Priyank Shah A20344797

Programming Assignment 3
CloudKon clone with Amazon EC2, S3, SQS, and DynamoDB

<u>1. Design Document</u>

Introduction:

Goal of this programming assignment is to working with Amazon SQS and Amazon DynamoDB, and being aware of its various functions and properties. Primary Objective of this assignment is divided into 3 major tasks:

1. Reading Data from Workload File
2. Uploading it on Amazon SQS
3. Fetching Data from Amazon SQS and Execute task.
4. Additionally, Use of DynamoDB to eliminate the duplicate task from Amazon SQS.

Amazon SQS:
Amazon SQS [Simple Queue Service] is a fast, reliable, scalable and fully managed service which guaranties that at least one message will be delivered to client, but SQS does not have provision for eliminating duplicate Task. SQS can transmit any volume of data at any level of throughput with the guaranty of delivering at least one message.

Amazon DynamoDB:
Amazon DynamoDB is fast and flexible NOSQL cloud Database, that manages both document and key-value store with single-digit millisecond latency.

Design:

The program is implemented in Python language with the help of boto package. Boto is the package used to commmunicate with amazon SQS and Amazon DyanamoDB. While checking the throughput of the Local and Remote Worker, I have used System Calls for better results and for efficiency I have used Thread with time() package's sleep function.

Local Client:

- It handles both local client and local workers with the help of threads, initially program will take arguments as number of threads and workload file name input, and execute the t ask from user directed file and number of threads. At the end of execution it will display the execution time taken.

Remote Client:

- Remote Client will receive the number of argument as queue name and number of spot instances to be created. Once user input is given, It will create the queue on amazon SQS, and create the Table on DynamoDB.
- It will copy the content of workload file on the Amazon SQS and wait for the workers to execute the task and send back into return queue. Once data is start flowing into return queue it will check with the dynamoDB id and verify it.

Remote Worker:

- Each Remote worker works independently from the client, and fetch data from the SQS queue and send it into DynamoDB table created earlier.
- DynamoDB check the data if duplicate found then discard it, other wise Remote worked will execute it.
- Once the Task has been executed, it will be sent back to the Return Queue located on Amazon SQS from where Remote Client will poll and verify that all the task has been executed.

Performance Measurement Formulas:

- Throughput is measured with the following formula:

  Throughput = Number of Task / Execution Time.

- Efficiency is measured with the following formula:

  Efficiency = (Ideal Time to Execute Task / Actual Execution Time) * 100

2. Manual

- **LIST-OF-FILES:**

1. client.py
2. credential.py [contains AWS KEY AND ACCESS ID]
3. worker.py
4. writer.py
5. data [resides in Worker folder]
6. install.sh
7. workerCopy.sh

- **PREREQUISITE-STEPS:**

1. Before Running program run install.sh to ensure boto installed on machine, which is essential for SQS/DynamoDB in Python.
2. Generate dataset as per given argumanet instruction to generate file with desired sleep time and tasks.
3. Copy Hostname of Clients into one file on server in filename hostname_client.
4. For Remote Experiment execute workercopy.sh on client to copy required file to the client.
5. Copy the Amazon Secret Access Key and ID into credential.py and change .pem key file name in script that is workercopy.sh
6. By Defalut File generated from writer.py will be Store in Worker Folder named as data, so If you are using that file than give argument to worklaod file as "/Workload/data"

- **File Arguments:**

Data_Generation:
python writer.py -t [number of Task] -v [Sleep Time]
i.e. python write.py -t 1000 -v 100

For local worker:
python client.py -s LOCAL -t [number of threads] -w <WORKLOAD_FILENAME>
i.e. python client.py -s LOCAL -t 4 -w Workload/data

For Remote Client:
python client.py -s [Queue_NAME] -w <WORKLOAD_FILENAME>
i.e. python client.py -s QUEUE -w Workload/data

For Remote Worker:
python worker.py -s [Queue_NAME] -t [Number of Workers]
i.e. python worker.py -s QUEUE -t 16

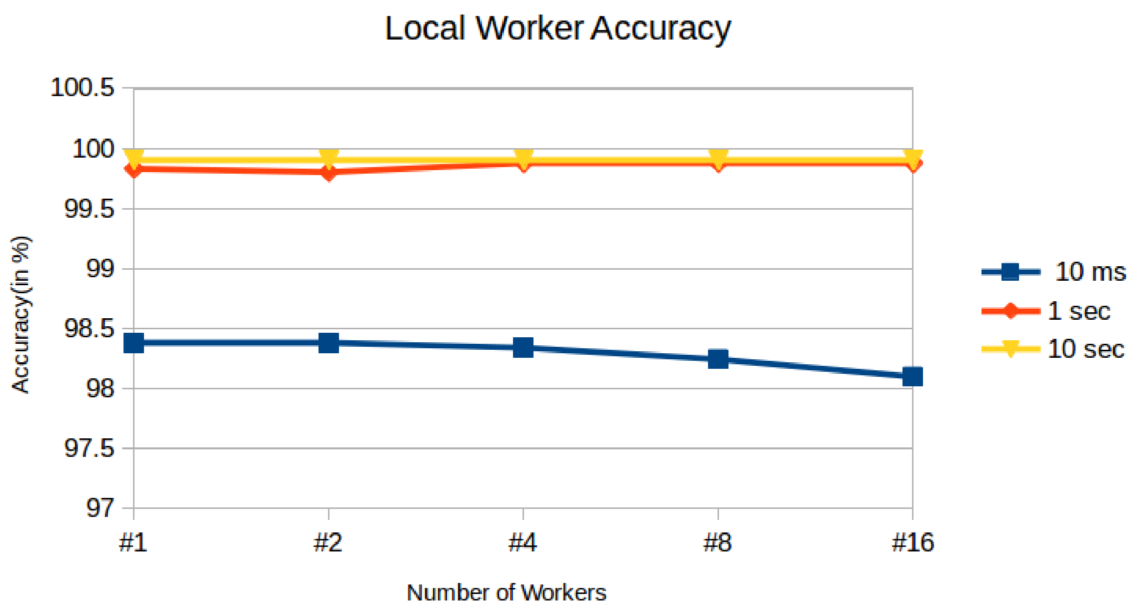
- **How--to--Run-?**

1. Fulfill the PreRequisite steps
2. For local Experiment, Just run file client.py with the above mentioned arguments.
3. For Remote Client, Run client.py with required argument
4. Run worker.py for remote worker with required argument.

3. Performance Evalution

1. Efficiency:

- **Local:**



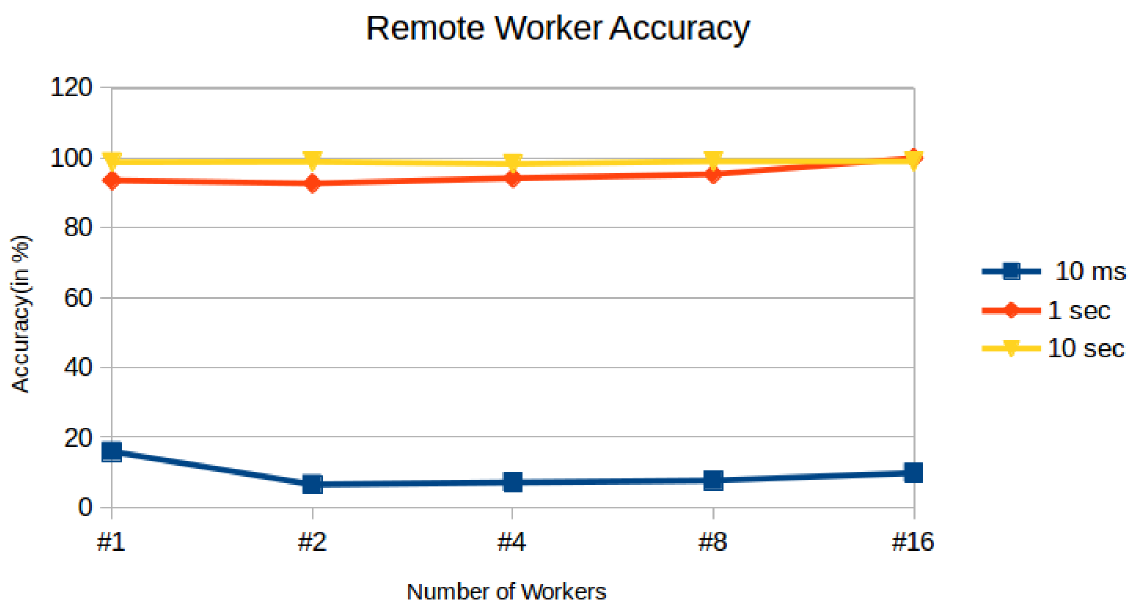
- **Local Execution Time Table:**

All the times are in Seconds, Column Denotes number of Workers

Time	1	2	4	8	16
10 ms	10.165	10.165	10.169	10.179	10.194
1 sec	100.17	100.199	100.122	100.121	100.123
10 sec	100.102	100.102	100.101	100.103	100.103

From the above graph, we can observe that accuracy for 10 ms experiment took so much time due to overhead. 1 Second and 10 Second experiment achieved very good accuracy.

- **Remote:**



- **Remote Execution Time Table:**

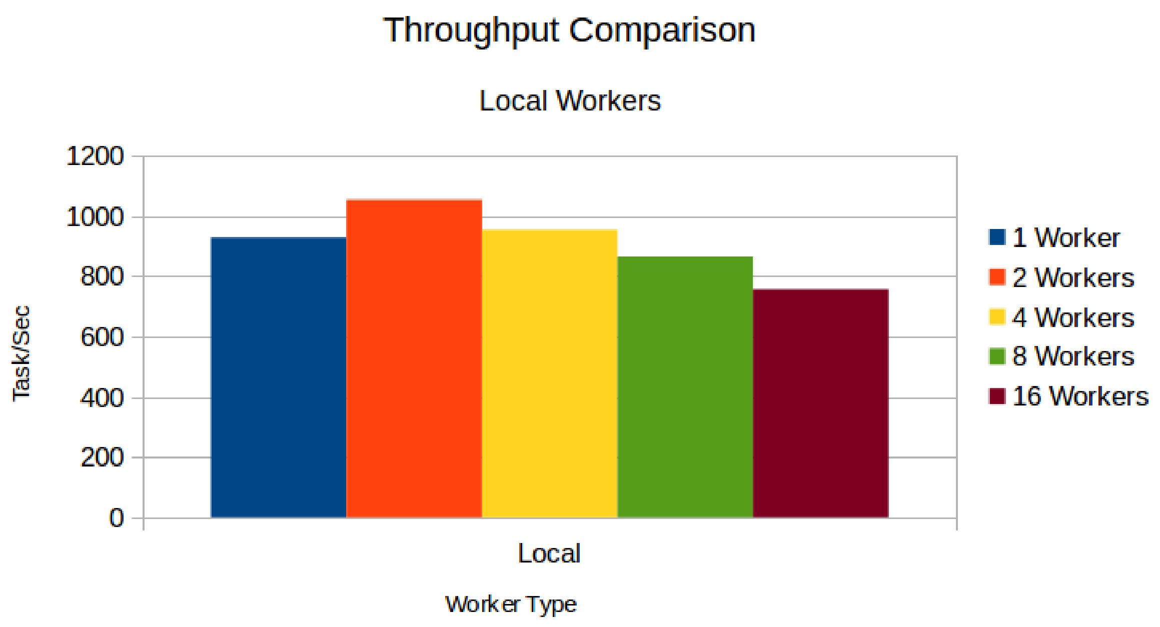
All the times are in Seconds, Column Denotes number of Workers

Time	1	2	4	8	16
10 ms	62.974	153.932	140.511	130.702	102.183
1 sec	106.86	107.854	106.124	104.872	99.988
10 sec	101.249	101.11	101.71	100.954	101.037

from the above graph, we can observe that accuracy for 10 ms experiment took so much time due to data retrieval from the Dynamodb. 1 Second and 10 Second experiment achieved very good accuracy.

2. Throughput:

- **Local:**



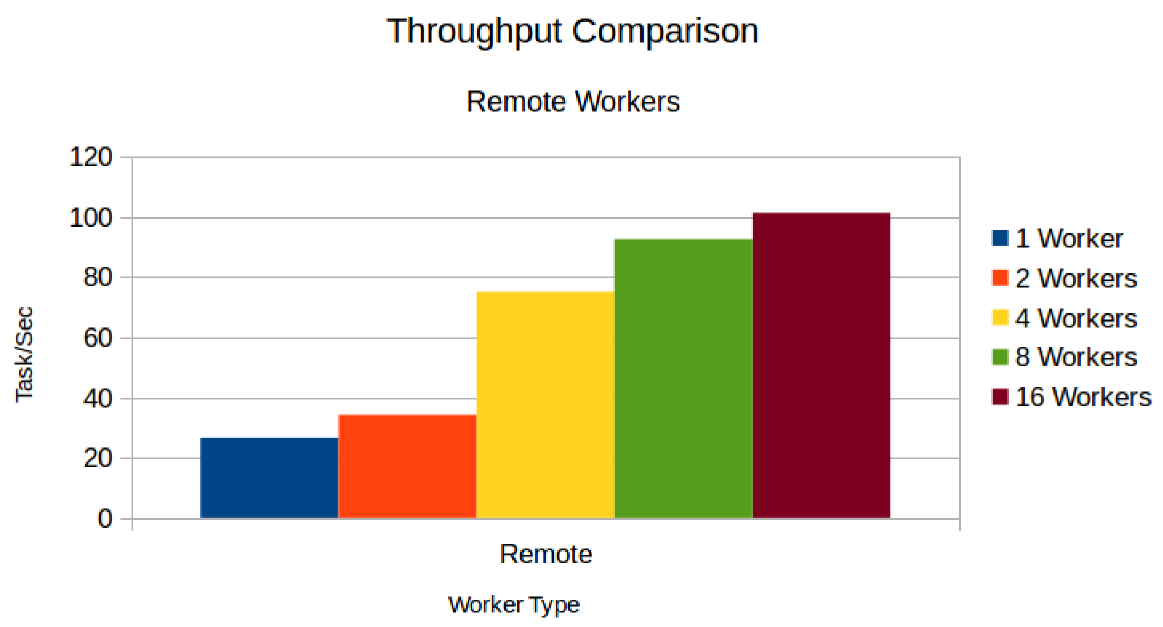
- **Local Worker Execution Time Table:**

All the times are in Seconds, Column Denotes number of Workers.
Task Executed was "Sleep 0"

Worker Type/Numbres	1	2	4	8	16
Local	10.774	9.491	10.48	11.553	13.213

from the above graph, we can observe that 2 workers were able to accomplish task quicker than other. 16 workers achieved less throughput due to single core system.

- **Remote:**



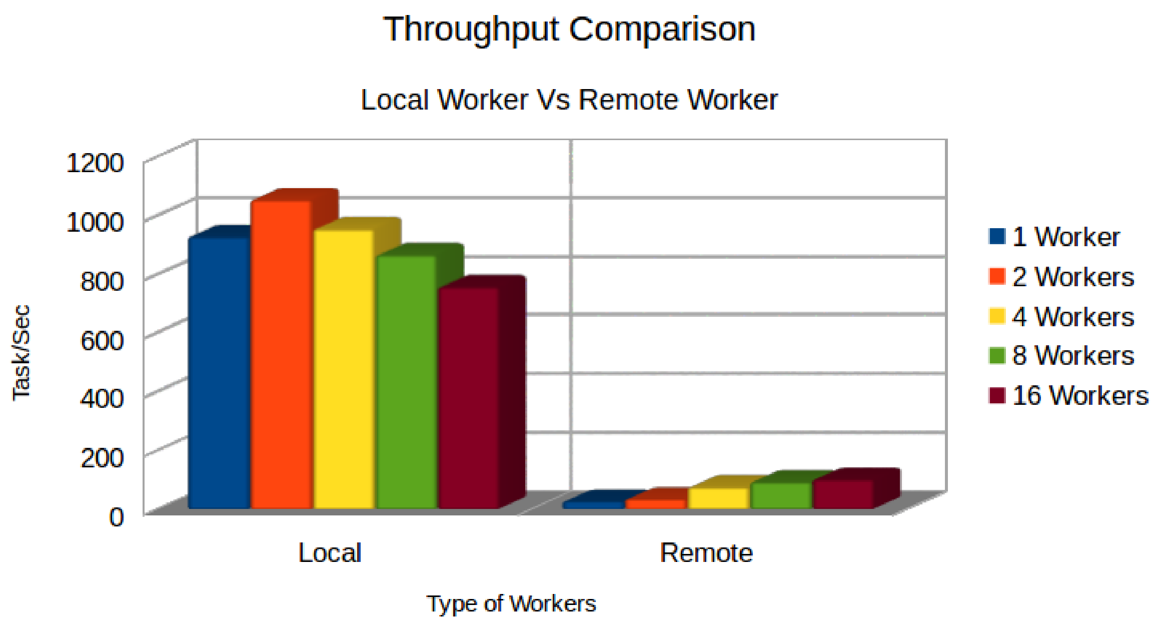
- **Remote Worker Execution Time Table:**

  All the times are in Seconds, Column Denotes number of Workers.
  Task Executed was "Sleep 0"

Worker Type/Numbres	**1**	**2**	**4**	**8**	**16**
Remote	376.59	292.616	133.088	108.07	98.729

from the above graph, we can observe that 16 workers were able to accomplish task quicker than other. 16 workers achieved highest throughput due to concurrently working with dynamoDB and SQS.

- **Local Vs Remote Worker Throughput Comparison:**



Worker	Remote	Local
1 Worker	26.55408	928.1604
2 Workers	34.17448	1053.63
4 Workers	75.13825	954.1985
8 Workers	92.53262	865.576
16 Workers	101.2874	756.8304

Local Worker Achieved higher throughput as they didn't had to deal with the SQS and DynamoDB, as it saves a lot of time.

4.References:

1. http://boto.cloudhackers.com/en/latest/sqs_tut.html
2. http://boto.cloudhackers.com/en/latest/dynamodb2_tut.html
3. http://programminggenin.blogspot.com/2012/10/getting-started-with-amazons-dynamodb.html
4. http://www.tutorialspoint.com/python/python_multithreading.htm