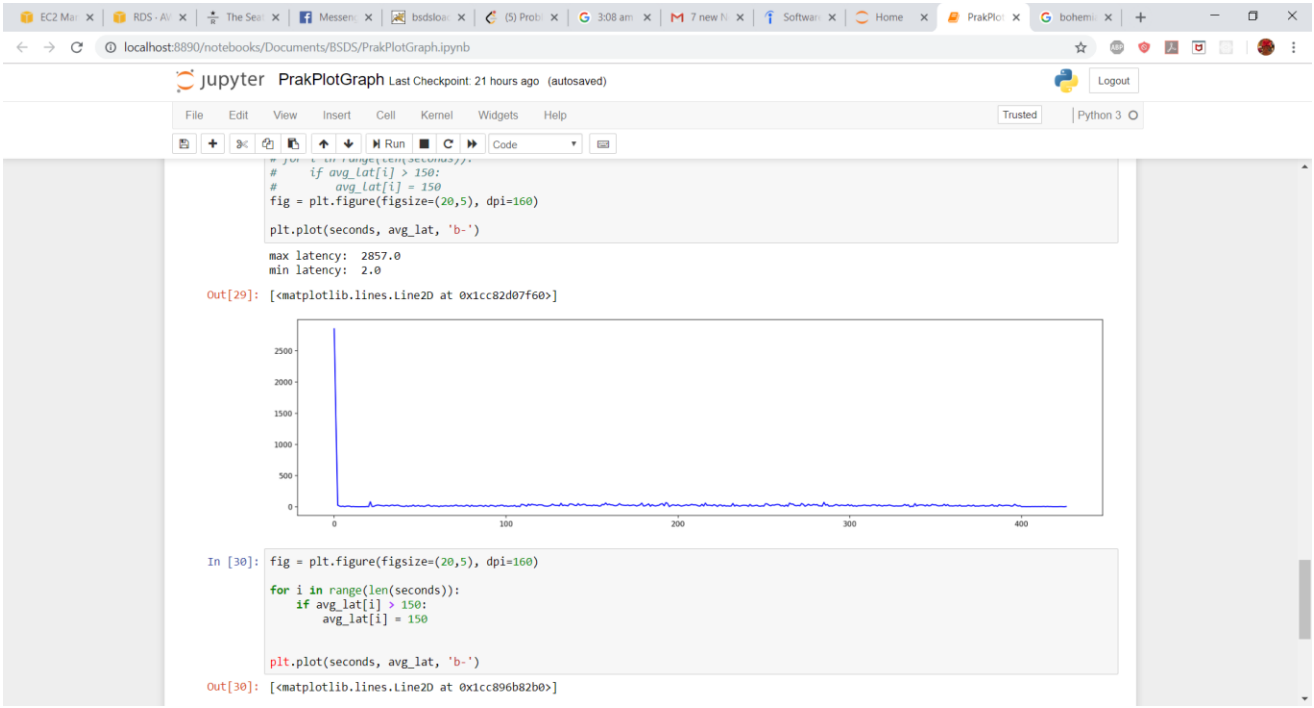




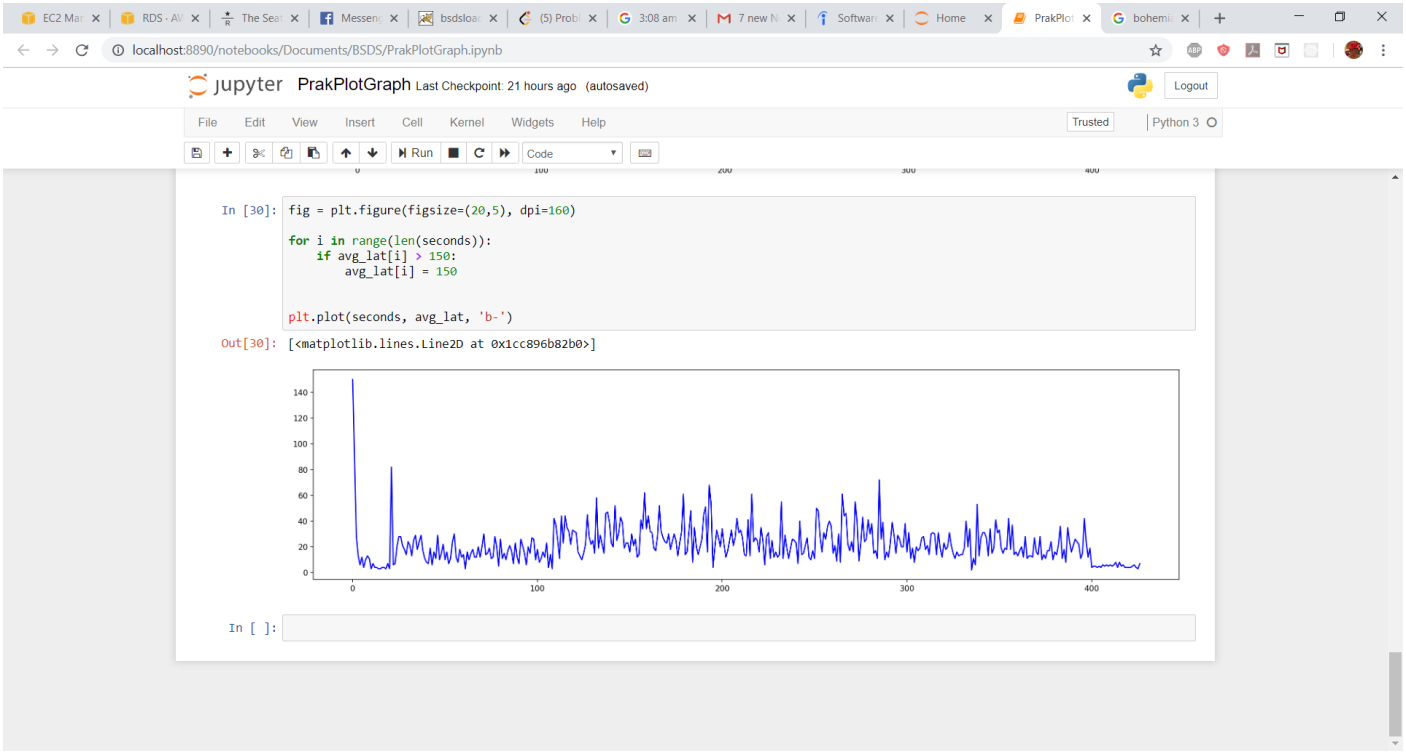
Throughput:



Latency:

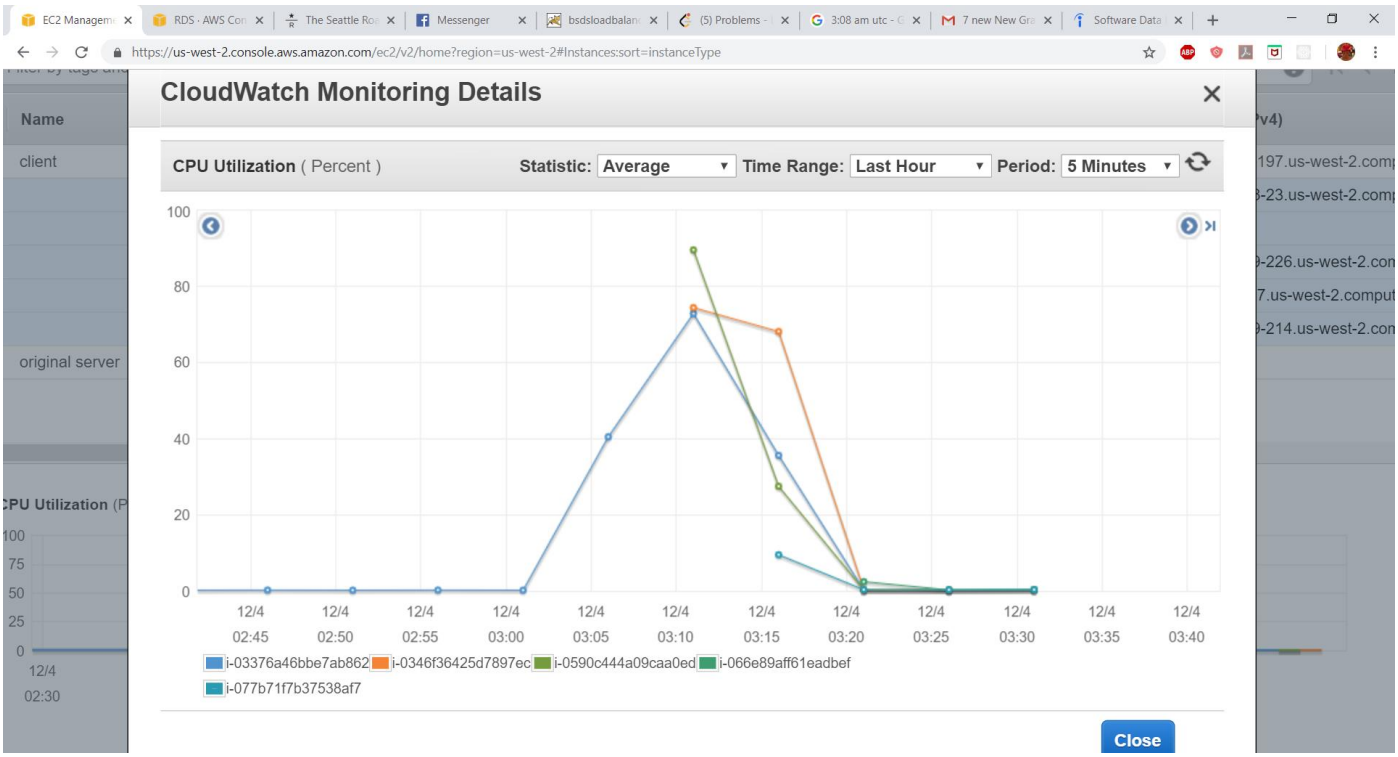


Latency averaged to 150

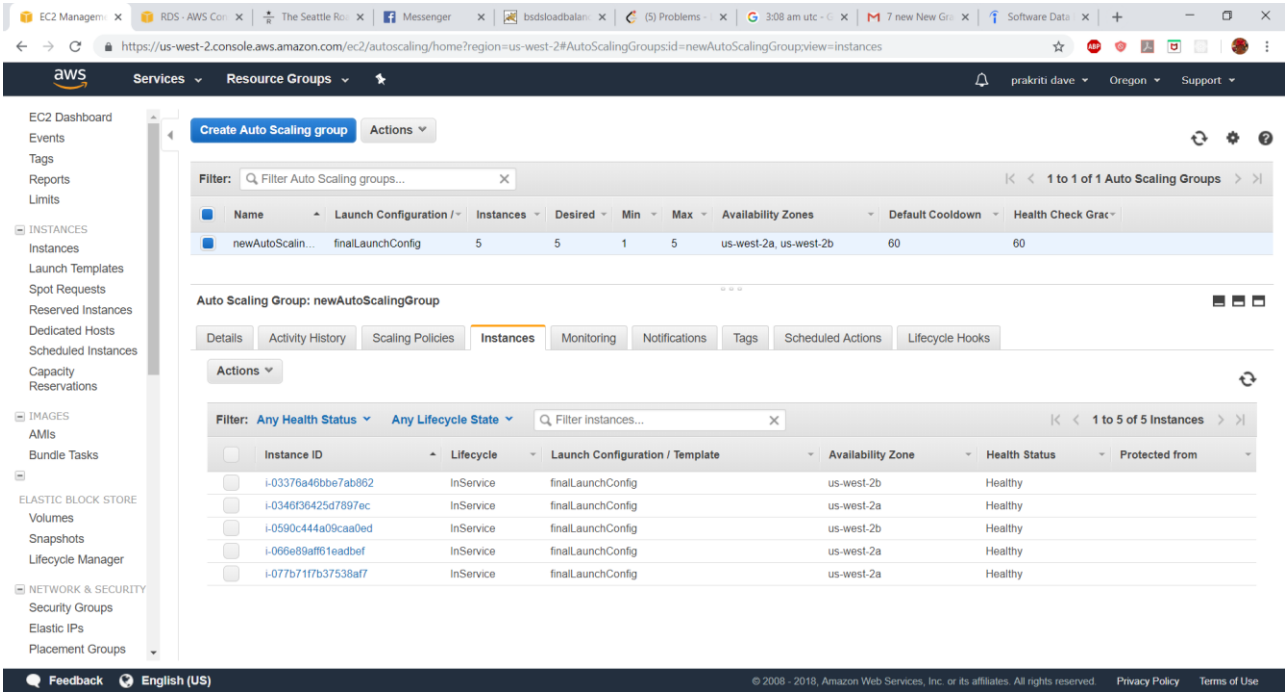


For 256 threads, all 5 instances were spawned, 4 were running during the peak phase, and at the end of peak 5th instance was spawned

CPU utilization of 5 instances

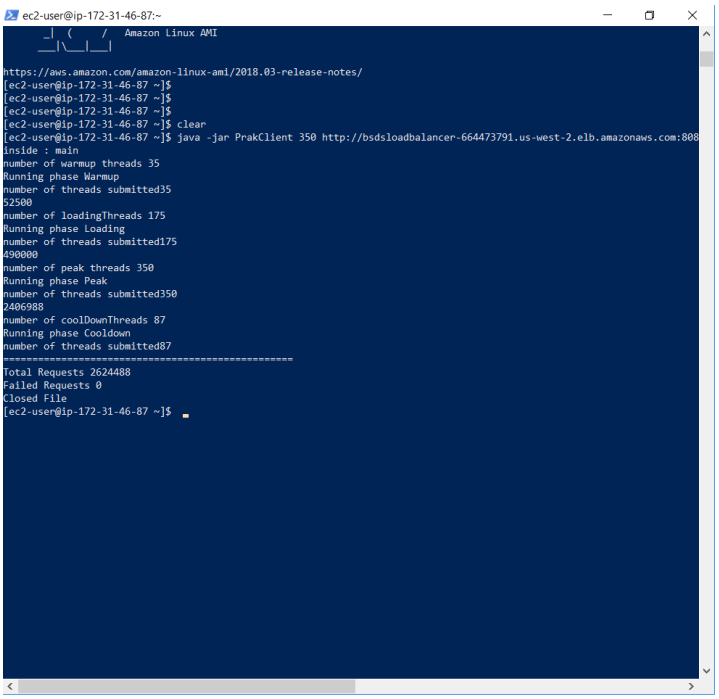


Instances Launched at Peak Phase:



Step 5: Load testing

Load testing with 350 threads:

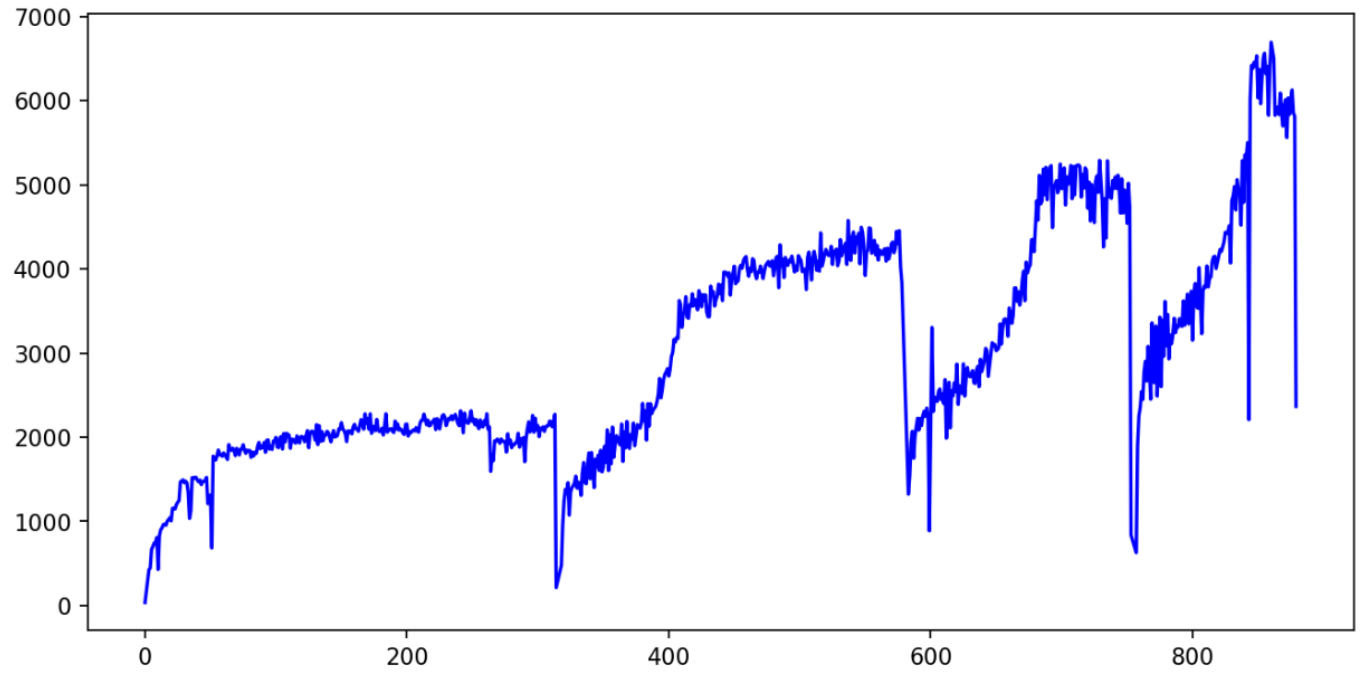


```
print("wallTime: ", wallTime, "seconds")
print("throughput: ", throughput)
print("median Latency: ", medianLatency, " milliseconds")
print("95th percentile latency: ", Latency[index95], "milliseconds")
print("99th percentile latency: ", Latency[index99], "milliseconds")
```

```
wallTime:  879.672 seconds
throughput:  2983
median Latency:  24  milliseconds
95th percentile latency:  321.0 milliseconds
99th percentile latency:  573.0 milliseconds
```

:

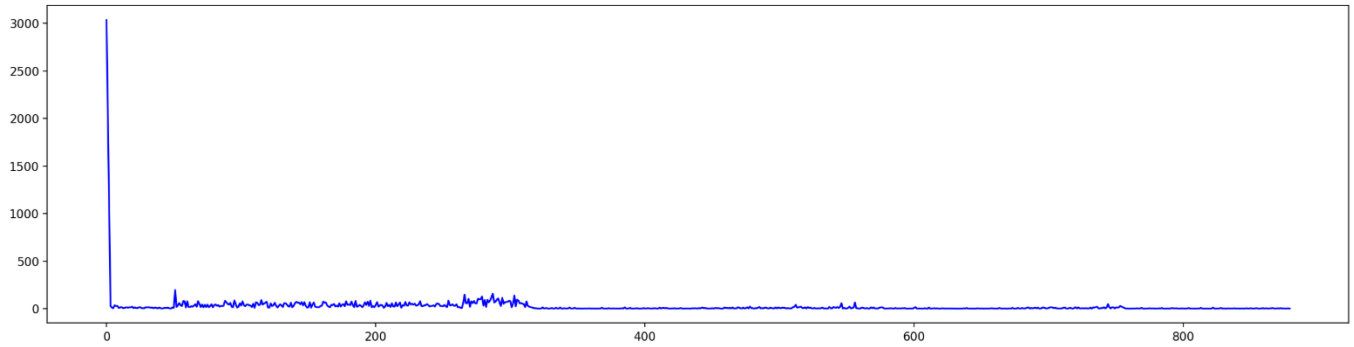
Throughput:



Latency:

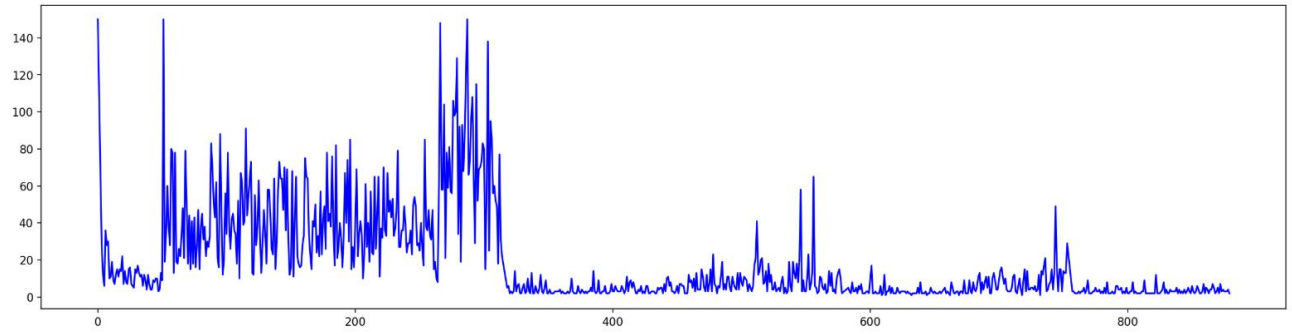
```
max latency:  5902.0
min latency:  1.0
```

```
[<matplotlib.lines.Line2D at 0x1fa28cd0630>]
```

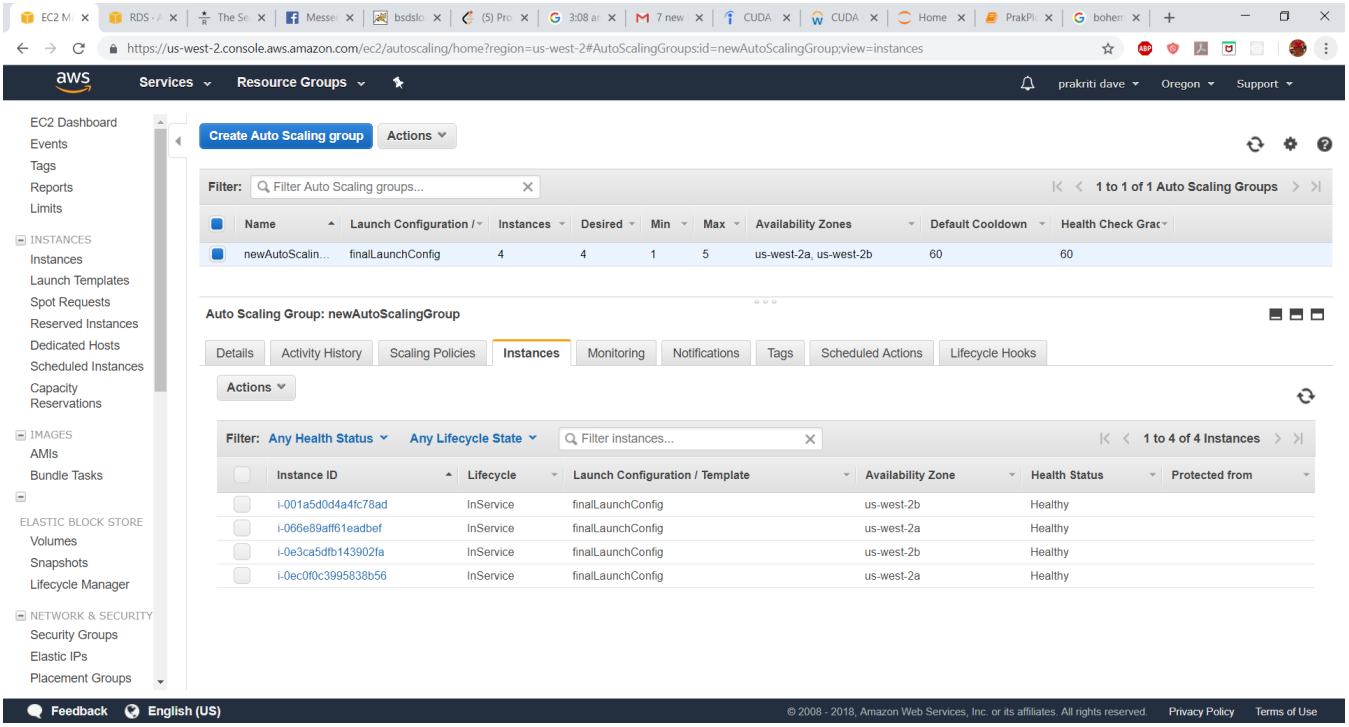


Latency averaged to 150:

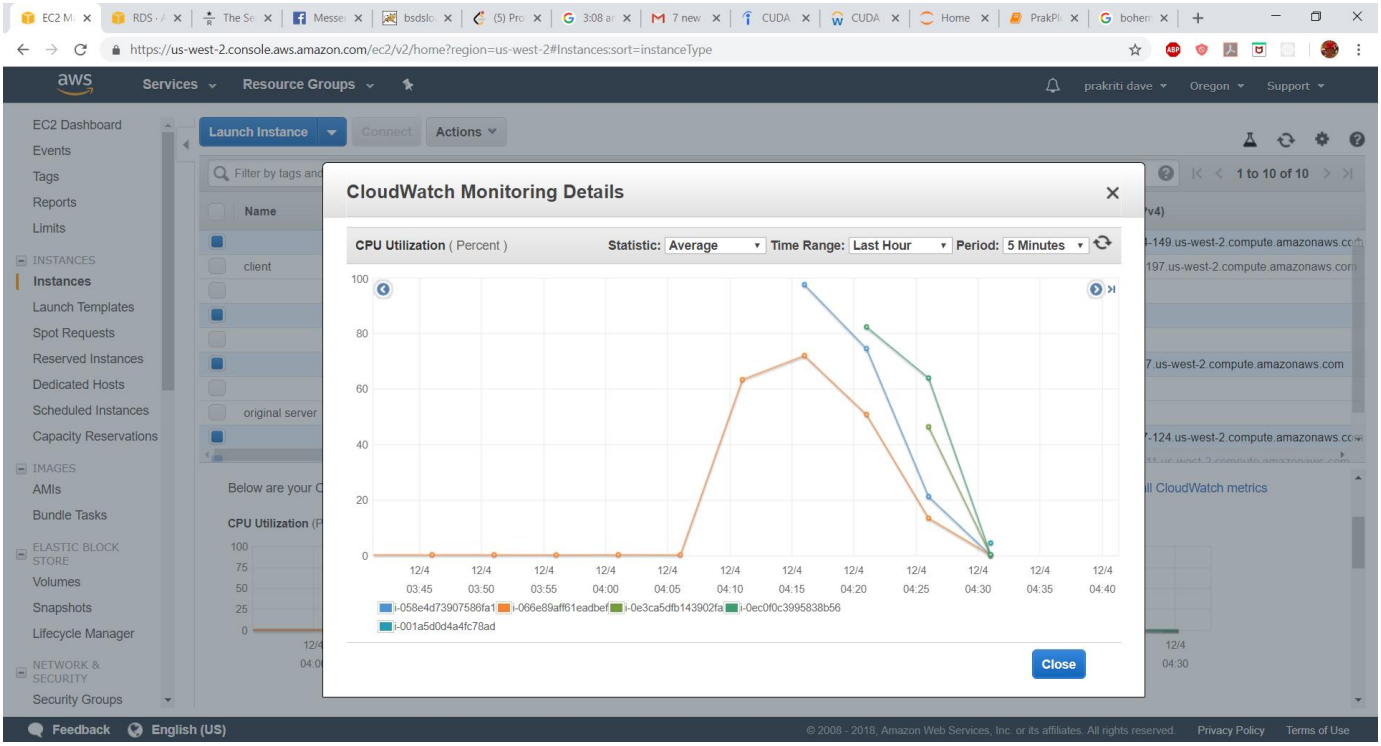
```
Out[5]: [<matplotlib.lines.Line2D at 0x1fa1499c940>]
```



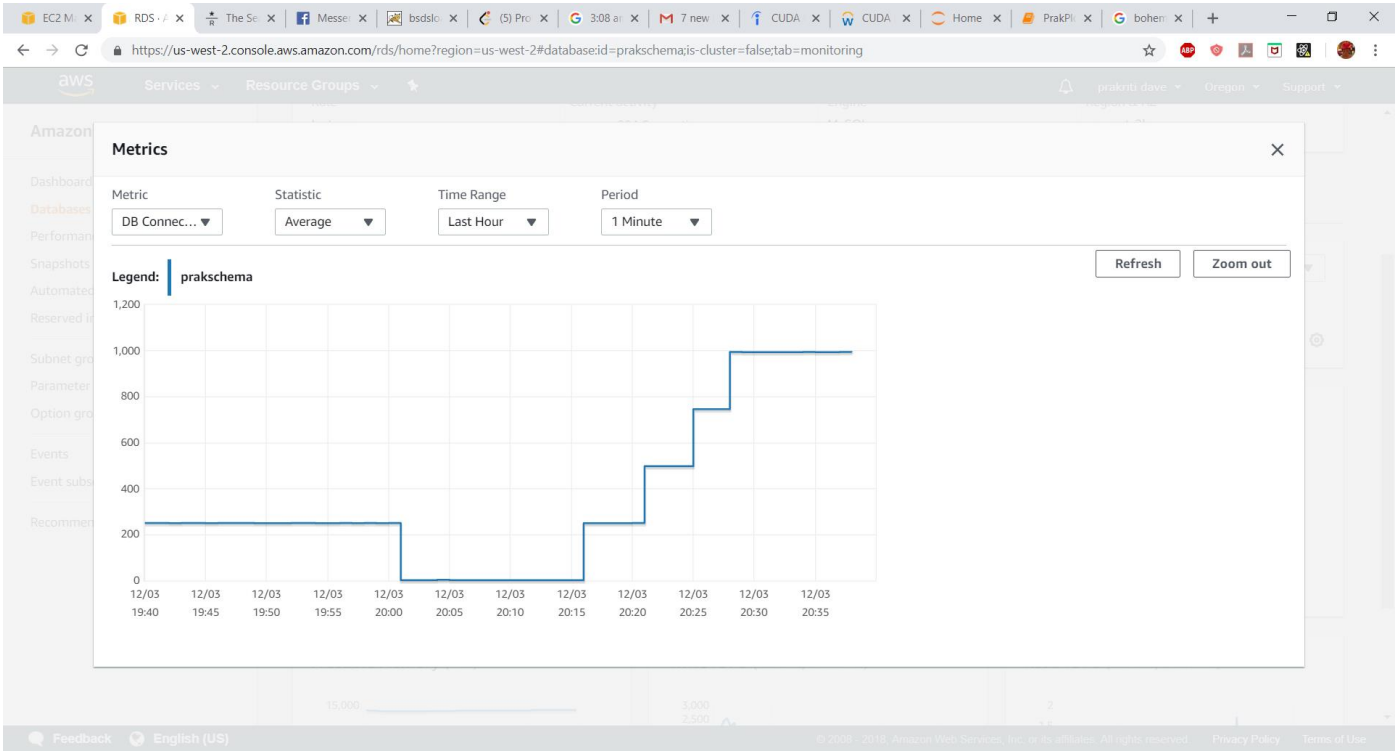
All 5 instances were spawned for 350 threads, during the Peak Phase



CPU utilization :

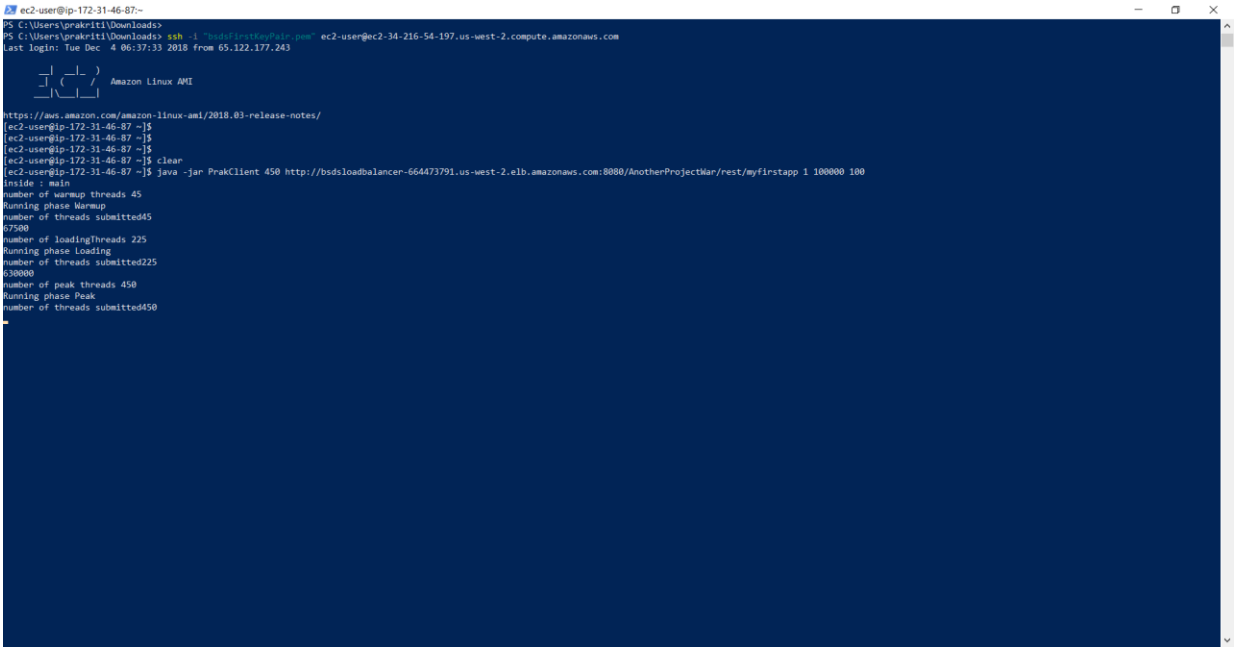


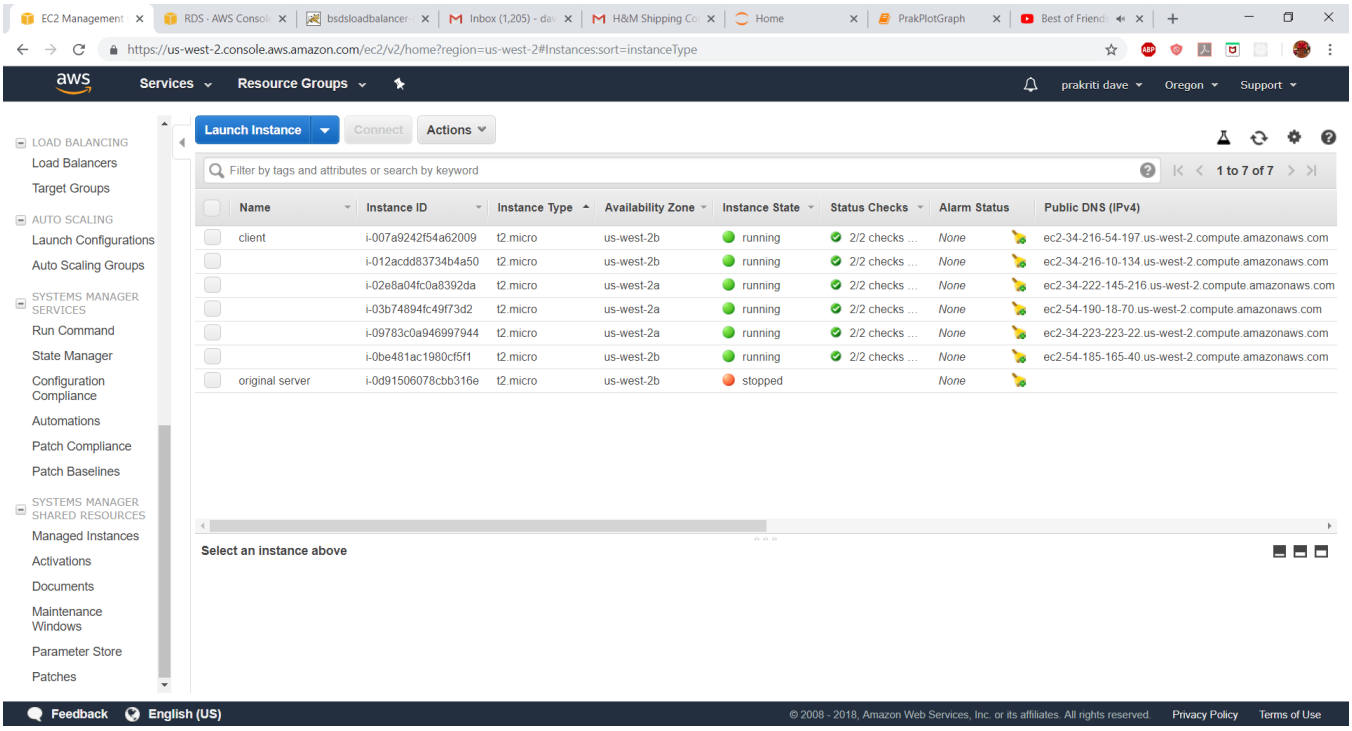
RDS connections:



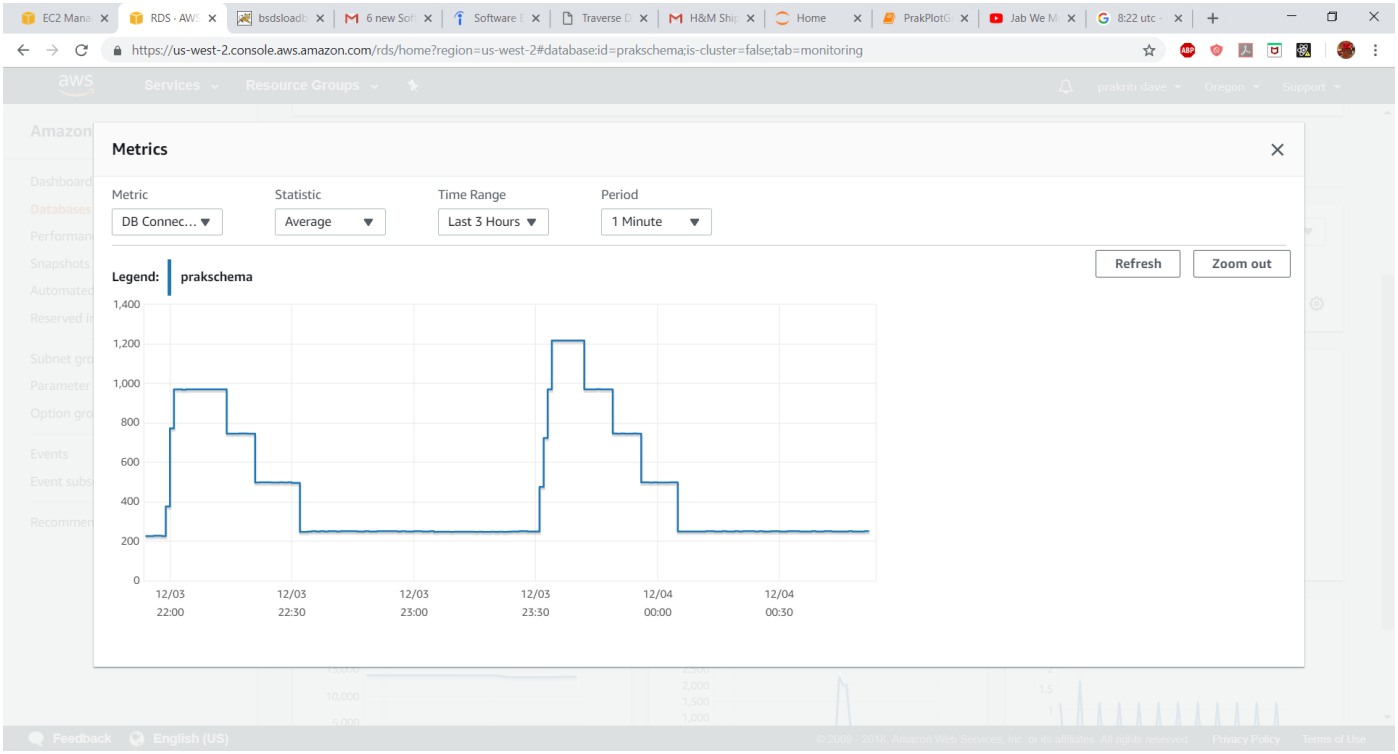
450 Threads, 100 iterations:

For 450 threads, all my 5 instances are spawned during peak phase. However my peak phase is still running and 4 of my instances have terminated so there seems to be a bottleneck from client to server, since the RDS size is t2.xlarge and it can handle 1400 connections. So after a while the throughput is pretty less even during the peak phase.





DB connections:



Points of observation/Lessons learned the hard way:

1. For almost a week I was stuck with 256 threads, I kept getting the error either **Connection Time out or Bind error exception, Address not available**. This happenned with the Load Balancer and I was not able to figure out why. This was the reason I moved my client to ec2 as well. I later realized I could get away with just one webTarget object but I was creating a new one for each request, which was one of the main reasons why my resources were getting exhausted at the network side. I also modifed a couple of my settings on the DBCP pool configuration and Tomcat server. I changed my RDS instance size to t2.medium and then later to t2.xlarge for better performance with load balancer.

2. Before I moved my client to ec2 for 256 threads , I had also executed my client on local, both with and without load balancer upto 128 threads, and surprisingly the throughput graph looks better for them. 😊

   I have uploaded my results for client run on local as well, in my github repo.