Cloud Assignment 3

The Objective of the assignment was to implement in memory and sqs queue locally and in aws to be processed by local and remote workers

Design Trade off and consideration

Client

1. The client takes input as input file , type of worker(local remote or animoto), total no of workers(for remote execution) and total no of threads (for local worker)
2. adds the data in local queue request depending on the type of sleep specified in file eg

For sleep 0 tasks will be added 10000 times for sleep 10 1000 times

1. In local queue hash map is used to store job id and sleep jobs

Job Id will increment when new job is added

1. Client then calls the local worker and process the queue
2. It checks if the response queue is processed with all data. if size of response queue is equal to no of tasks it will print data in response queue

Local Worker

1. Local worker is invoked from client
2. In memory queue is passed when local worker starts
3. Local worker will take each task from queue and process it
4. Executor service is used to execute the threads and process sleep jobs in local worker
5. process the sleep job and adds the end result to response queue
6. hash map is used to store the client task id and job in LocalWorkerQueue class which is class for in memory queue
7. client stores the task one by one in hash map and local worker take task from hashmap process it and removes the task from hashmap
8. executor.execute service and runnable threads Is used for each performing each task in threads

Remote Worker

1. Remote worker is started manually
2. When it starts it checks the sqs request queue to see if there are any jobs there to process
3. Remote worker will take each task from queue and process it
4. Executor service is used to execute the threads and process sleep jobs in remote worker
5. In remote worker thread each task is passed to dynamo db to check if there are any duplicates
6. Table is created in dynamo db with primary key as task
7. Dynamo db will insert each task id in the table if there are any primary key constraint violation it will throw an error if duplicate task id are added
8. Messages in message queue will be processes one by one and we will start task id and job info in class Task Info which has jobid and job
9. In remote worker thread the job id is split to get the sleep time and perform sleep operation
10. If dynamo db insert returns 1 there is no violation else it returns 0
11. If 1 is returned and task is unique it process the sleep job and adds the end output to response queue

Animoto

1. Remote worker is started manually
2. When it starts it checks the sqs request queue to see if there are any jobs there to process
3. Remote worker will take each task from queue and process it
4. Executor service is used to execute the threads and process each job of merging 60 pictures into video in local worker
5. In remote worker thread each task is passed to dynamo db to check if there are any duplicates
6. Table is created in dynamo db with primary key as task
7. Dynamo db will insert each task id in the table if there are any primary key constraint violation it will throw an error if duplicate task id are added
8. If dynamo db insert returns 1 there is no violation else it returns 0 if duplicate is found
9. Each worker thread will download the 60 images and process them into single video
10. Worker thread will then upload the video into s3 bucket
11. It will pass the url of each video uploaded in s3 to response queue to client
12. Used ffmpeg to convert and merge image to video

Design TradeOff

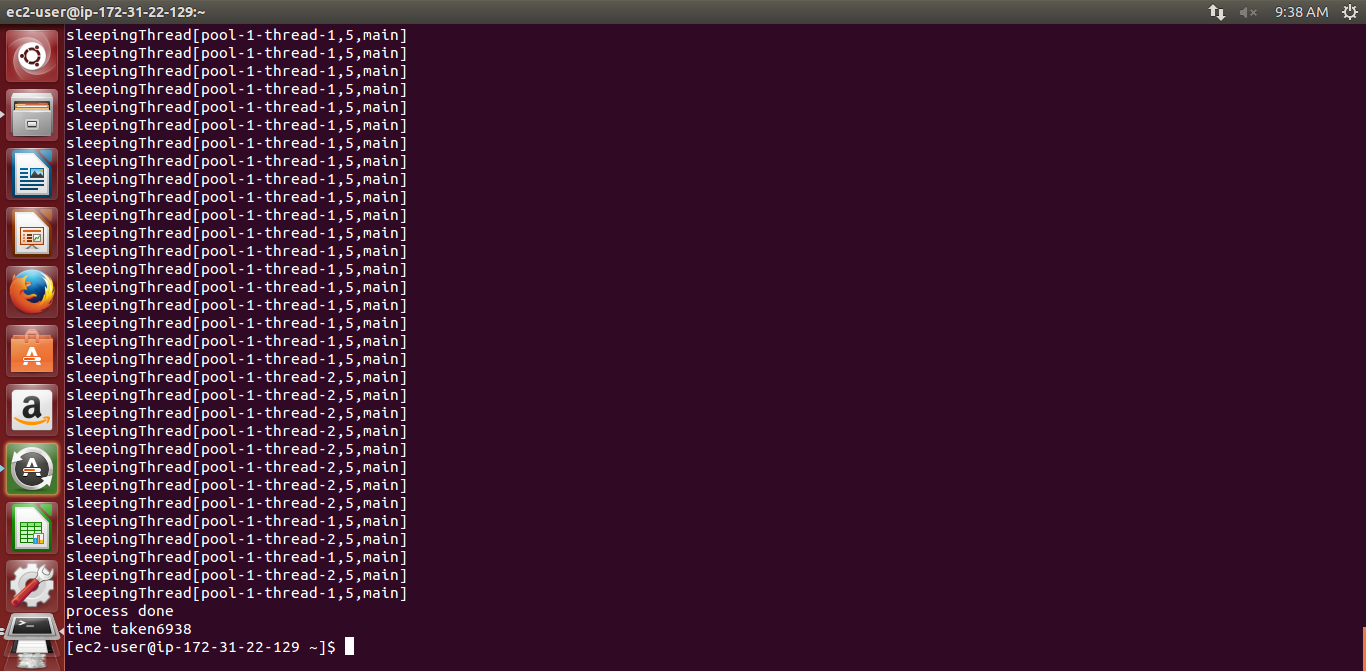
As Sqs insert was slow increased the read write capacity for dynamo db

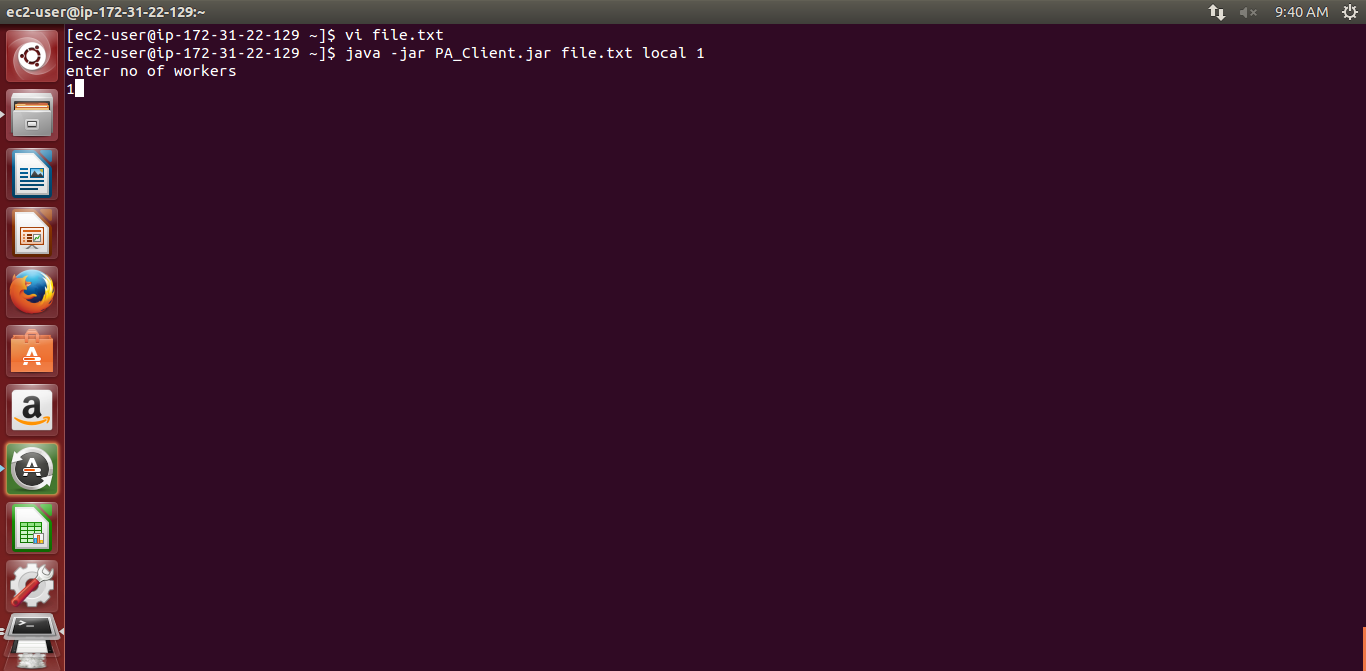
LocalQueue Implementation pass output response queue for each sleep

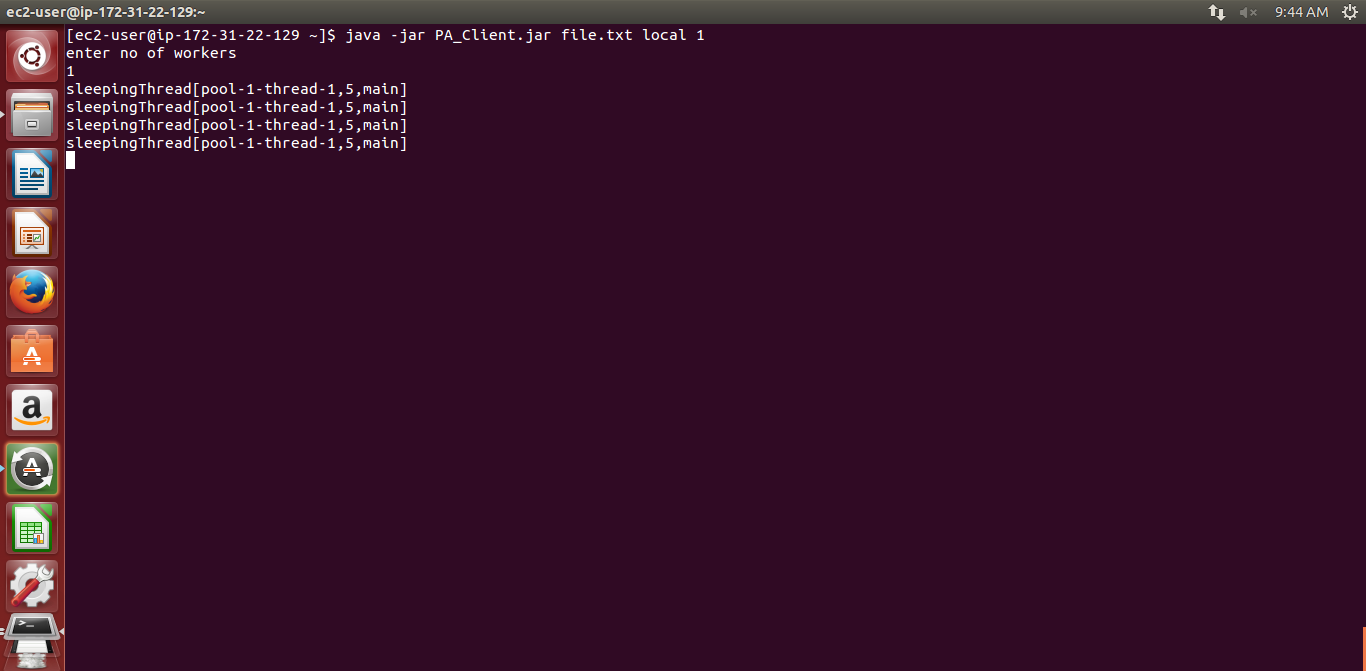
Implemented dynamo db to check duplicate key constrains by creating a table in aws

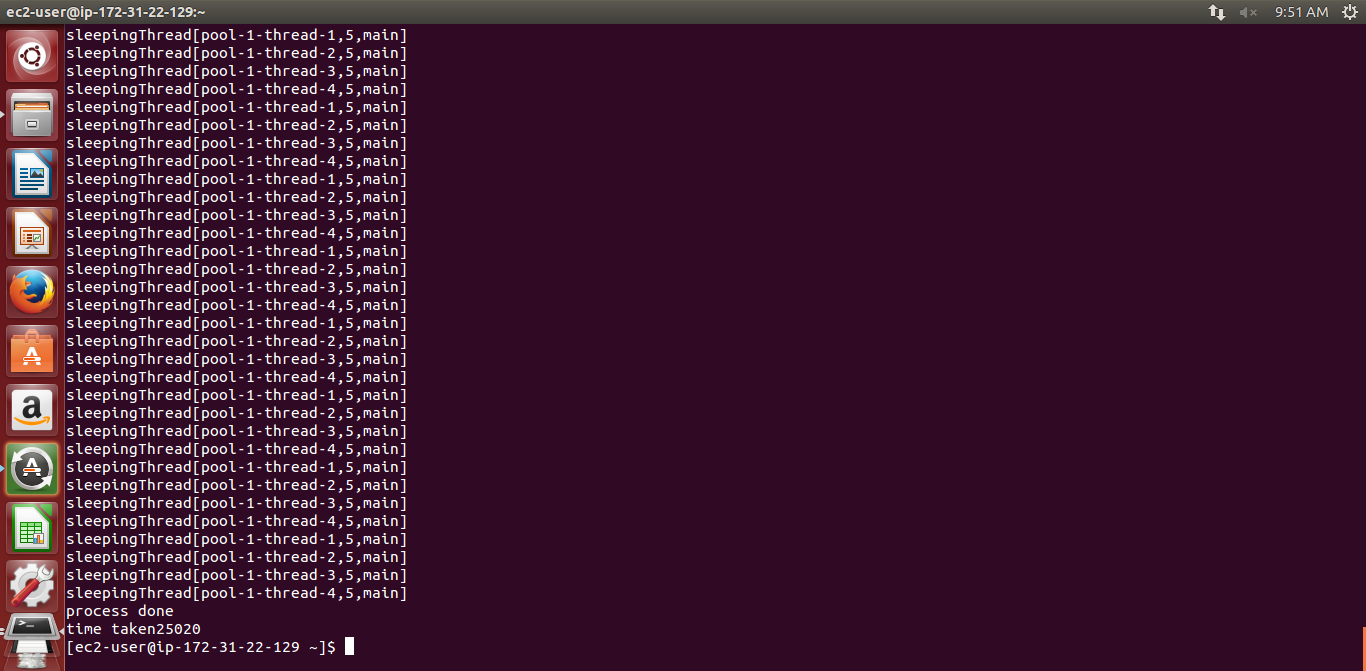
Screen shot

Thread sleep 0

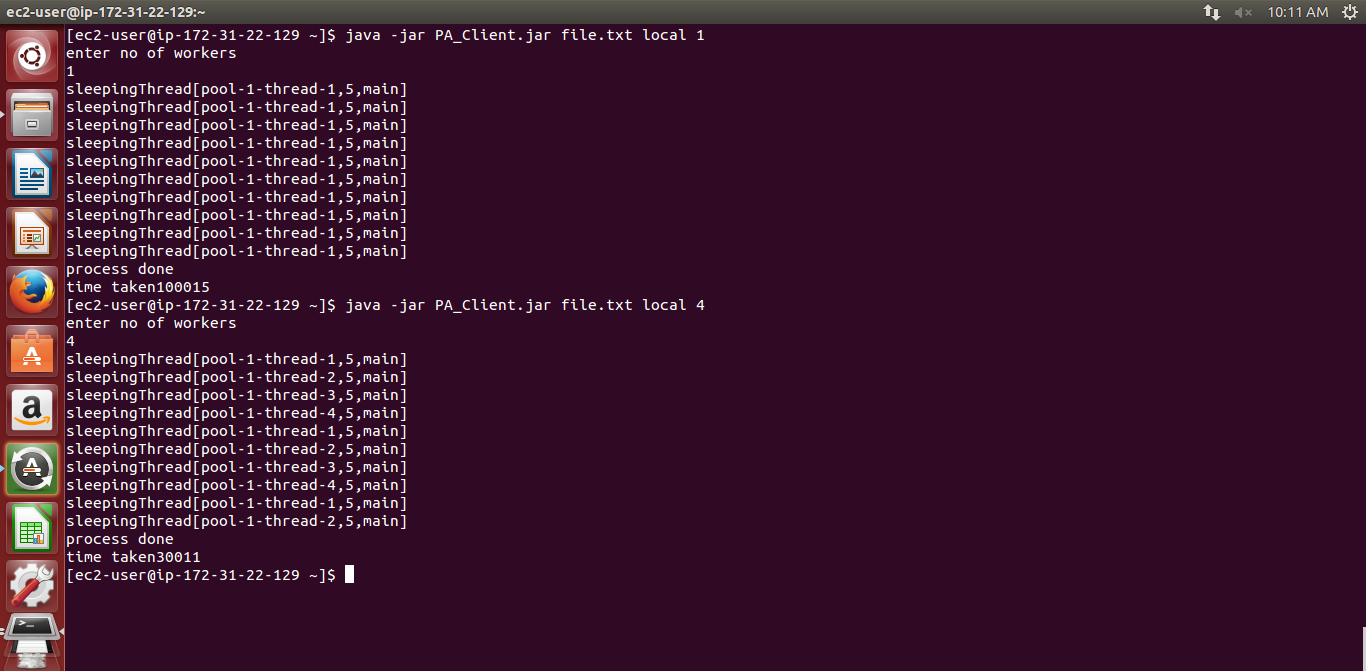




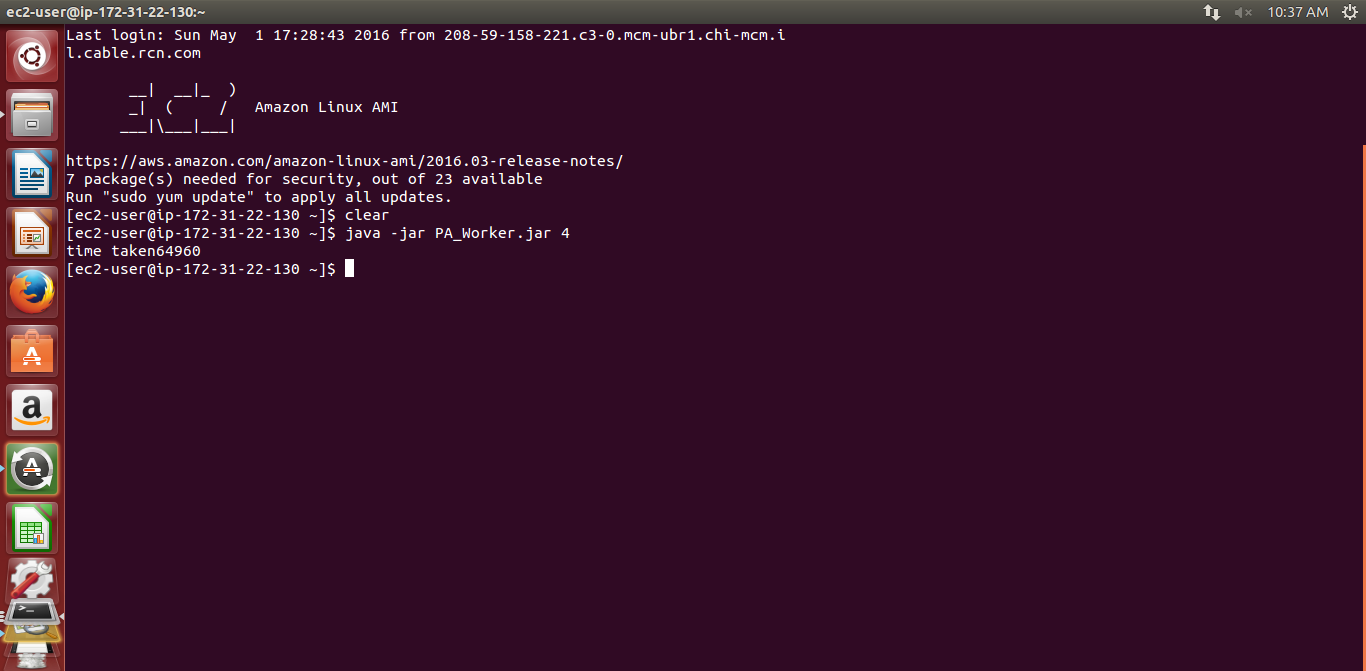


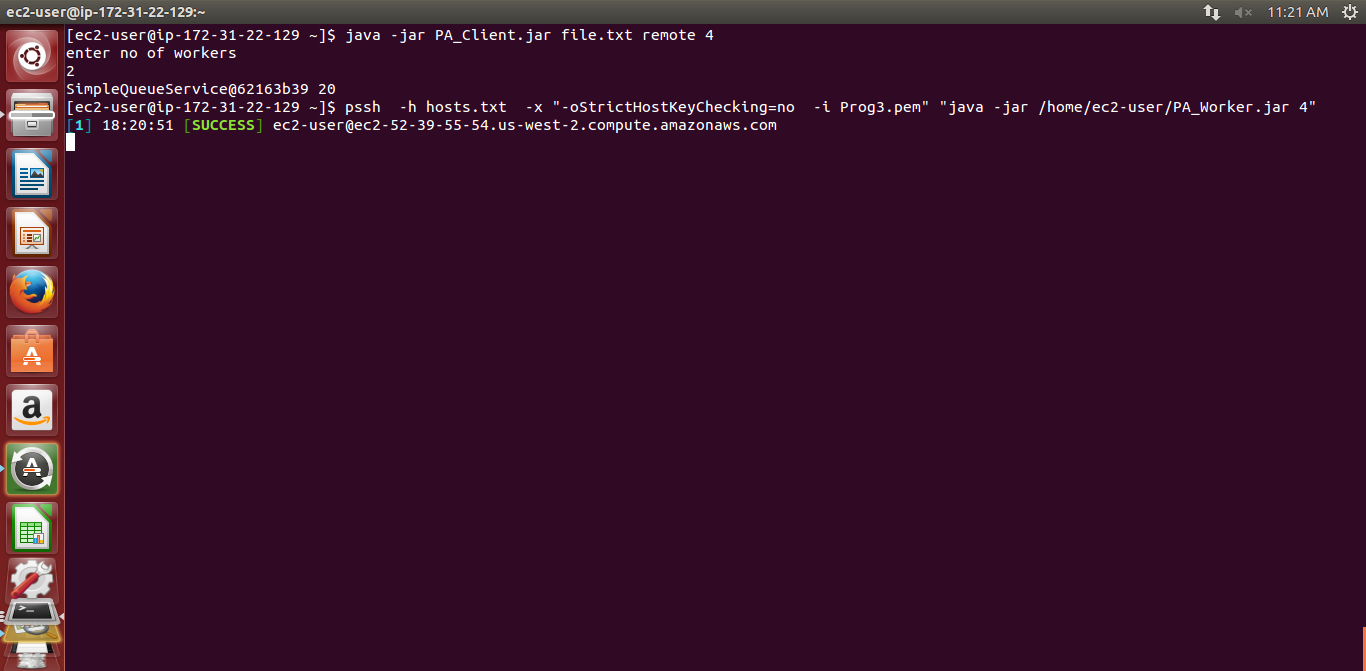


Local worker 10000 nodes

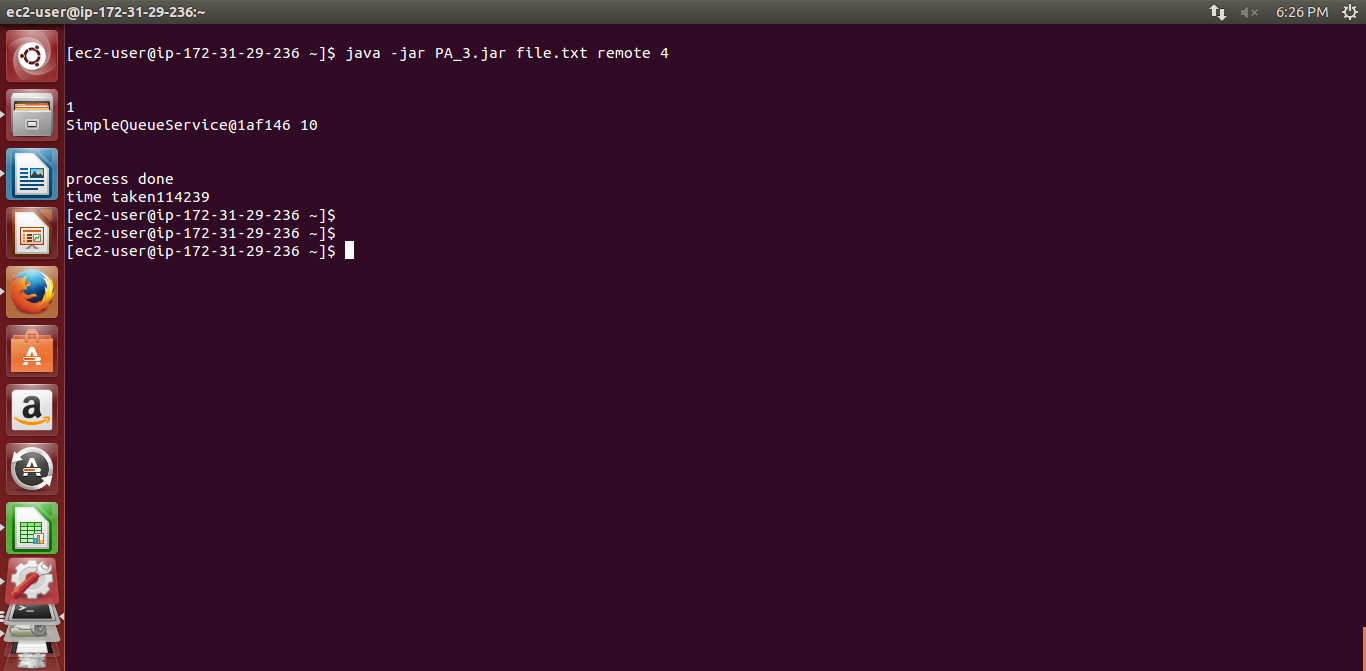


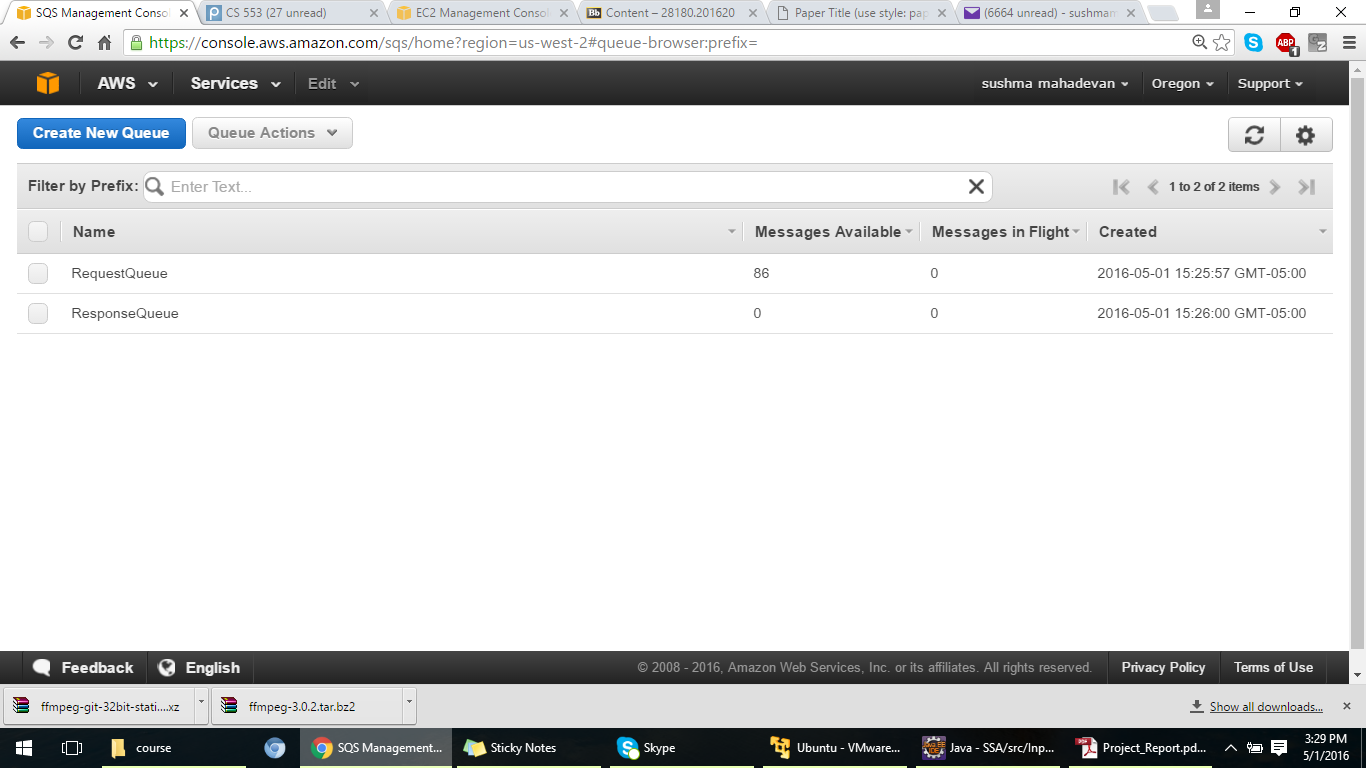
SQS Remote Worker





Sleep 10000 in sqs





Local

Throughput Sleep 0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Threads | Time Taken in millisec | Throughput/millisec | Throughput/sec |  |
| 1 | 7268 | =10000/7268=1.37 | 1370 |  |
| 2 | 6938 | =10000/6938=1.44 | 1440 |  |
| 4 | 6755 | =10000/6755=1.48 | 1480 |  |
| 8 | 7219 | =10000/7219=1.38 | 1380 |  |

Throughput Graph for local

Local Efficiency

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Sleep Time | Threads | TimeTaken | Ideal Time | Efficiency  =(Ideal/time)\*100 |
| 10 | 1 | 10158 | 10 | 98.4 |
| 10 | 2 | 5090 | 10 | 98.23 |
| 10 | 4 | 2557 | 10 | 97 |
| 10 | 8 | 1290 | 10 | 96.89 |
| 1000 | 1 | 100037 | 1000 | 99 |
| 1000 | 2 | 50022 | 1000 | 99 |
| 1000 | 4 | 25020 | 1000 | 99 |
| 1000 | 8 | 13014 | 1000 | 99.8 |
| 10000 | 1 | 100015 | 10000 | 99 |
| 10000 | 2 | 50013 | 10000 | 99s |
| 10000 | 4 | 30011 | 10000 | 83 |
| 10000 | 8 | 20009 | 10000 | 62 |

Remote Efficiency

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Workers | Time Taken in sec | Sleep Task | Ideal Time in sec | Efficiency Calculation |
| 1 | 132 sec | 10000 | 10000\*10=100000=100 sec | 75 |
| 2 | 234 sec | 10000 | 2\*10000\*10=100000=200 sec | 85.4 |
| 4 | 423 sec | 10000 | 4\*10000\*10=400 sec | 94 |
| 8 | 856 sec | 10000 | 8\*10000\*10=800 sec | 93.48 |
| 16 | 1690 sec | 10000 | 16\*10000\*10=1600 sec | 94.67 |
| 1 | 110 sec | 1000 | 1000\*100=100000=100 sec | 90.9 |
| 2 | 212 sec | 1000 | 200 sec | 94.33 |
| 4 | 402 sec | 1000 | 400 sec | 99.5 |
| 8 | 803.5 sec | 1000 | 800 sec | 99.56 |
| 16 | 1670 sec | 1000 | 1600 sec | 95.8 |
| 1 | 12.4 sec | 10 | 10\*1000=10000=10 sec | 80.64 |
| 2 | 21 sec | 10 | 2\*10000\*1=10000=20 sec | 95.23 |
| 4 | 43 sec | 10 | 4\*10\*1000=10000=40 sec | 93 |
| 8 | 85 sec | 10 | 8\*10\*1000=80 sec | 94.11 |
| 16 | 167 sec | 10 | 16\*10\*1000=160 sec | 95.8 |

Remote Throughput

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sleep Time | Workers | TimeTaken | Throughput/Millisec | Throughput/Sec |  |
| 0 | 1 | 64 sec | 10000/64=  156.25 | 15625 |  |
| 0 | 2 | 74 sec | 20000/74=  270.27 | 27027 |  |
| 0 | 4 | 77.4 sec | 40000/77.4=516.79 | 51679 |  |
| 0 | 8 | 81..2 sec | =80000/81.2=985.22 | 98522 |  |
| 0 | 16 | 84.3 sec | =160000/84.3=1897.9 | 189798 |  |

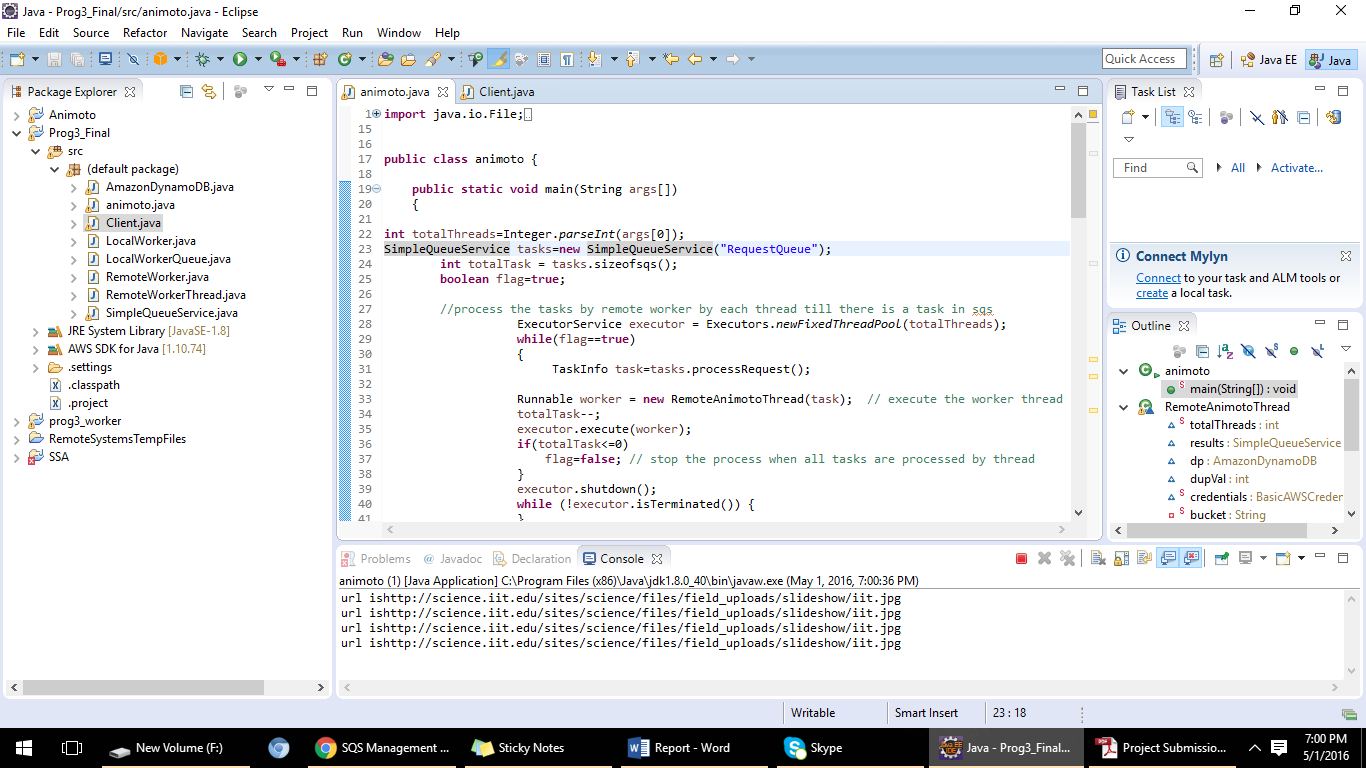
Local

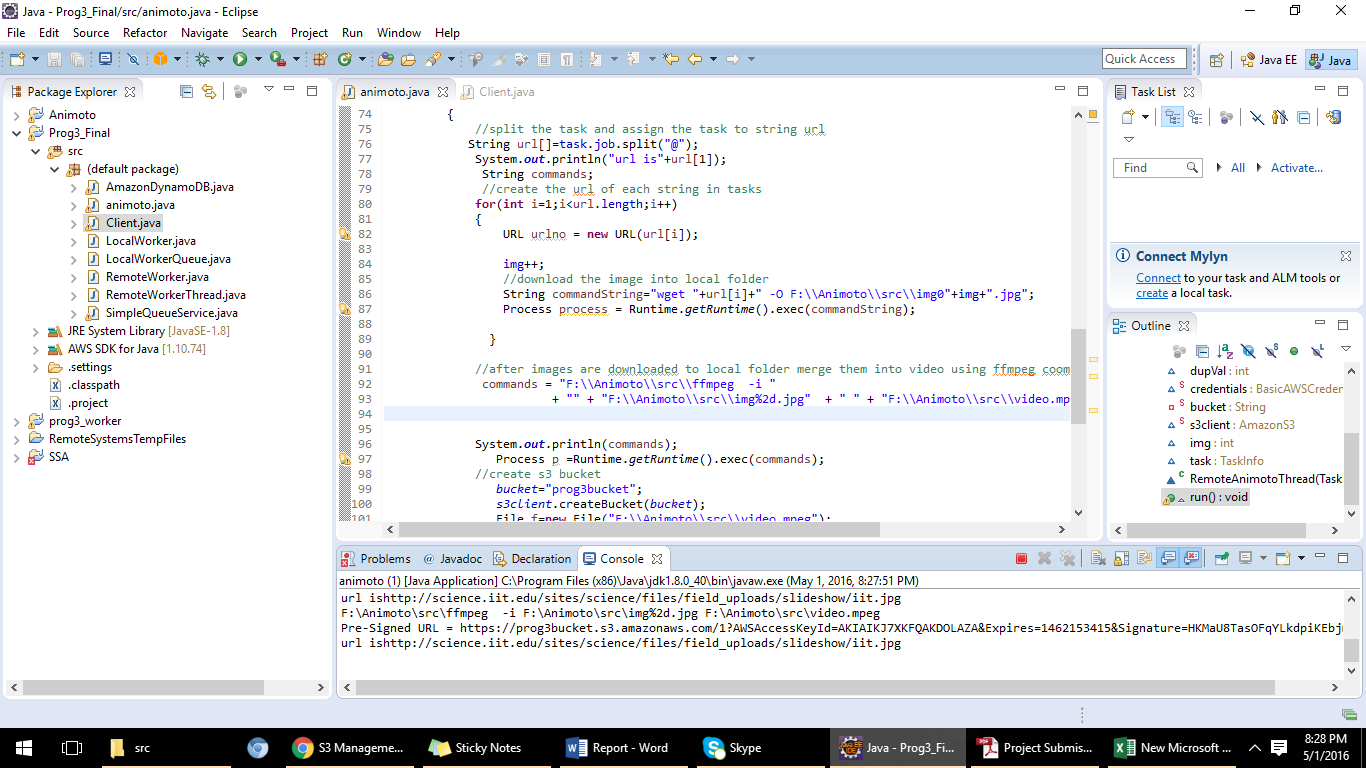
For local worker as we increase the threads the efficiency and throughput increases as more threads perform the task in local queue

For remote worker as we increase no of nodes and the tasks based on nodes time increases for performing the task but since there are many worker nodes parallel task are executed more faster compare to single node hence efficiency and throughput increases.

As we compare sqs is better to execute when there are lots of data and in remote when more workers are there to execute it scales well

Animoto Execution in eclipse





Output displays url for s3 bucket uploaded videos