CS 553 PROGRAMMING ASSIGNMENT-1

EVALUATION DOCUMENT

ROHIT SINGH

A20361198

In this assignment I want to assign a set of jobs (sleep tasks) into a queue which will be done by the workers. The jobs will be varied according to the number of workers and the time for the sleep job. It has to be made sure that the workers do not get duplicate tasks. To do so I do two implementations local and remote. Then I find out the throughput and the efficiency of the implementation.

I have also done the Animoto Clone, which is the remote implementation, with the jobs changes. Instead of sleep jobs the workers need to download HD images and combine them to make a video.

In this assignment I learnt the uses of multithreading through the local implantation and about SQS, DynamoDB and S3 through the remote implementations.

1. AWS Specifications

The specifications used can be summarized as follows:

- Instance : t2.micro
- Region : Oregon(West)
- vCPU : 1
- RAM : 1GB
- OS : Ubuntu Server
- Java : Version 1.7
- SQS (Simple Queuing Service)
- DynamoDB: Read & Write Throughput 160L.
- S3

2. Local

In the local implementation I have implemented multithreading using the **Executor Service** and Queues using the collection class. The experiment reads jobs from the file one by one and pushes it into the queue. Before the tasks can be pushed into the queue, there needs to be some amount of string handling to be done, since, the jobs written in the file are in the following format

sleep 0
sleep 0
…..
sleep 0

The time to sleep (in the above example is 0ms) is pushed into the queue. A pool of threads are created, and one by one the tasks are popped from the queue and assigned to the threads. The threads keep on sensing the queue until the queue is empty, as shown below:

```
while ( queue.isEmpty() == false)
{
        //pop tasks one by one and feed it into the executer service
        Runnable worker = new WorkerThread(popTask(queue));
        executor.execute(worker);
}
```

To execute this experiment the following command can be used:

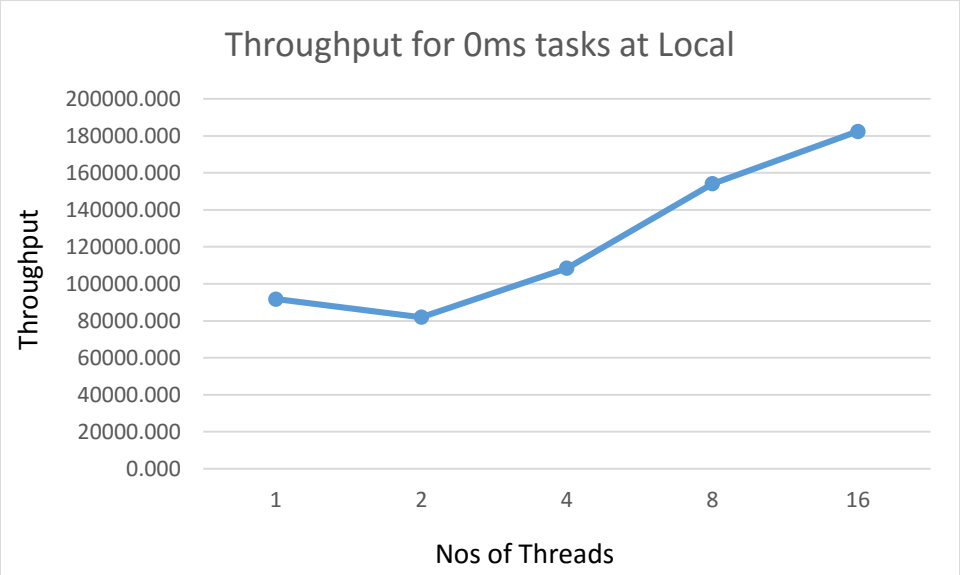
java -jar LOCAL.jar <#Threads> <FileName>

The local queue guarantees that once an item is popped the item will not get repeated. Hence, the threads will always have unique jobs to perform. Here, the local code emulates the client and the threads as the workers when compared to the remote experiment.

For evaluation we take experiments with 10,000 tasks of 0ms so as to find the throughput of the system.

Table 1 Local experiment of 0ms tasks with 10000 tasks per thread

Nos of Threads	Time taken (in sec)	Throughput (tasks/sec)
1	0.109	91743.119
2	0.122	81967.213
4	0.09225	108401.084
8	0.064875	154142.582
16	0.0548125	182440.137



From the graph above we can observe that the throughput increases with the increase in threads. This is obvious since the number of jobs done parallel will be the proportional to the number of workers. However, it is not always true, as the threads will itself take time to co-ordinat. This becomes an overhead.

To get the efficiency Tasks per worker per instruction we take 3 cases:

- 10ms tasks with 1000 tasks per thread
- 1s tasks with 100 tasks per thread
- 10s tasks with 10 tasks per thread

To calculate the efficiency, I used this formula:

$$Efficiency = \frac{Time\ per\ task * Tasks\ per\ workers}{Time\ taken} * 100 = \frac{Idle\ Time}{Time\ Taken} * 100$$

Table 2 Local experiment of 10ms tasks with 1000 tasks per thread

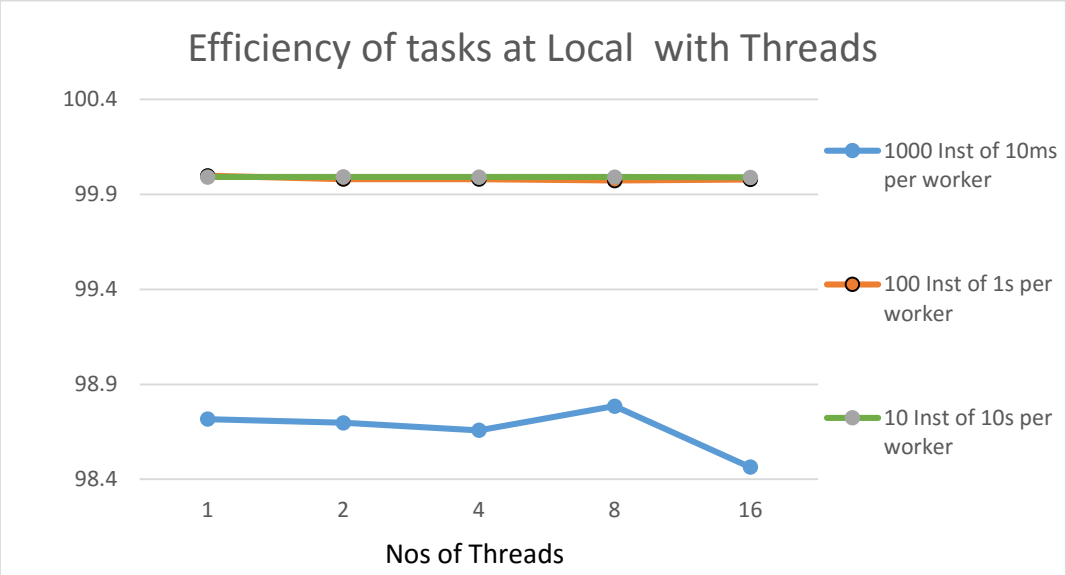
Nos of Threads	Time taken (in sec)	Efficiency(%)
1	10.13	98.7167
2	10.132	98.6972
4	10.136	98.6582
8	10.123	98.7849
16	10.156	98.464

Table 3 Local experiment of 1s tasks with 100 tasks per thread

Nos of Threads	Time taken(in sec)	Efficiency(%)
1	100.002	99.998
2	100.02	99.98
4	100.02	99.98
8	100.027	99.973
16	100.021	99.979

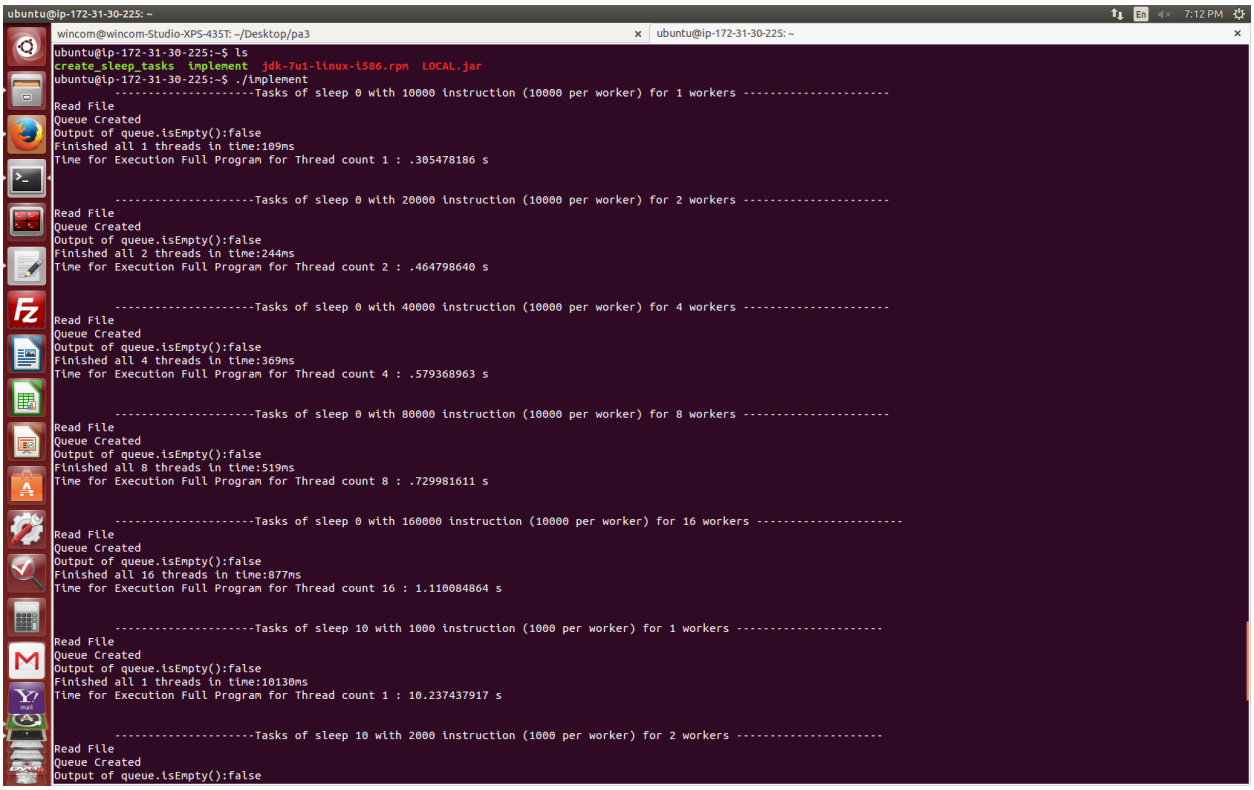
Table 4 Local experiment of 1s tasks with 10 tasks per thread

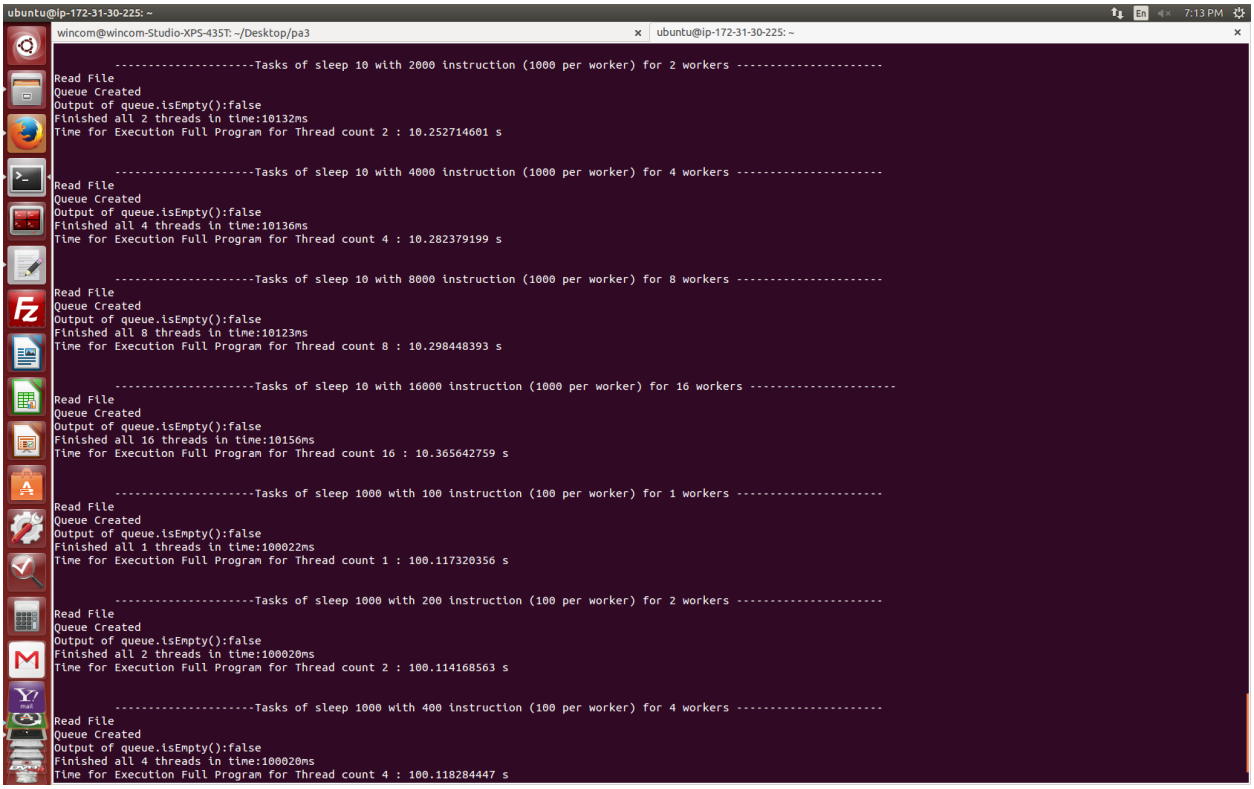
Nos of Threads	Time taken(in sec)	Efficiency(%)
1	100.009	99.991
2	100.008	99.992
4	100.009	99.991
8	100.009	99.991
16	100.01	99.99

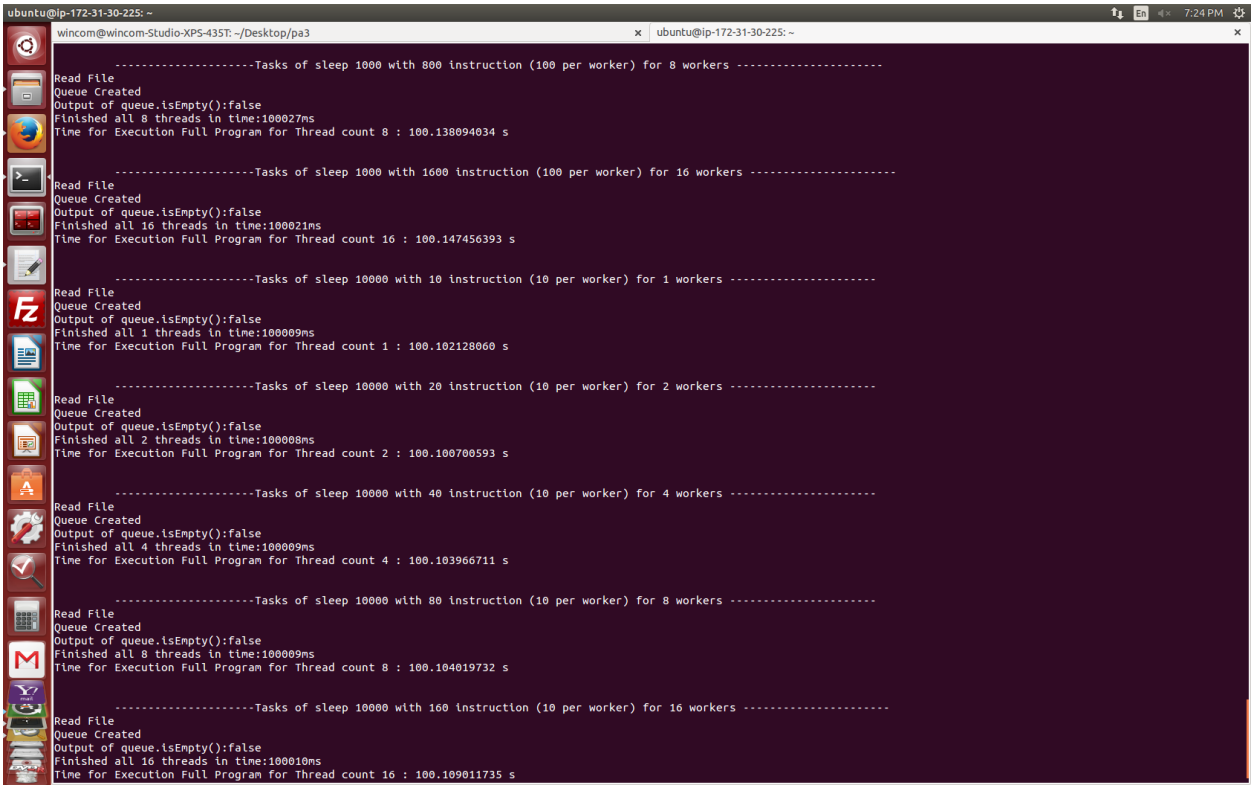


The efficiency for the local is high when the number of instructions are low and high when the number of instructions are low. Since, there is no excess overhead occurring the efficiency is neatly 100%. When the number of instructions are low the probability of the threads not working properly increases.

Screenshots





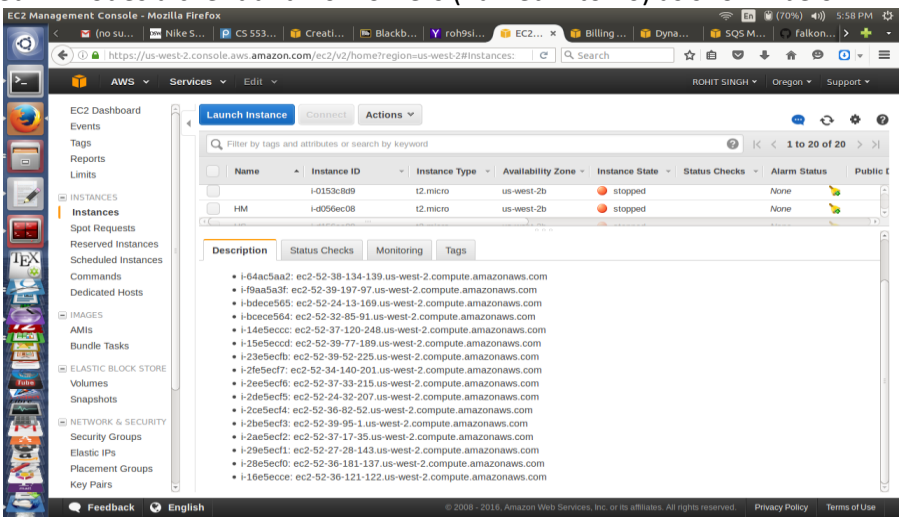


3. Remote

In the Remote implementation I have implemented a client and a set of workers which vary from 1, 2, 4, 8 and 16 in number.

The experiment uses SQS and DynamoDB to make a queue and check for duplications respectively.

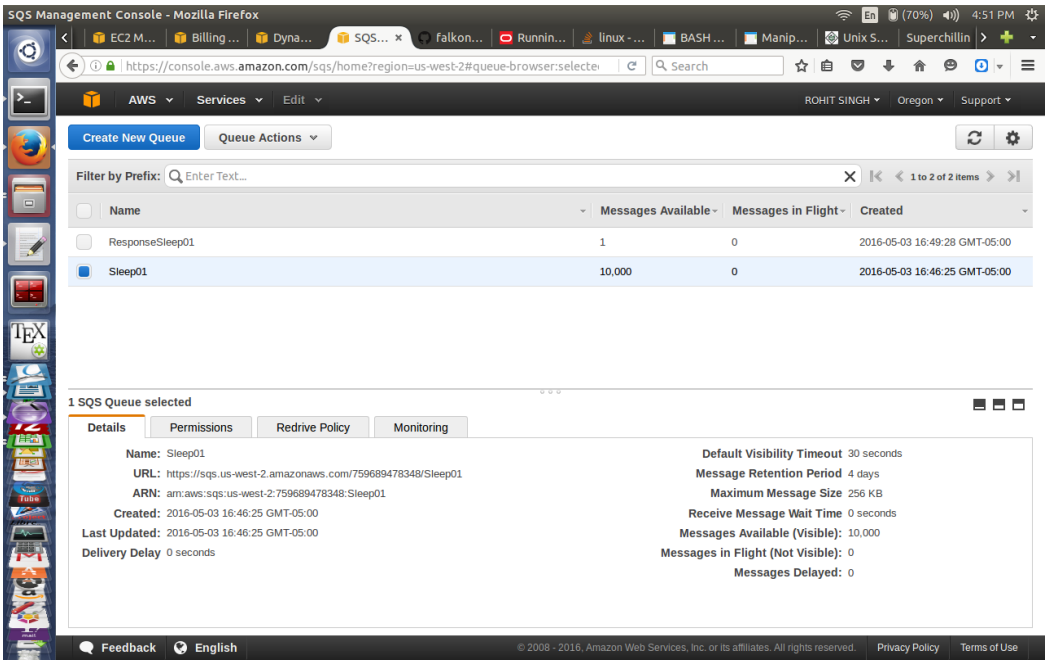
I have created 17 nodes a client and 16 workers (named 1 to 16) as shown below:



The **client** submits the tasks into a queue here is SQS using a command line interface. After the client submits all the tasks into the queue, it then submits the URL to the queue into DynamoDB into a Table named QueueURL, so that the workers have access to the SQS. The client pushes the message as shown below:

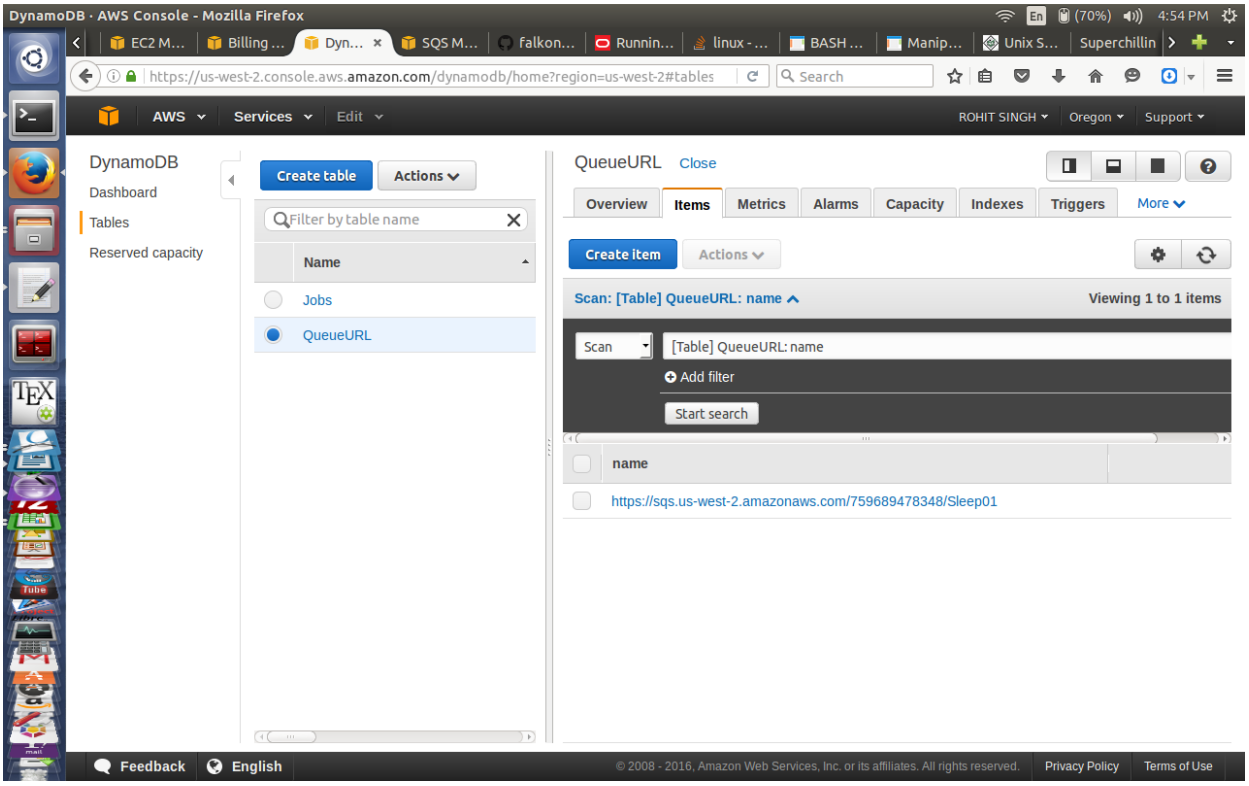
1 sleep 0
2 sleep 0
.....
10000 sleep 0

The first number present in each line provides each task with a unique id, this id can be used later by the workers to keep track of duplicated data. The SQS service is used to handle the queue of requests to load balance across multiple workers. The workers also create a response Queue, so that the client know that the tasks has been done. The following snapshot shows the SQS named Sleep01 with tasks of 0 ms.



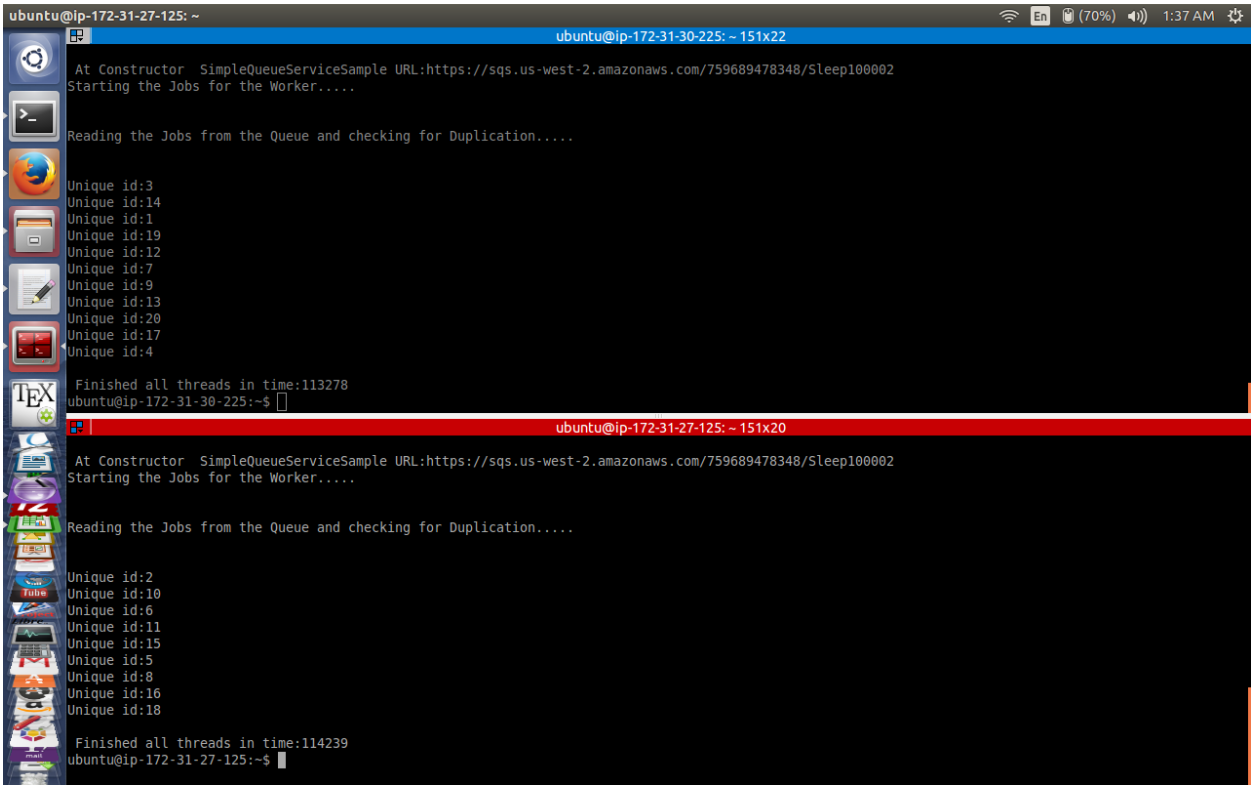
The workers retrieve the tasks from the SQS, whose url is present in the DynamoDB with table name **QueueURL**. The workers pull data from the SQS and starts executing them, independently without other workers' intervention. The workers keep on running until the SQS is empty.

The following is the snapshot of the table QueueURL where the Client has submitted the SQS's url for the workers to access the data.



Since, SQS cannot guarantee that messages from SQS are delivered exactly once, we use DynamoDB to keep track. I created a new table called **Jobs** which is common to all the workers. The jobs table has 2 attributes: **name** and **task**. The workers before executing the job, checks for the existence of the job in the table. Since, all the tasks are same, the unique id associated with the task can be used to identify the duplication. If a worker founds that the unique id already exists in the table, it will skip the task and pull a new data from the SQS. Else it will insert the task it is going to do and then run the task.

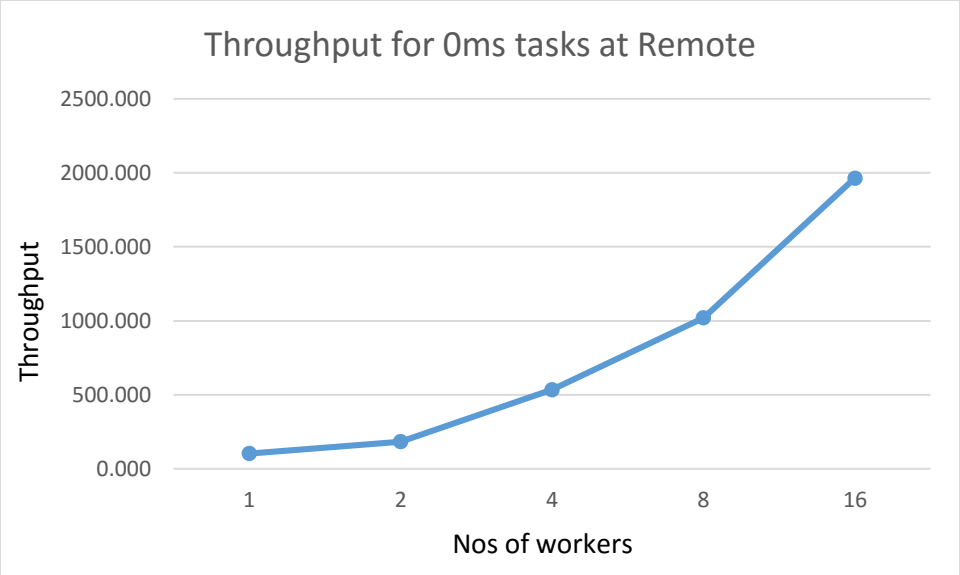
The following is a snapshot of 2 workers working on the same queue, however they do not get duplicate tasks from the queue. Since, the unique id is being monitored by the workers from the jobs table. This also throws light on the fact that all workers do not get equal number of tasks. The following is a simulation of 10s tasks with 10 tasks per worker. So, for 2 workers there are 20 tasks in the queue, out of which the 1st worker has done 11 unique tasks and the 2nd 9 unique tasks. However, it should have been 10 unique tasks each. This gives rise to lesser efficiency, since the 2nd worker will sit idle while the other finishes the extra task.



For evaluation we take experiments with 10,000 tasks of 0ms so as to find the throughput of the system.

Table 5 Remote experiment of 0ms tasks with 10000 tasks per thread

Nos of Threads	Time taken (in sec)	Throughput (tasks/sec)
1	97.02	103.072
2	54.45	183.655
4	18.7	534.759
8	9.8	1020.408
16	5.09	1964.637



From the graph above we can observe that the throughput increases with the increase in workers. This is obvious since the number of jobs done parallel will be the proportional to the number of workers. However, it is not always true, as there might be an uneven distribution of tasks amongst the workers, there might be workers who are sitting idle. Hence, we need to schedule equal or nearly equal amount of jobs to the workers.

To get the efficiency Tasks per worker per instruction we take 3 cases:

- 10ms tasks with 1000 tasks per worker
- 1s tasks with 100 tasks per worker
- 10s tasks with 10 tasks per worker

To calculate the efficiency, I used this formula:

$$Efficiency = \frac{Time\ per\ task * Tasks\ per\ workers}{Time\ taken} * 100 = \frac{Idle\ Time}{Time\ Taken} * 100$$

Table 6 Remote experiment of 10ms tasks with 1000 tasks per thread

Nos of Workers	Time taken (in sec)	Efficiency(%)
1	32.19	31.0655
2	56.76	35.2361
4	97.89	40.8622
8	181.17	44.1574
16	316.89	50.4907

Table 7 Remote experiment of 1s tasks with 100 tasks per thread

Nos of Workers	Time taken(in sec)	Efficiency(%)
1	155.5	64.3087
2	289.9	68.9893

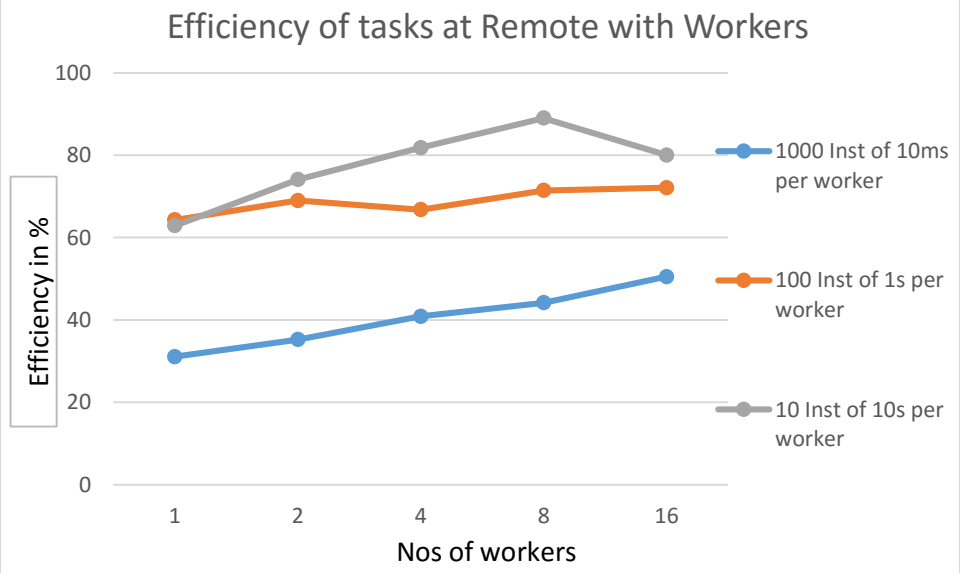
4	598.99	66.7791
8	1119.9	71.4349
16	2219.07	72.1023

Table 8 Remote experiment of 1s tasks with 10 tasks per thread

Nos of Workers	Time taken(in sec)	Efficiency(%)
1	159	62.8931
2	270	74.0741
4	489	81.7996
8	899	88.9878
16	1999	80.04

Since, the workers can be doing random number of tasks at any instance of time, hence I consider the maximum time taken by the workers, as the final time.

It is observed that as we increase the sleep time the time taken by the workers will increase. It is true because the idle time increases.

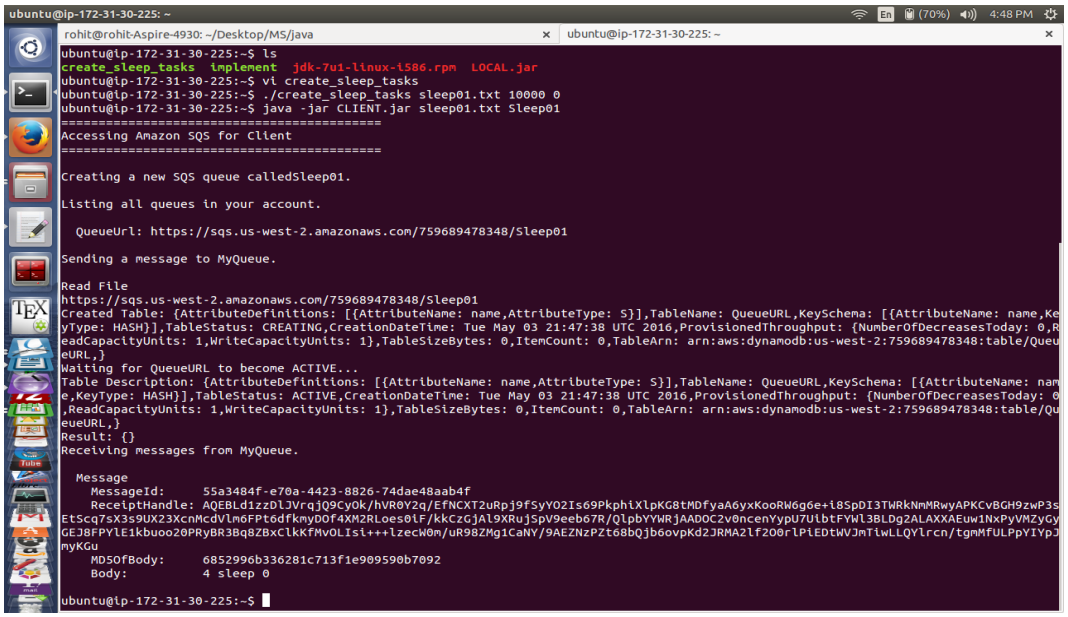


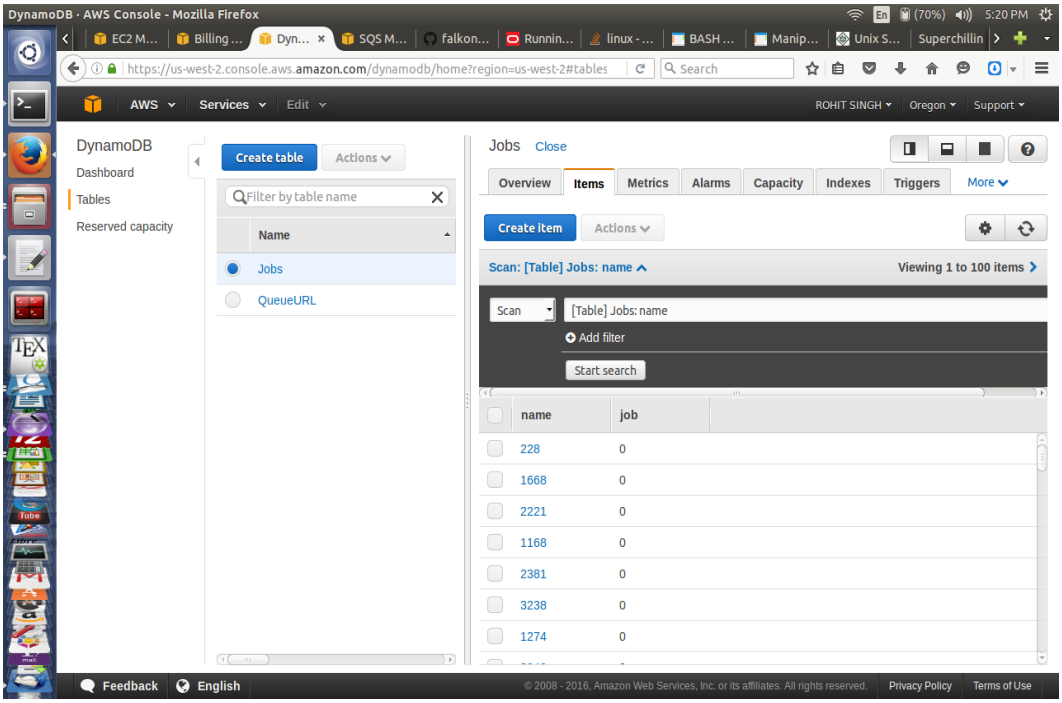
However, if we observe the Efficiency graph in Figure above, we see that the 10ms job with 1000 per worker has very less efficiency compared to the other two jobs. The reason being that there might be few workers who won't be working properly. Some workers might be sitting idle while others doing the maximum tasks. The jobs aren't getting distributed properly amongst the workers. With the increase in workers the number of total tasks also increases (i.e. **Total Task=Worker*Tasks per Worker**). So the time tasks will also increase fairly, as the works aren't distributed properly.

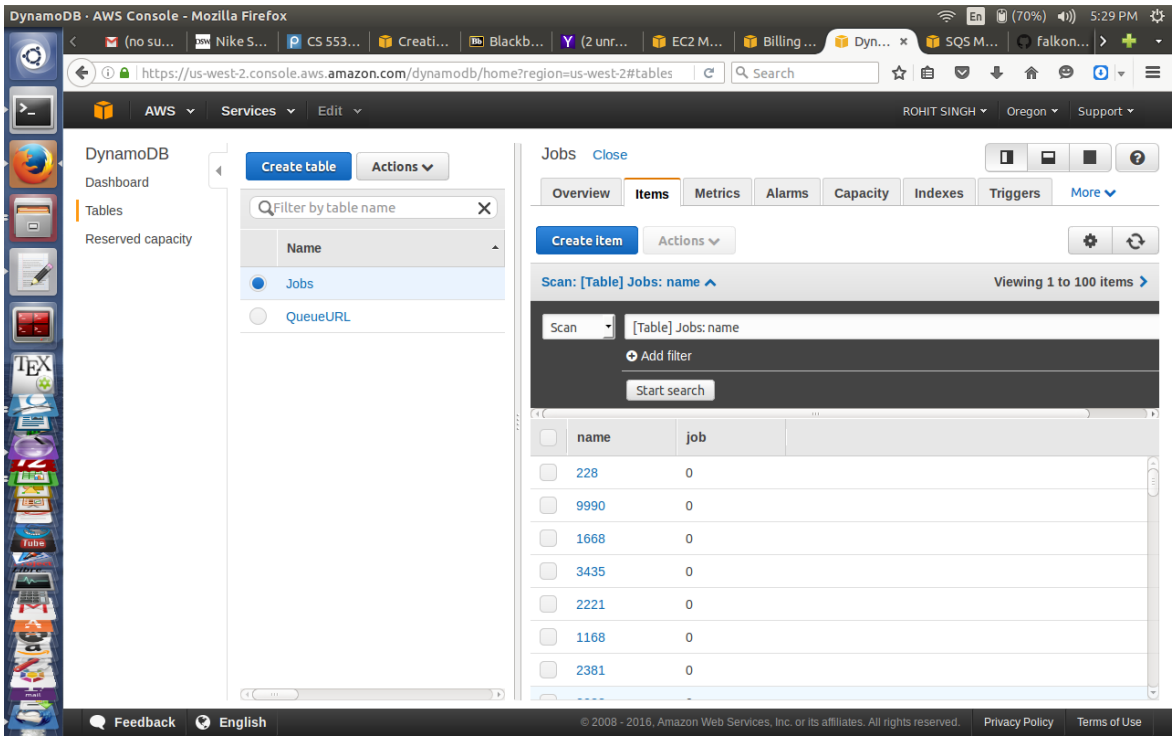
As the sleep task increases the probability that the workers will have tasks evenly distributed will be greater.

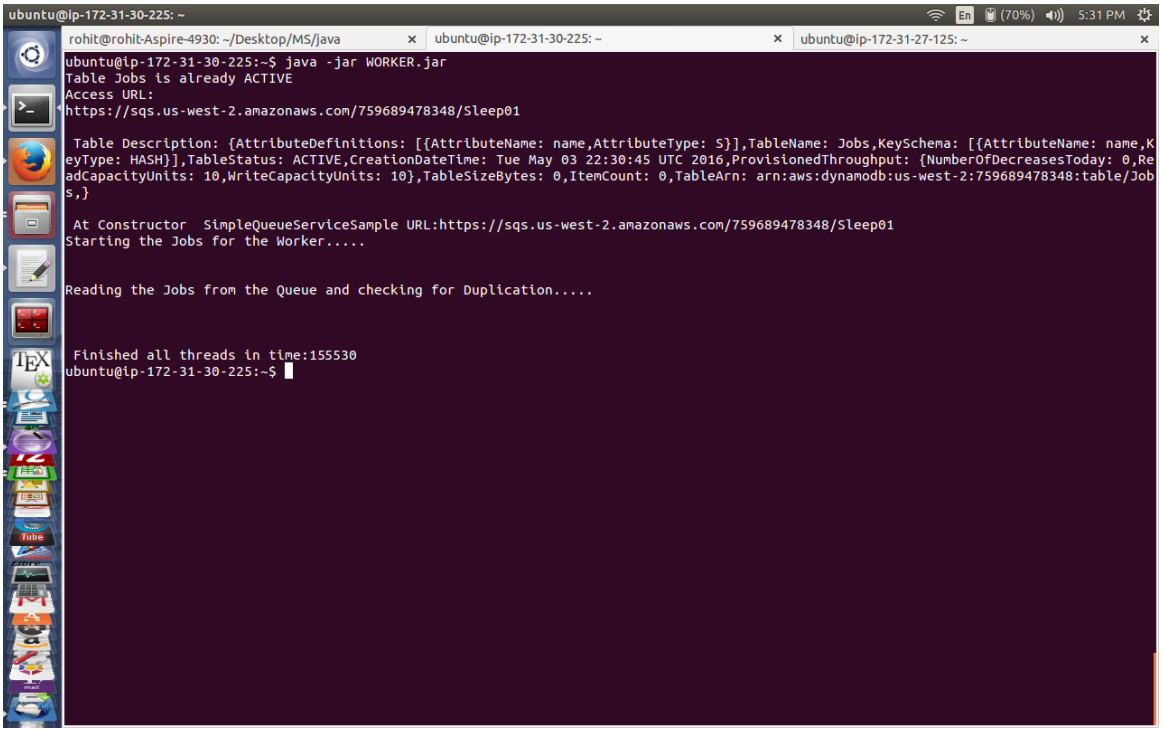
The other reason for low efficiency is due to the duplication check. As we keep track of the tasks done by putting it into the DynamoDB. A worker first checks if the task is already present in the table (Jobs). The Jobs is like a Log Book, where each worker reports the task it is going to do. Since, each task, though same, has a unique id associated with it, the id acts as a primary key for the workers to distinguish the unique tasks.

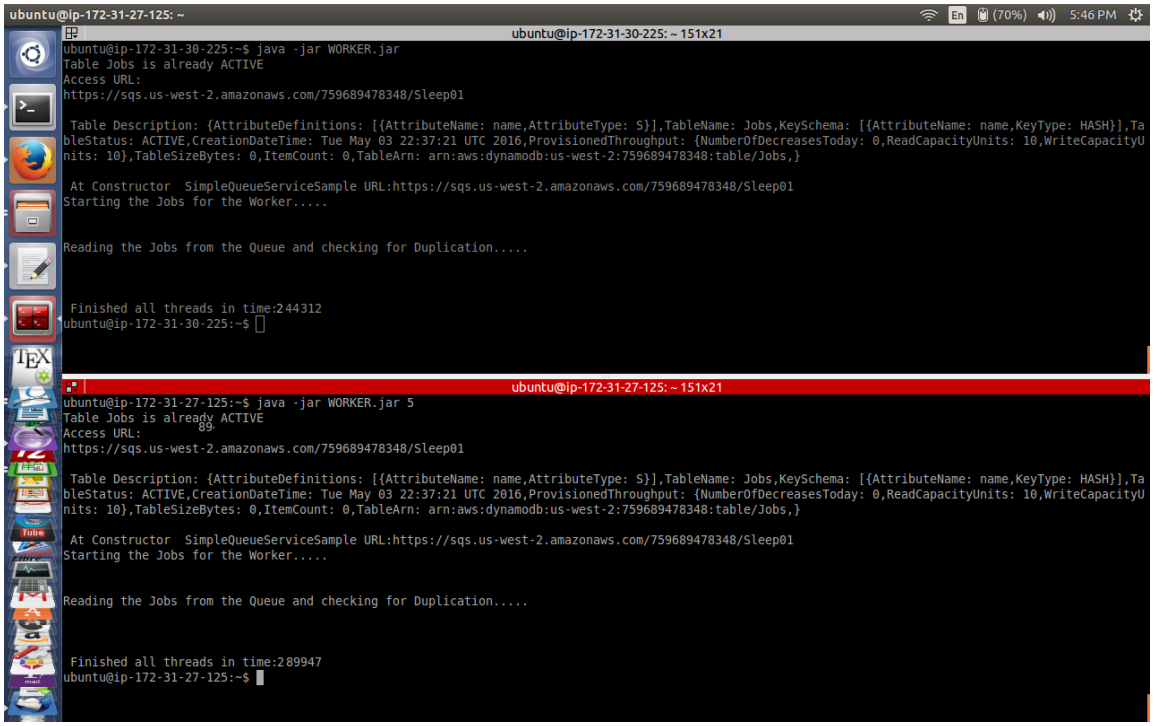
Snapshots

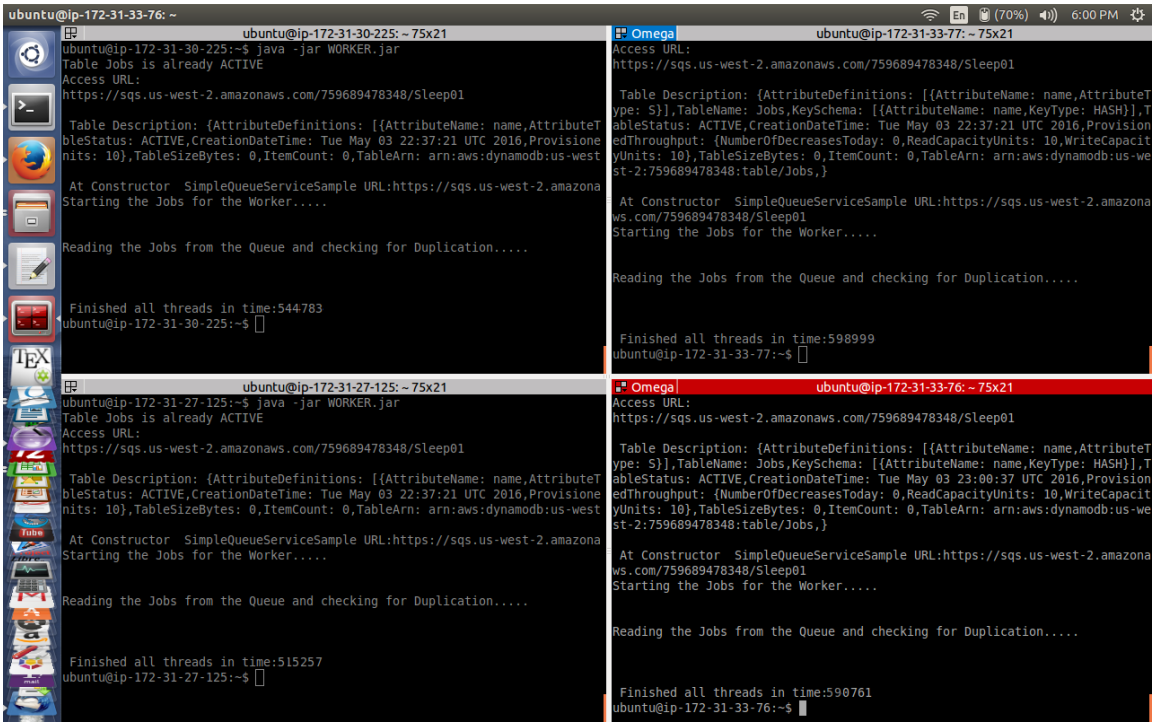


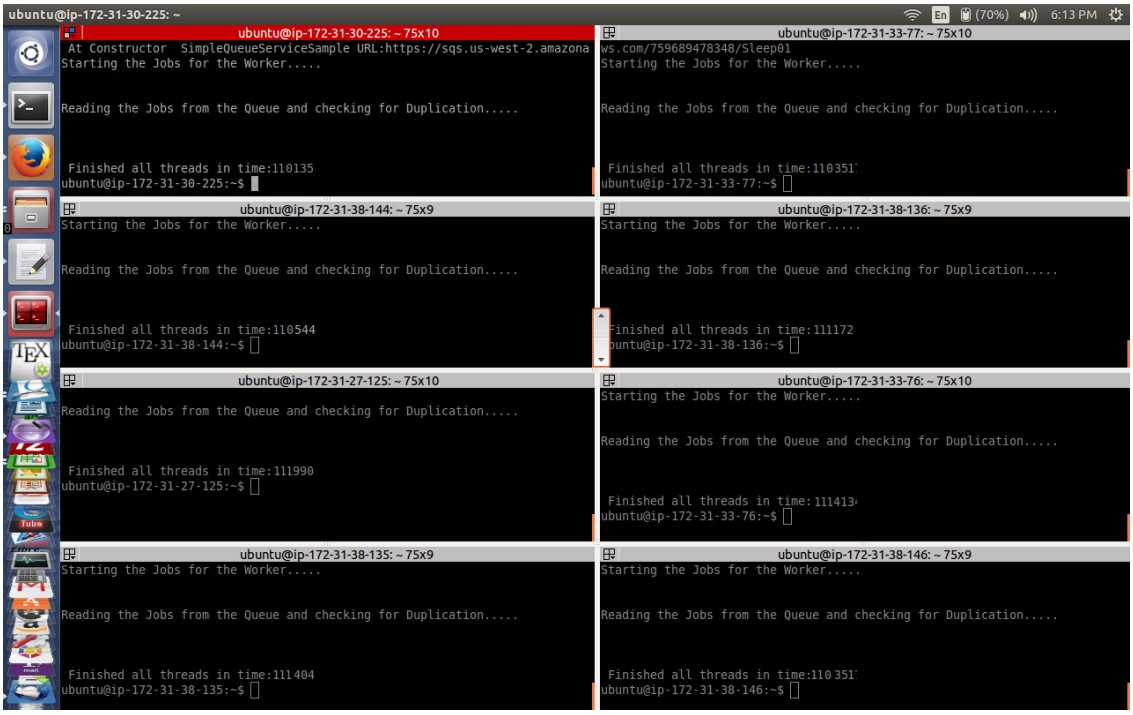


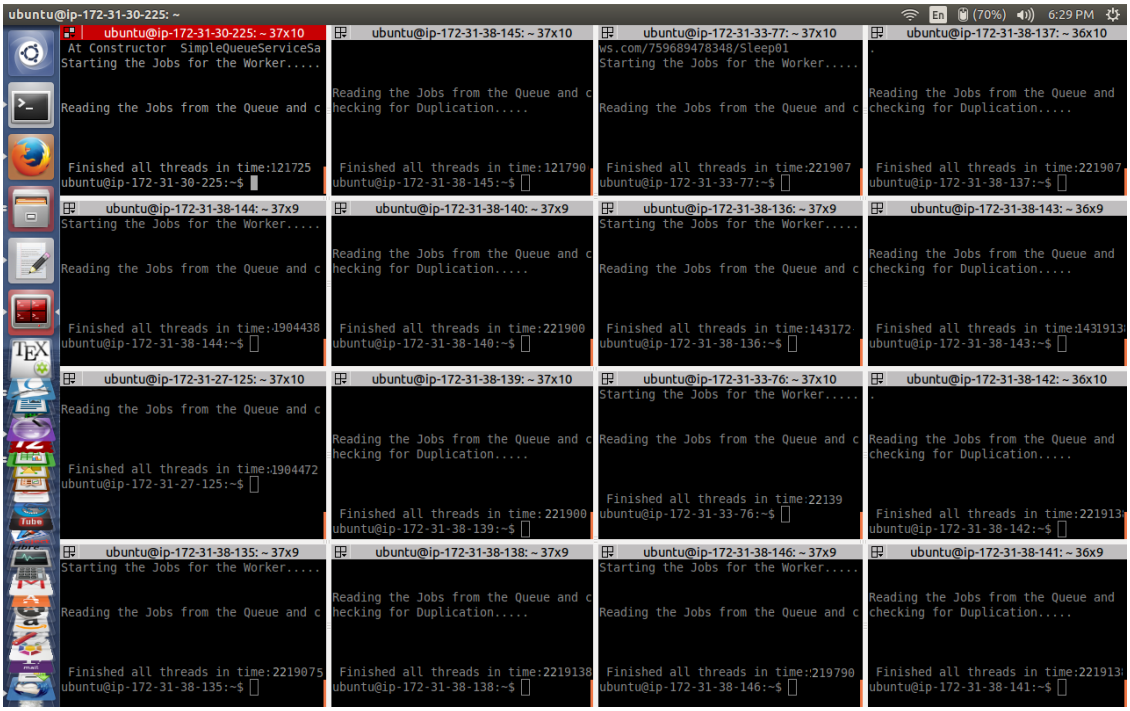

DynamoDB · AWS Console - Mozilla Firefox
https://us-west-2.console.aws.amazon.com/dynamodb/home?region=us-west-2#tables
AWS
Services
Edit
ROHIT SINGH
Oregon
Support
DynamoDB
Dashboard
Tables
Reserved capacity
Create table
Actions
Filter by table name
Name
Jobs
QueueURL
Jobs
Close
Overview
Items
Metrics
Alarms
Capacity
Indexes
Triggers
More
Create item
Actions
Scan: [Table] Jobs: name
Viewing 1 to 100 items
Scan
[Table] Jobs: name
Add filter
Start search
name
job
228 0
9990 0
1668 0
3435 0
2221 0
1168 0
2381 0
Feedback
English
© 2008 - 2016, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

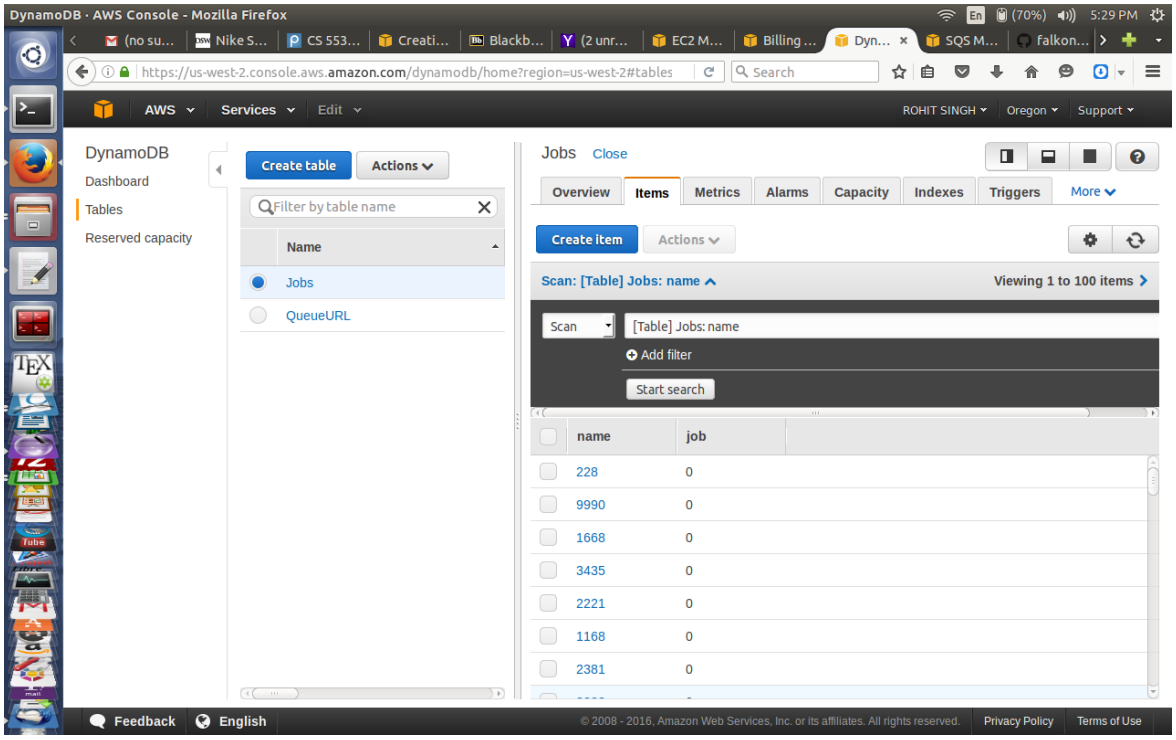

ubuntu@ip-172-31-30-225: ~
rohit@rohit-Aspire-4930: ~/Desktop/MS/java
ubuntu@ip-172-31-30-225: ~
ubuntu@ip-172-31-27-125: ~
ubuntu@ip-172-31-30-225:~$ java -jar WORKER.jar
Table Jobs is already ACTIVE
Access URL:
https://sqs.us-west-2.amazonaws.com/759689478348/Sleep01
Table Description: {AttributeDefinitions: [{AttributeName: name,AttributeType: S}],TableName: Jobs,KeySchema: [{AttributeName: name,KeyType: HASH}],TableStatus: ACTIVE,CreationDateTime: Tue May 03 22:30:45 UTC 2016,ProvisionedThroughput: {NumberOfDecreasesToday: 0,ReadCapacityUnits: 10,WriteCapacityUnits: 10},TableSizeBytes: 0,ItemCount: 0,TableArn: arn:aws:dynamodb:us-west-2:759689478348:table/Jobs,}
At Constructor SimpleQueueServiceSample URL:https://sqs.us-west-2.amazonaws.com/759689478348/Sleep01
Starting the Jobs for the Worker.....
Reading the Jobs from the Queue and checking for Duplication.....
Finished all threads in time:155530
ubuntu@ip-172-31-30-225:~$

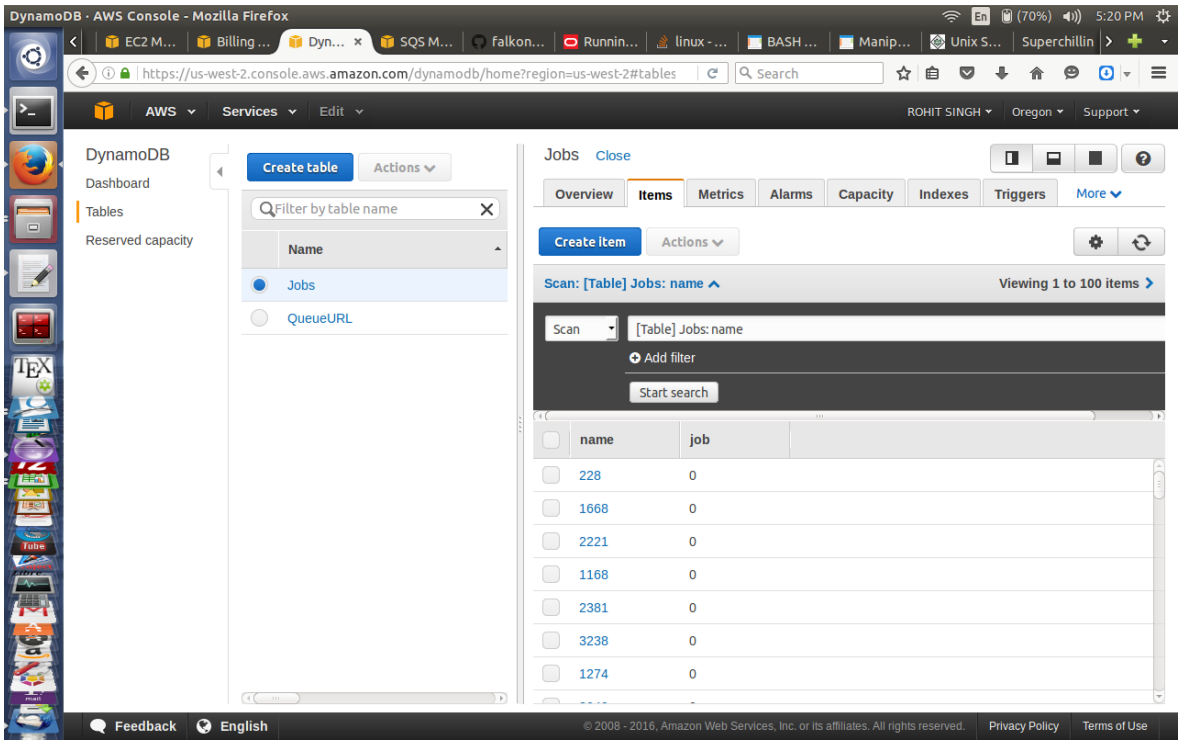

ubuntu@ip-172-31-27-125: ~
5:46 PM
ubuntu@ip-172-31-30-225: ~ 151x21
ubuntu@ip-172-31-30-225:~$ java -jar WORKER.jar
Table Jobs is already ACTIVE
Access URL:
https://sqs.us-west-2.amazonaws.com/759689478348/Sleep01
Table Description: {AttributeDefinitions: [{AttributeName: name,AttributeType: S}],TableName: Jobs,KeySchema: [{AttributeName: name,KeyType: HASH}],TableStatus: ACTIVE,CreationDateTime: Tue May 03 22:37:21 UTC 2016,ProvisionedThroughput: {NumberOfDecreasesToday: 0,ReadCapacityUnits: 10,WriteCapacityUnits: 10},TableSizeBytes: 0,ItemCount: 0,TableArn: arn:aws:dynamodb:us-west-2:759689478348:table/Jobs,}
At Constructor SimpleQueueServiceSample URL:https://sqs.us-west-2.amazonaws.com/759689478348/Sleep01
Starting the Jobs for the Worker.....
Reading the Jobs from the Queue and checking for Duplication.....
Finished all threads in time:244312
ubuntu@ip-172-31-30-225:~$
ubuntu@ip-172-31-27-125: ~ 151x21
ubuntu@ip-172-31-27-125:~$ java -jar WORKER.jar 5
Table Jobs is already ACTIVE
Access URL:
https://sqs.us-west-2.amazonaws.com/759689478348/Sleep01
Table Description: {AttributeDefinitions: [{AttributeName: name,AttributeType: S}],TableName: Jobs,KeySchema: [{AttributeName: name,KeyType: HASH}],TableStatus: ACTIVE,CreationDateTime: Tue May 03 22:37:21 UTC 2016,ProvisionedThroughput: {NumberOfDecreasesToday: 0,ReadCapacityUnits: 10,WriteCapacityUnits: 10},TableSizeBytes: 0,ItemCount: 0,TableArn: arn:aws:dynamodb:us-west-2:759689478348:table/Jobs,}
At Constructor SimpleQueueServiceSample URL:https://sqs.us-west-2.amazonaws.com/759689478348/Sleep01
Starting the Jobs for the Worker.....
Reading the Jobs from the Queue and checking for Duplication.....
Finished all threads in time:289947
ubuntu@ip-172-31-27-125:~$

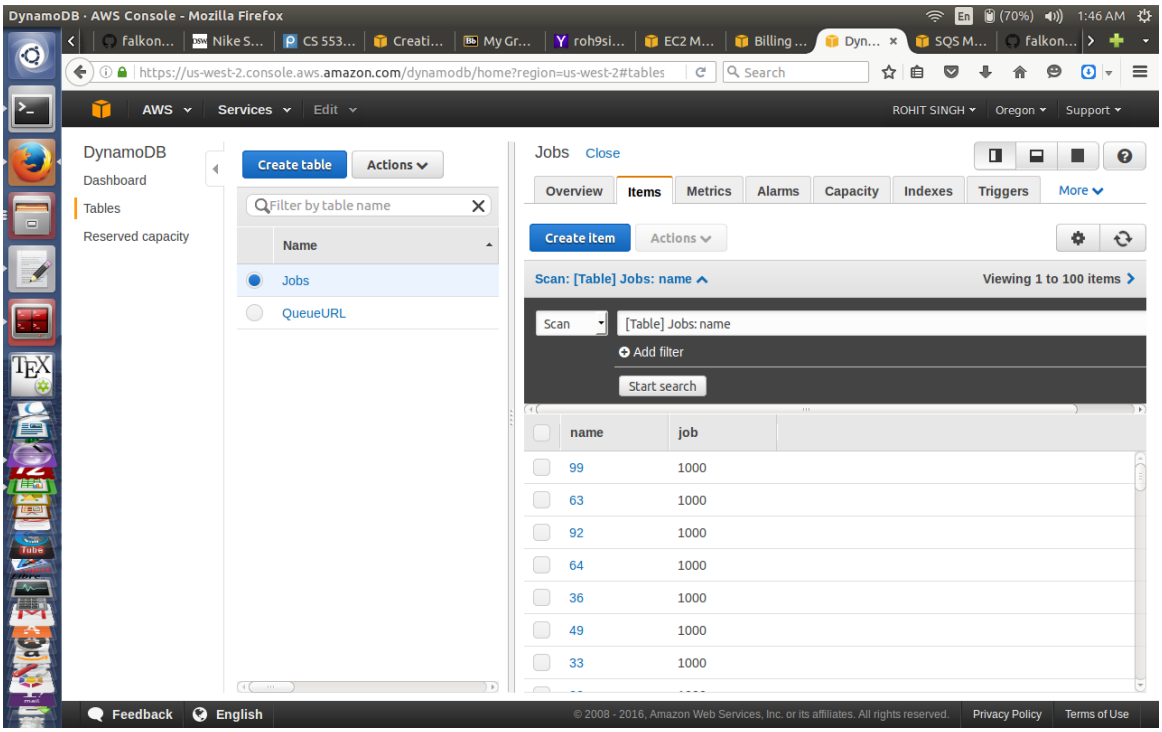

ubuntu@ip-172-31-33-76: ~
6:00 PM
ubuntu@ip-172-31-30-225: ~ 75x21
ubuntu@ip-172-31-30-225:~$ java -jar WORKER.jar
Table Jobs is already ACTIVE
Access URL:
https://sqs.us-west-2.amazonaws.com/759689478348/Sleep01
Reading the Jobs from the Queue and checking for Duplication.....
Finished all threads in time:544783
ubuntu@ip-172-31-30-225:~$
Omega
ubuntu@ip-172-31-33-77: ~ 75x21
Access URL:
https://sqs.us-west-2.amazonaws.com/759689478348/Sleep01
Reading the Jobs from the Queue and checking for Duplication.....
Finished all threads in time:598999
ubuntu@ip-172-31-33-77:~$
ubuntu@ip-172-31-27-125: ~ 75x21
ubuntu@ip-172-31-27-125:~$ java -jar WORKER.jar
Table Jobs is already ACTIVE
Access URL:
https://sqs.us-west-2.amazonaws.com/759689478348/Sleep01
Reading the Jobs from the Queue and checking for Duplication.....
Finished all threads in time:515257
ubuntu@ip-172-31-27-125:~$
Omega
ubuntu@ip-172-31-33-76: ~ 75x21
Access URL:
https://sqs.us-west-2.amazonaws.com/759689478348/Sleep01
Reading the Jobs from the Queue and checking for Duplication.....
Finished all threads in time:590761
ubuntu@ip-172-31-33-76:~$

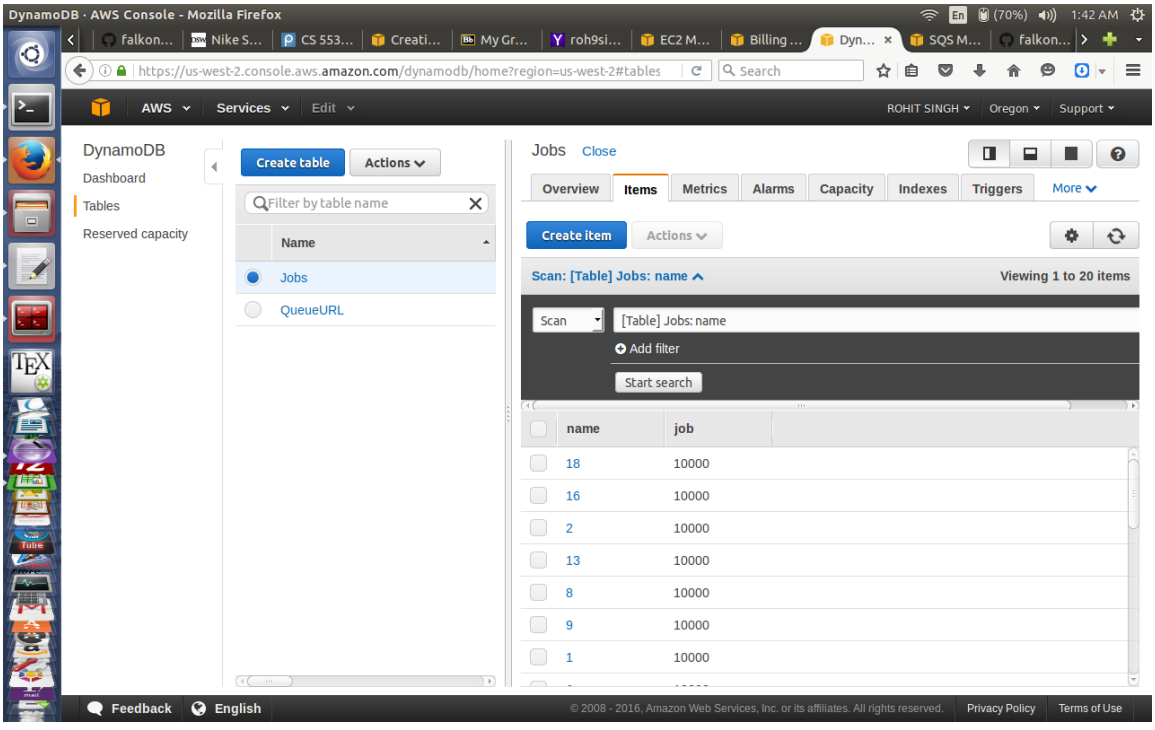

ubuntu@ip-172-31-30-225: ~
6:13 PM
ubuntu@ip-172-31-30-225: ~ 75x10
At Constructor SimpleQueueServiceSample URL:https://sqs.us-west-2.amazona
Starting the Jobs for the Worker.....
Reading the Jobs from the Queue and checking for Duplication.....
Finished all threads in time:110135
ubuntu@ip-172-31-30-225:~$
ubuntu@ip-172-31-33-77: ~ 75x10
ws.com/759689478348/Sleep01
Starting the Jobs for the Worker.....
Reading the Jobs from the Queue and checking for Duplication.....
Finished all threads in time:110351
ubuntu@ip-172-31-33-77:~$
ubuntu@ip-172-31-38-144: ~ 75x9
Starting the Jobs for the Worker.....
Reading the Jobs from the Queue and checking for Duplication.....
Finished all threads in time:110544
ubuntu@ip-172-31-38-144:~$
ubuntu@ip-172-31-38-136: ~ 75x9
Starting the Jobs for the Worker.....
Reading the Jobs from the Queue and checking for Duplication.....
Finished all threads in time:111172
ubuntu@ip-172-31-38-136:~$
ubuntu@ip-172-31-27-125: ~ 75x10
Reading the Jobs from the Queue and checking for Duplication.....
Finished all threads in time:111990
ubuntu@ip-172-31-27-125:~$
ubuntu@ip-172-31-33-76: ~ 75x10
Starting the Jobs for the Worker.....
Reading the Jobs from the Queue and checking for Duplication.....
Finished all threads in time:111413
ubuntu@ip-172-31-33-76:~$
ubuntu@ip-172-31-38-135: ~ 75x9
Starting the Jobs for the Worker.....
Reading the Jobs from the Queue and checking for Duplication.....
Finished all threads in time:111404
ubuntu@ip-172-31-38-135:~$
ubuntu@ip-172-31-38-146: ~ 75x9
Starting the Jobs for the Worker.....
Reading the Jobs from the Queue and checking for Duplication.....
Finished all threads in time:110351
ubuntu@ip-172-31-38-146:~$


ubuntu@ip-172-31-30-225: ~
6:29 PM
ubuntu@ip-172-31-30-225: ~ 37x10
At Constructor SimpleQueueServiceSa
Starting the Jobs for the Worker.....
Reading the Jobs from the Queue and c
Finished all threads in time:121725
ubuntu@ip-172-31-30-225:~$
ubuntu@ip-172-31-38-145: ~ 37x10
Reading the Jobs from the Queue and c
hecking for Duplication.....
Finished all threads in time:121790
ubuntu@ip-172-31-38-145:~$
ubuntu@ip-172-31-33-77: ~ 37x10
ws.com/759689478348/Sleep01
Starting the Jobs for the Worker.....
Reading the Jobs from the Queue and c
Finished all threads in time:221907
ubuntu@ip-172-31-33-77:~$
ubuntu@ip-172-31-38-137: ~ 36x10
Reading the Jobs from the Queue and
checking for Duplication.....
Finished all threads in time:221907
ubuntu@ip-172-31-38-137:~$
ubuntu@ip-172-31-38-144: ~ 37x9
Starting the Jobs for the Worker.....
Reading the Jobs from the Queue and c
Finished all threads in time:1904438
ubuntu@ip-172-31-38-144:~$
ubuntu@ip-172-31-38-140: ~ 37x9
Reading the Jobs from the Queue and c
hecking for Duplication.....
Finished all threads in time:221900
ubuntu@ip-172-31-38-140:~$
ubuntu@ip-172-31-38-136: ~ 37x9
Starting the Jobs for the Worker.....
Reading the Jobs from the Queue and c
Finished all threads in time:143172
ubuntu@ip-172-31-38-136:~$
ubuntu@ip-172-31-38-143: ~ 36x9
Reading the Jobs from the Queue and
checking for Duplication.....
Finished all threads in time:1431913
ubuntu@ip-172-31-38-143:~$
ubuntu@ip-172-31-27-125: ~ 37x10
Reading the Jobs from the Queue and c
Finished all threads in time:1904472
ubuntu@ip-172-31-27-125:~$
ubuntu@ip-172-31-38-139: ~ 37x10
Reading the Jobs from the Queue and c
hecking for Duplication.....
Finished all threads in time:221900
ubuntu@ip-172-31-38-139:~$
ubuntu@ip-172-31-33-76: ~ 37x10
Starting the Jobs for the Worker.....
Reading the Jobs from the Queue and c
Finished all threads in time:22139
ubuntu@ip-172-31-33-76:~$
ubuntu@ip-172-31-38-142: ~ 36x10
Reading the Jobs from the Queue and
checking for Duplication.....
Finished all threads in time:221913
ubuntu@ip-172-31-38-142:~$
ubuntu@ip-172-31-38-135: ~ 37x9
Starting the Jobs for the Worker.....
Reading the Jobs from the Queue and c
Finished all threads in time:2219075
ubuntu@ip-172-31-38-135:~$
ubuntu@ip-172-31-38-138: ~ 37x9
Reading the Jobs from the Queue and c
hecking for Duplication.....
Finished all threads in time:2219138
ubuntu@ip-172-31-38-138:~$
ubuntu@ip-172-31-38-146: ~ 37x9
Starting the Jobs for the Worker.....
Reading the Jobs from the Queue and c
Finished all threads in time:219790
ubuntu@ip-172-31-38-146:~$
ubuntu@ip-172-31-38-141: ~ 36x9
Reading the Jobs from the Queue and
checking for Duplication.....
Finished all threads in time:221913
ubuntu@ip-172-31-38-141:~$


DynamoDB · AWS Console - Mozilla Firefox
5:29 PM
https://us-west-2.console.aws.amazon.com/dynamodb/home?region=us-west-2#tables
AWS
Services
Edit
ROHIT SINGH
Oregon
Support
DynamoDB
Dashboard
Tables
Reserved capacity
Create table
Actions
Filter by table name
Name
Jobs
QueueURL
Jobs
Close
Overview
Items
Metrics
Alarms
Capacity
Indexes
Triggers
More
Create item
Actions
Scan: [Table] Jobs: name
Viewing 1 to 100 items
Scan
[Table] Jobs: name
Add filter
Start search
name
job
228 0
9990 0
1668 0
3435 0
2221 0
1168 0
2381 0
Feedback
English
© 2008 - 2016, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use


DynamoDB · AWS Console - Mozilla Firefox
5:20 PM
https://us-west-2.console.aws.amazon.com/dynamodb/home?region=us-west-2#tables
AWS
Services
Edit
ROHIT SINGH
Oregon
Support
DynamoDB
Dashboard
Tables
Reserved capacity
Create table
Actions
Filter by table name
Name
Jobs
QueueURL
Jobs
Close
Overview
Items
Metrics
Alarms
Capacity
Indexes
Triggers
More
Create item
Actions
Scan: [Table] Jobs: name
Viewing 1 to 100 items
Scan
[Table] Jobs: name
Add filter
Start search
name
job
228 0
1668 0
2221 0
1168 0
2381 0
3238 0
1274 0
Feedback
English
© 2008 - 2016, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use


DynamoDB · AWS Console - Mozilla Firefox
1:46 AM
https://us-west-2.console.aws.amazon.com/dynamodb/home?region=us-west-2#tables
AWS
Services
Edit
ROHIT SINGH
Oregon
Support
DynamoDB
Dashboard
Tables
Reserved capacity
Create table
Actions
Filter by table name
Name
Jobs
QueueURL
Jobs
Close
Overview
Items
Metrics
Alarms
Capacity
Indexes
Triggers
More
Create item
Actions
Scan: [Table] Jobs: name
Viewing 1 to 100 items
Scan
[Table] Jobs: name
Add filter
Start search
name
job
99 1000
63 1000
92 1000
64 1000
36 1000
49 1000
33 1000
Feedback
English
© 2008 - 2016, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use


DynamoDB · AWS Console - Mozilla Firefox
1:42 AM
https://us-west-2.console.aws.amazon.com/dynamodb/home?region=us-west-2#tables
AWS
Services
Edit
ROHIT SINGH
Oregon
Support
DynamoDB
Dashboard
Tables
Reserved capacity
Create table
Actions
Filter by table name
Name
Jobs
QueueURL
Jobs
Close
Overview
Items
Metrics
Alarms
Capacity
Indexes
Triggers
More
Create item
Actions
Scan: [Table] Jobs: name
Viewing 1 to 20 items
Scan
[Table] Jobs: name
Add filter
Start search
name
job
18 10000
16 10000
2 10000
13 10000
8 10000
9 10000
1 10000
Feedback
English
© 2008 - 2016, Amazon Web Services, Inc. or its affiliates. All rights reserved.
Privacy Policy
Terms of Use

4. Animoto Clone [Extra Credit]

Animoto is a cloud-based video creation service that produces video from photos, video clips, and music into video slideshows [7]. In this assignment we need to make a clone of the same using the cloud services provided by AWS namely SQS and S3. For this experiment I have used the **ffmpeg** to create a video out of the photos which have been downloaded from the urls provided by the file as an input, these set of urls will be sent by the Client for the S3 to download using **wget**. This work is to be done by the workers and after each work is done by a worker it submit the video into S3, and returns the link to the S3 to a response queue (SQS) for the client to view. There are about 160 such tasks which is to be done by a set of workers. The worker set varies as 1,2,4,8 and 16.

FFmpeg is a free software project that produces libraries and programs for handling multimedia data. FFmpeg includes libavcodec, an audio/video codec library used by several other projects, libavformat, an audio/video container mux and de-mux library, and the ffmpeg command line program for transcoding multimedia files [8-12]. The following code combines all the images named as image%d.jpg, where %d is a number present in the source path folder.

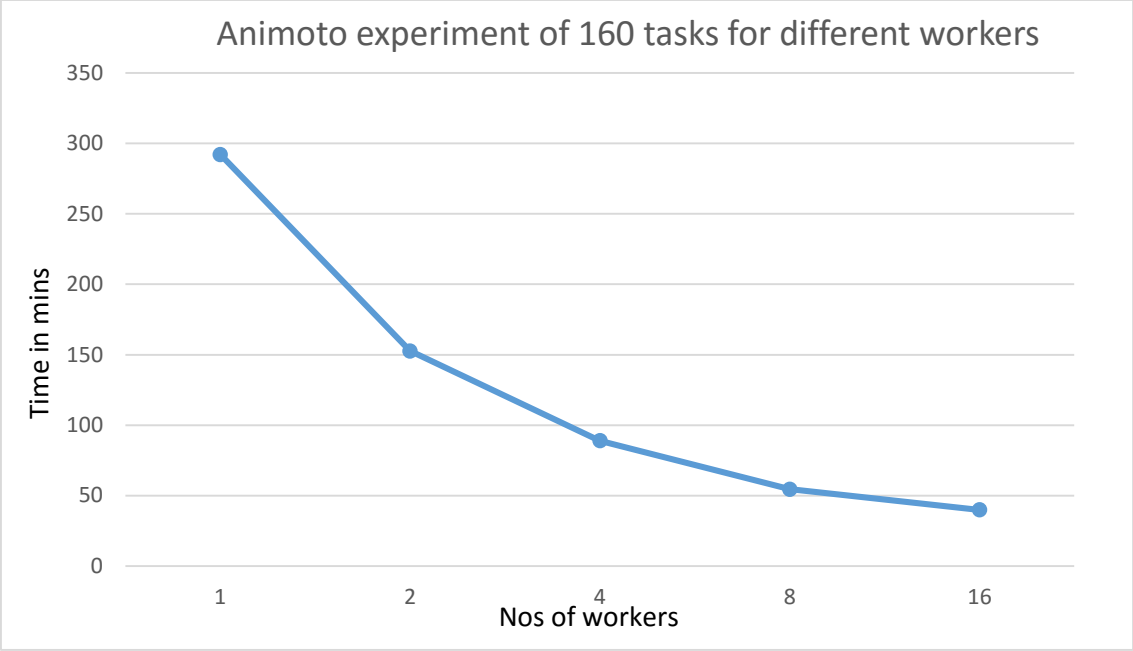
ffmpeg -f image2 -i <source path>/image%d.jpg <destination path>/output.mpg

All the images are of 1920x1080 resolution. The urls provided by the Client are all 1920x1080 resolution, since we want to make a 60 seconds long video clip with HD 1080P resolution. To make the video 60 sec long I have slowed the fps (frames per second), by using the following code:

ffmpeg -i output.mpg -filter:v "setpts=10*PTS" output.mpg

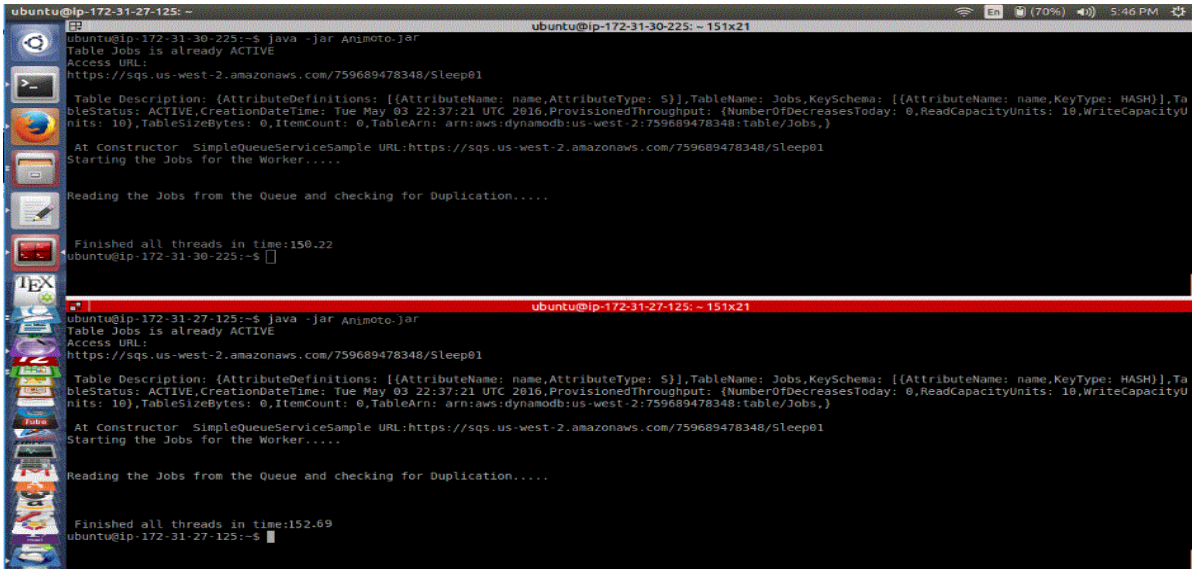
Table 9 Animoto experiment of 160 tasks with 60 images

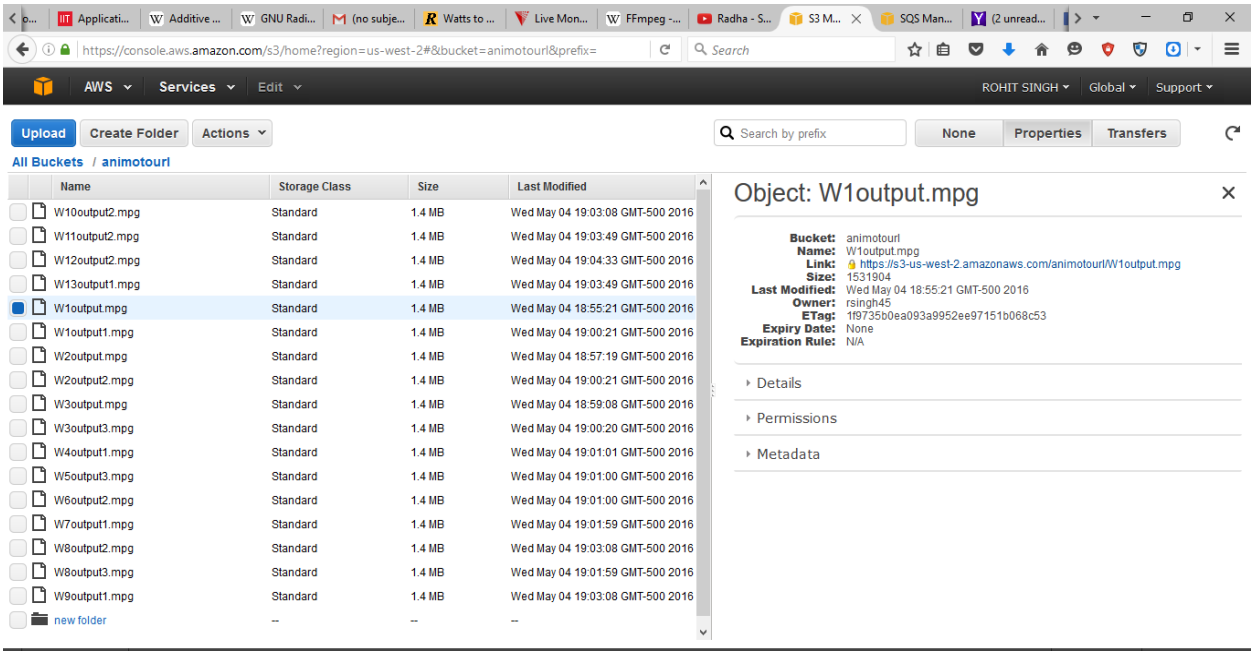
Nos of Threads	Time taken(in min)
1	292.13
2	152.69
4	88.98
8	54.51
16	39.91

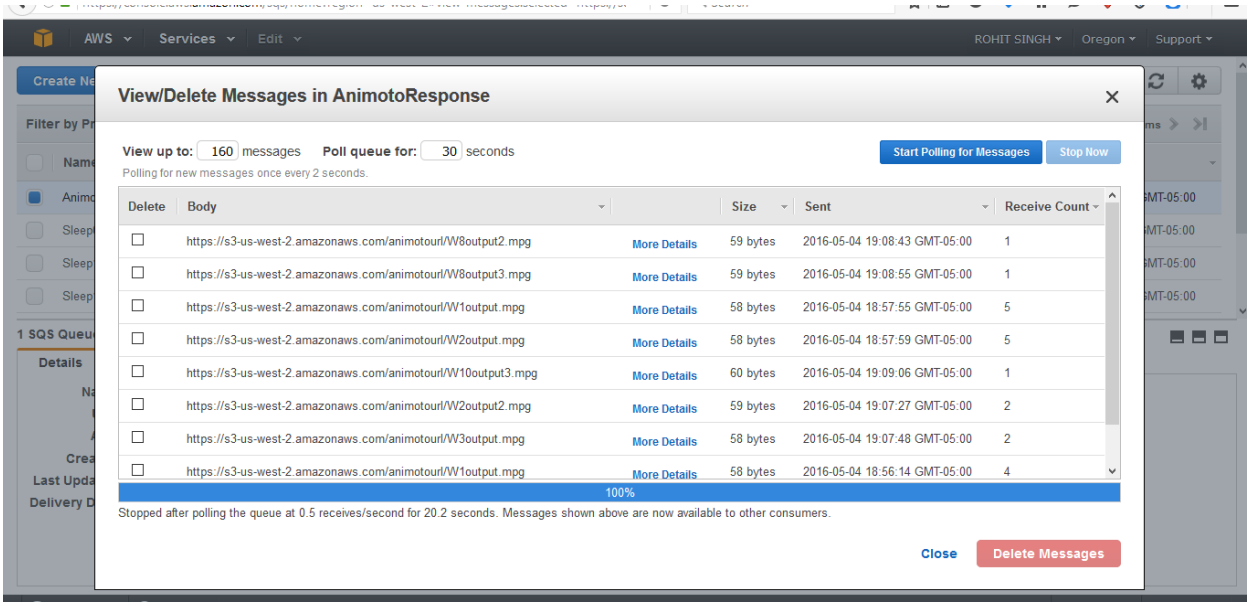


The experiment was performed with 160 such tasks with 1 to 16 workers. The time decreases with increasing number of workers as shown in Figure , since the tasks get distributed. The major time required to do the task is to download the HD quality pictures. So, the timing is largely governed by the network bandwidth at that instance.

Snapshots




https://console.aws.amazon.com/s3/home?region=us-west-2#&bucket=animotourl&prefix=
AWS
Services
Edit
ROHIT SINGH
Global
Support
Upload
Create Folder
Actions
Search by prefix
None
Properties
Transfers
All Buckets / animotourl
Name
Storage Class
Size
Last Modified
W10output2.mpg Standard 1.4 MB Wed May 04 19:03:08 GMT-500 2016
W11output2.mpg Standard 1.4 MB Wed May 04 19:03:49 GMT-500 2016
W12output2.mpg Standard 1.4 MB Wed May 04 19:04:33 GMT-500 2016
W13output1.mpg Standard 1.4 MB Wed May 04 19:03:49 GMT-500 2016
W1output.mpg Standard 1.4 MB Wed May 04 18:55:21 GMT-500 2016
W1output1.mpg Standard 1.4 MB Wed May 04 19:00:21 GMT-500 2016
W2output.mpg Standard 1.4 MB Wed May 04 18:57:19 GMT-500 2016
W2output2.mpg Standard 1.4 MB Wed May 04 19:00:21 GMT-500 2016
W3output.mpg Standard 1.4 MB Wed May 04 18:59:08 GMT-500 2016
W3output3.mpg Standard 1.4 MB Wed May 04 19:00:20 GMT-500 2016
W4output1.mpg Standard 1.4 MB Wed May 04 19:01:01 GMT-500 2016
W5output3.mpg Standard 1.4 MB Wed May 04 19:01:00 GMT-500 2016
W6output2.mpg Standard 1.4 MB Wed May 04 19:01:00 GMT-500 2016
W7output1.mpg Standard 1.4 MB Wed May 04 19:01:59 GMT-500 2016
W8output2.mpg Standard 1.4 MB Wed May 04 19:03:08 GMT-500 2016
W8output3.mpg Standard 1.4 MB Wed May 04 19:01:59 GMT-500 2016
W9output1.mpg Standard 1.4 MB Wed May 04 19:03:08 GMT-500 2016
new folder
Object: W1output.mpg
Bucket: animotourl
Name: W1output.mpg
Link: https://s3-us-west-2.amazonaws.com/animotourl/W1output.mpg
Size: 1531904
Last Modified: Wed May 04 18:55:21 GMT-500 2016
Owner: rsingh45
ETag: f19735b0ea093a9952ee97151b068c53
Expiry Date: None
Expiration Rule: N/A
Details
Permissions
Metadata


AWS
Services
Edit
ROHIT SINGH
Oregon
Support
View/Delete Messages in AnimotoResponse
View up to: 160 messages
Poll queue for: 30 seconds
Start Polling for Messages
Stop Now
Polling for new messages once every 2 seconds.
Delete
Body
Size
Sent
Receive Count
https://s3-us-west-2.amazonaws.com/animotourl/W8output2.mpg More Details 59 bytes 2016-05-04 19:08:43 GMT-05:00 1
https://s3-us-west-2.amazonaws.com/animotourl/W8output3.mpg More Details 59 bytes 2016-05-04 19:08:55 GMT-05:00 1
https://s3-us-west-2.amazonaws.com/animotourl/W1output.mpg More Details 58 bytes 2016-05-04 18:57:55 GMT-05:00 5
https://s3-us-west-2.amazonaws.com/animotourl/W2output.mpg More Details 58 bytes 2016-05-04 18:57:59 GMT-05:00 5
https://s3-us-west-2.amazonaws.com/animotourl/W10output3.mpg More Details 60 bytes 2016-05-04 19:09:06 GMT-05:00 1
https://s3-us-west-2.amazonaws.com/animotourl/W2output2.mpg More Details 59 bytes 2016-05-04 19:07:27 GMT-05:00 2
https://s3-us-west-2.amazonaws.com/animotourl/W3output.mpg More Details 58 bytes 2016-05-04 19:07:48 GMT-05:00 2
https://s3-us-west-2.amazonaws.com/animotourl/W1output.mpg More Details 58 bytes 2016-05-04 18:56:14 GMT-05:00 4
100%
Stopped after polling the queue at 0.5 receives/second for 20.2 seconds. Messages shown above are now available to other consumers.
Close
Delete Messages

5. Reference

[1] http://docs.aws.amazon.com/AWSToolkitEclipse/latest/ug/tke_setup_install.html
[2] http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/AppendixSampleDataCodeJava.html
[3] http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/ScanJavaDocumentAPI.html
[4] http://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Expressions.SpecifyingConditions.htmlhttp://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Expressions.SpecifyingConditions.html
[5] http://www.avajava.com/tutorials/lessons/how-do-i-save-an-image-from-a-url-to-a-file.html
[6] http://askubuntu.com/questions/699502/ffmpeg-command-not-found
[7] https://en.wikipedia.org/wiki/Animoto
[8] https://en.wikipedia.org/wiki/FFmpeg
[9] http://www.ffmpeg.org/faq.html#How-do-I-encode-single-pictures-into-movies_003f
[10] https://trac.ffmpeg.org/wiki/How%20to%20speed%20up%20/%20slow%20down%20a%20video
[11] http://stackoverflow.com/questions/24961127/ffmpeg-create-video-from-images
[12] http://www.mkyong.com/java/how-to-execute-shell-command-from-java/
[13] http://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/API_CreateQueue.html
[14] http://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/API_GetQueueUrl.html
[15] http://docs.aws.amazon.com/AWSSimpleQueueService/latest/APIReference/API_ReceiveMessage.html
[16] http://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_GetItem.html
[17] http://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_Scan.html
[18] http://docs.aws.amazon.com/AmazonS3/latest/API/Welcome.html