Manual

CloudKon clone with Amazon EC2, S3, SQS, and DynamoDB

Pavankumar Shetty
CWID-A20354961
CS-553 Spring 2016
Main Campus

MANUAL

1. **Place all the JAR files (which are submitted in blackboard under the folder name, "executables" in a subfolder of source code folder.) in your EC2 t2 micro instance.**
    a. **For client local, Place the jar file named "RunLocal.jar" in t2-micro instance**

    b. **Place all the workload files for running.**
        i. **10KSleepTasks.txt**
        ii. **In "Sleep10Tasks Text Files" folder and 16000Sleep10Tasks.txt, 8000Sleep10Tasks.txt, 4000Sleep10Tasks.txt, 2000Sleep10Tasks.txt, 1000Sleep10Tasks.txt.**
        iii. **In "Sleep1000Tasks Text Files" folder and 1600Sleep1000Tasks.txt, 800Sleep1000Tasks.txt, 400Sleep1000Tasks.txt, 200Sleep1000Tasks.txt, 100Sleep1000Tasks.txt.**
        iv. **In "Sleep10000Tasks Text Files" folder and 160Sleep10000Tasks.txt, 80Sleep10000Tasks.txt, 40Sleep10000Tasks.txt, 20Sleep10000Tasks.txt, 10Sleep10000Tasks.txt.**

    c. **Run the experiment locally,**
        i. **# 10k Sleep 0**

        java -jar RunLocal.jar client -s LOCAL -t 1 -w ./10KSleepTasks.txt
        java -jar RunLocal.jar client -s LOCAL -t 2 -w ./10KSleepTasks.txt
        java -jar RunLocal.jar client -s LOCAL -t 4 -w ./10KSleepTasks.txt
        java -jar RunLocal.jar client -s LOCAL -t 8 -w ./10KSleepTasks.txt
        java -jar RunLocal.jar client -s LOCAL -t 16 -w ./10KSleepTasks.txt
        ii. **# 1000k Sleep 10ms**

        java -jar RunLocal.jar client -s LOCAL -t 1 -w ./Sleep10Tasks\ Text\ Files/16000Sleep10Tasks.txt
        java -jar RunLocal.jar client -s LOCAL -t 8 -w ./Sleep10Tasks\ Text\ Files/8000Sleep10Tasks.txt
        java -jar RunLocal.jar client -s LOCAL -t 4 -w ./Sleep10Tasks\ Text\ Files/4000Sleep10Tasks.txt
        java -jar RunLocal.jar client -s LOCAL -t 2 -w ./Sleep10Tasks\ Text\ Files/2000Sleep10Tasks.txt
        java -jar RunLocal.jar client -s LOCAL -t 1 -w ./Sleep10Tasks\ Text\ Files/1000Sleep10Tasks.txt
        iii. **# 100 Sleep 1000ms(1 secs)**

        java -jar RunLocal.jar client -s LOCAL -t 16 -w ./Sleep1000Tasks\ Text\ Files/1600Sleep1000Tasks.txt
        java -jar RunLocal.jar client -s LOCAL -t 8 -w ./Sleep1000Tasks\ Text\ Files/800Sleep1000Tasks.txt
        java -jar RunLocal.jar client -s LOCAL -t 4 -w ./Sleep1000Tasks\ Text\ Files/400Sleep1000Tasks.txt
        java -jar RunLocal.jar client -s LOCAL -t 2 -w ./Sleep1000Tasks\ Text\ Files/200Sleep1000Tasks.txt
        java -jar RunLocal.jar client -s LOCAL -t 1 -w ./Sleep1000Tasks\ Text\ Files/100Sleep1000Tasks.txt
        iv. **# 10 Sleep 10000ms(10 secs)**

java -jar RunLocal.jar client -s LOCAL -t 16 -w ./Sleep10000Tasks\ Text\ Files/160Sleep10000Tasks.txt

java -jar RunLocal.jar client -s LOCAL -t 8 -w ./Sleep10000Tasks\ Text\ Files/80Sleep10000Tasks.txt

java -jar RunLocal.jar client -s LOCAL -t 4 -w ./Sleep10000Tasks\ Text\ Files/40Sleep10000Tasks.txt

java -jar RunLocal.jar client -s LOCAL -t 2 -w ./Sleep10000Tasks\ Text\ Files/20Sleep10000Tasks.txt

java -jar RunLocal.jar client -s LOCAL -t 1 -w ./Sleep10000Tasks\ Text\ Files/10Sleep10000Tasks.txt

v. **Time elapsed will be printed in the console.**

d. **Run the experiment REMOTELY:**

i. **Start 17 instances, one dedicated for client and 16 more for workers.**

ii. **Before running the Jar files, you need to have "credentials" file in your instance from where you are running the jar files.**

1. **For that you need to place the file to all the instances using SCP**
   scp -i ~/YOUR_PEM**.pem -r ./.aws** ec2-user@ec2-54-191-182-185.us-west-2.compute.amazonaws.com:~/

iii. **Place the RunClient.jar file in the instance dedicated for the client.**

iv. **Place the RunWorker.jar file in all the instances dedicated for worker.**

v. **And install pssh in your local using "sudo apt-get install pssh". I used Amazon Linux system as my local terminal. (So I did it using "sudo yum install pssh")**

vi. **Open up 2 terminals, one for Client execution and one to do PSSH.**

vii. **In client Execution, These below commands has to be given for each experiment execution.**

1. **#10K Sleep 0 Tasks**
   java -jar RunClient.jar client -s 10KSleep0 -w ./10KSleepTasks.txt
2. **#Sleep 10ms Tasks**
   java -jar RunClient.jar client -s 16KSleep10 -w ./Sleep10Tasks\ Text\ Files/16000Sleep10Tasks.txt
   java -jar RunClient.jar client -s 8KSleep10 -w ./Sleep10Tasks\ Text\ Files/8000Sleep10Tasks.txt
   java -jar RunClient.jar client -s 4KSleep10 -w ./Sleep10Tasks\ Text\ Files/4000Sleep10Tasks.txt
   java -jar RunClient.jar client -s 2KSleep10 -w ./Sleep10Tasks\ Text\ Files/2000Sleep10Tasks.txt
   java -jar RunClient.jar client -s 1KSleep10 -w ./Sleep10Tasks\ Text\ Files/1000Sleep10Tasks.txt
3. **#Sleep 1000ms (1 Secs) Tasks**
   java -jar RunClient.jar client -s 1600Sleep1s -w ./Sleep1000Tasks\ Text\ Files/1600Sleep1000Tasks.txt
   java -jar RunClient.jar client -s 800Sleep1s -w ./Sleep1000Tasks\ Text\ Files/800Sleep1000Tasks.txt
   java -jar RunClient.jar client -s 400Sleep1s -w ./Sleep1000Tasks\ Text\ Files/400Sleep1000Tasks.txt
   java -jar RunClient.jar client -s 200Sleep1s -w ./Sleep1000Tasks\ Text\ Files/200Sleep1000Tasks.txt
   java -jar RunClient.jar client -s 100Sleep1s -w ./Sleep1000Tasks\ Text\ Files/100Sleep1000Tasks.txt
4. **#Sleep 10000ms (10 secs) Tasks**

java -jar RunClient.jar client -s 160Sleep10s -w ./Sleep10000Tasks\ Text\ Files/160Sleep10000Tasks.txt
java -jar RunClient.jar client -s 80Sleep10s -w ./Sleep10000Tasks\ Text\ Files/80Sleep10000Tasks.txt
java -jar RunClient.jar client -s 40Sleep10s -w ./Sleep10000Tasks\ Text\ Files/40Sleep10000Tasks.txt
java -jar RunClient.jar client -s 20Sleep10s -w ./Sleep10000Tasks\ Text\ Files/20Sleep10000Tasks.txt
java -jar RunClient.jar client -s 10Sleep10s -w ./Sleep10000Tasks\ Text\ Files/10Sleep10000Tasks.txt

viii. **In worker Execution, These below commands has to be given for each experiment execution in different console to do PSSH.**

1. **#Sleep 0 Tasks**
   pssh -i -t 1000000 -h ipAddressOfWorkers.txt -l ec2-user -x "-oStrictHostKeyChecking=no -i ./CloudPA3.pem" 'java -jar RunWorker.jar worker -s 10KSleep0 -t 1'
2. **#Sleep 10ms Tasks**

   pssh -i -t 1000000 -h ipAddressOfWorkers16.txt -l ec2-user -x "-oStrictHostKeyChecking=no -i ./CloudPA3.pem" 'java -jar RunWorker.jar worker -s 16KSleep10 -t 16'
   pssh -i -t 1000000 -h ipAddressOfWorkers8.txt -l ec2-user -x "-oStrictHostKeyChecking=no -i ./CloudPA3.pem" 'java -jar RunWorker.jar worker -s 8KSleep10 -t 8'
   pssh -i -t 1000000 -h ipAddressOfWorkers4.txt -l ec2-user -x "-oStrictHostKeyChecking=no -i ./CloudPA3.pem" 'java -jar RunWorker.jar worker -s 4KSleep10 -t 4'
   pssh -i -t 1000000 -h ipAddressOfWorkers2.txt -l ec2-user -x "-oStrictHostKeyChecking=no -i ./CloudPA3.pem" 'java -jar RunWorker.jar worker -s 2KSleep10 -t 2'
   pssh -i -t 1000000 -h ipAddressOfWorkers1.txt -l ec2-user -x "-oStrictHostKeyChecking=no -i ./CloudPA3.pem" 'java -jar RunWorker.jar worker -s 1KSleep10 -t 1'
3. **#Sleep 1000ms Tasks**
   pssh -i -t 1000000 -h ipAddressOfWorkers16.txt -l ec2-user -x "-oStrictHostKeyChecking=no -i ./CloudPA3.pem" 'java -jar RunWorker.jar worker -s 1600Sleep1s -t 16'

   pssh -i -t 1000000 -h ipAddressOfWorkers8.txt -l ec2-user -x "-oStrictHostKeyChecking=no -i ./CloudPA3.pem" 'java -jar RunWorker.jar worker -s 800Sleep1s -t 8'

   pssh -i -t 1000000 -h ipAddressOfWorkers4.txt -l ec2-user -x "-oStrictHostKeyChecking=no -i ./CloudPA3.pem" 'java -jar RunWorker.jar worker -s 400Sleep1s -t 4'

   pssh -i -t 1000000 -h ipAddressOfWorkers2.txt -l ec2-user -x "-oStrictHostKeyChecking=no -i ./CloudPA3.pem" 'java -jar RunWorker.jar worker -s 200Sleep1s -t 2'

   pssh -i -t 1000000 -h ipAddressOfWorkers1.txt -l ec2-user -x "-oStrictHostKeyChecking=no -i ./CloudPA3.pem" 'java -jar RunWorker.jar worker -s 100Sleep1s -t 1'
4. **#Sleep 10000ms Tasks**

pssh -i -t 1000000 -h ipAddressOfWorkers16.txt -l ec2-user -x "-oStrictHostKeyChecking=no -i ./CloudPA3.pem" 'java -jar RunWorker.jar worker -s 160Sleep10s -t 16'

pssh -i -t 1000000 -h ipAddressOfWorkers8.txt -l ec2-user -x "-oStrictHostKeyChecking=no -i ./CloudPA3.pem" 'java -jar RunWorker.jar worker -s 80Sleep10s -t 8'

pssh -i -t 1000000 -h ipAddressOfWorkers4.txt -l ec2-user -x "-oStrictHostKeyChecking=no -i ./CloudPA3.pem" 'java -jar RunWorker.jar worker -s 40Sleep10s -t 4'

pssh -i -t 1000000 -h ipAddressOfWorkers2.txt -l ec2-user -x "-oStrictHostKeyChecking=no -i ./CloudPA3.pem" 'java -jar RunWorker.jar worker -s 20Sleep10s -t 2'

pssh -i -t 1000000 -h ipAddressOfWorkers1.txt -l ec2-user -x "-oStrictHostKeyChecking=no -i ./CloudPA3.pem" 'java -jar RunWorker.jar worker -s 10Sleep10s -t 1'

ix. **Please check the client terminal for every time elapsed**

e. **Animoto**

Please follow the same steps as of remote execution.
One thing to add is **ImagesURLs.txt** and **IPAddrList1.txt, IPAddrList2.txt, IPAddrList4.txt, IPAddrList8.txt, IPAddrList16.txt** file to the client instance.
And add **AnimotoClient.jar** to client instance.
ssh -i ~/CloudPA3.pem ubuntu@ec2-54-187-35-221.us-west-2.compute.amazonaws.com
scp -i ~/CloudPA3.pem ./AnimotoClient.jar ubuntu@ec2-54-187-35-221.us-west-2.compute.amazonaws.com:~/

scp -i ~/CloudPA3.pem -r ./IPAddrList* ubuntu@ec2-54-187-35-221.us-west-2.compute.amazonaws.com:~/
scp -i ~/CloudPA3.pem ./AnimotoClient.jar ubuntu@ec2-54-187-35-221.us-west-2.compute.amazonaws.com:~/
scp -i ~/CloudPA3.pem ./ImagesURLs.txt ubuntu@ec2-54-187-35-221.us-west-2.compute.amazonaws.com:~/

Add AnimotoWorker.jar to worker instances.

ssh -i ~/CloudPA3.pem ubuntu@ec2-54-186-245-234.us-west-2.compute.amazonaws.com
scp -i ~/CloudPA3.pem ./AnimotoWorker.jar ubuntu@ec2-54-186-245-234.us-west-2.compute.amazonaws.com:~/

Install pssh

sudo apt-get install pssh
or
// To install PSSH
sudo apt-get install pssh

If installed properly it shows when typed pssh:

Usage: pssh [OPTIONS] command [...]
pssh: error: Command not specified.

If the above output is not seen, then You need to create a Symlink to the installed path:
sudo ln -s /usr/bin/parallel-ssh /usr/local/bin/pssh
Then again type :pssh on terminal.

pssh -i –t 1000000 -l ubuntu -h IPAddrList.txt -x "-oStrictHostKeyChecking=no -i <YOUR_KEY>.pem" uptime.

Install ffmpeg
sudo add-apt-repository ppa:mc3man/trusty-media
sudo apt-get update
sudo apt-get dist-upgrade
sudo apt-get install ffmpeg
And the take the image and create rest of the workers.

TO execute, use the follow commands
Client terminal:

```
java -jar AnimotoClient.jar client -s animato -w ./ImagesURLs.txt 1

java -jar AnimotoClient.jar client -s animato -w ./ImagesURLs.txt 2
java -jar AnimotoClient.jar client -s animato -w ./ImagesURLs.txt 4

java -jar AnimotoClient.jar client -s animato -w ./ImagesURLs.txt 8

java -jar AnimotoClient.jar client -s animato -w ./ImagesURLs.txt 16
```

Duplicate client terminal:

```
pssh -i -t 1000000 -h IPAddrList1.txt -l ubuntu -x "-
oStrictHostKeyChecking=no  -i ./CloudPA3.pem" 'java -jar AnimotoWorker.jar
worker -s animato -t 1'
pssh -i -t 1000000 -h IPAddrList2.txt -l ubuntu -x "-
oStrictHostKeyChecking=no  -i ./CloudPA3.pem" 'java -jar AnimotoWorker.jar
worker -s animato -t 2'
pssh -i -t 1000000 -h IPAddrList4.txt -l ubuntu -x "-
oStrictHostKeyChecking=no  -i ./CloudPA3.pem" 'java -jar AnimotoWorker.jar
worker -s animato -t 4'
pssh -i -t 1000000 -h IPAddrList8.txt -l ubuntu -x "-
oStrictHostKeyChecking=no  -i ./CloudPA3.pem" 'java -jar AnimotoWorker.jar
worker -s animato -t 8'
pssh -i -t 1000000 -h IPAddrList16.txt -l ubuntu -x "-
oStrictHostKeyChecking=no  -i ./CloudPA3.pem" 'java -jar AnimotoWorker.jar
worker -s animato -t 16'
```

Design Document

CloudKon clone with Amazon EC2, S3, SQS, and DynamoDB

Pavankumar Shetty
CWID-A20354961
CS-553 Spring 2016
Main Campus

Table of Contents

Introduction

This project aims at gaining experience with

- Amazon Web Services, specifically the EC2 cloud (http://aws.amazon.com/ec2/)
    - EC2 **t2-micro** instances are used to run the different set of experiments. Two components were used with instances, client (to submits tasks to SQS) and workers (to retrieve tasks from SQS and execute them)
- SQS queuing service (http://aws.amazon.com/sqs/)
    - The SQS service is used to handle the queue of requests to load balance across multiple workers.
- S3 (http://aws.amazon.com/s3/)
    - Used to store video as part of Animato Extra Credit part.
- DynamoDB (http://aws.amazon.com/dynamodb/)
    - This is used in the experiment to avoid duplicates of tasks being executed at worker end (i.e., on SQS)

and distribute load balancing between clients and workers.

Overview and Key points:

- Two sets of experiments are done.
- One running the different set of experiments locally (ConcurrentLinkedQueue implementation) and one on remote (i.e., on t2-micro instances and SQS)
- The ratio between the time taken to conduct the experiment in ideal condition and the time taken with my implementation is termed as **EFFICIENCY**. Usually measure in percentage.
- The ratio between no of tasks and total time taken to execute (measured in secs). No of tasks executed per seconds as **THROUGHPUT.** In this experiment it is measured as task per second.

Language & Platform

Language used: JAVA programming, which covers Multithreading.

Platform: Built on Amazon EC2 Linux environment.
Instance type used: t2. micro

Design Principle

1. LOCAL

1. **Class name:** `MessagePassingThreadPool` which is the main flow of the system which creates **ConcurrentLinkedQueue**<TaskToQueueThreadPool> requestQueue and **ConcurrentLinkedQueue**<TaskToQueueThreadPool> responseQueue are used as a replacement to SQS on local.

```
ExecutorService executorConsumer = Executors.newFixedThreadPool(noOfThreads);//cre
for(int i=0;i<noOfThreads;i++) {
    Runnable workerConsumer = new ConsumerThreadPool(requestQueue,responseQueue);
    executorConsumer.execute(workerConsumer);
}
```

Created a thread pool for workers and executed tasks as per the requirement using **ExecutorService**.

2. **Class name:** `TaskToQueueThreadPool`, which is the task bean class, which has taskID, status (Processed or not by workers) and message as three parameters.
3. **Class name:** `PublisherThreadPool`, which implements the client run part of Client. Reading the file and making it as tasks and putting it in requestQueue by setting status as false(Means unprocessed).
4. **Class name:** `ConsumerThreadPool`, which implements the run part of every worker. Reading from the requestQueue and write it to responseQueue, after executing the task by changing the status as true.

2. REMOTE

1. **Class name:** `RemoteClientSQS`, which implements the client run part of rremote Client. Reading the file and making it as tasks and putting it in requestQueue by setting status as false(Means unprocessed). **ConcurrentHashMap<Integer, String> mySubmittedTasks** has been used to keep track of sent messages.

AmazonDynamoDBClient dynamoDB and AmazonSQS sqs are used as a AWSClients for both **dynamodb and sqs.**

```
// Create a queue
//System.out.println("Accessing SQS queue: "+requestQueueName);
String myRequestQueueUrl = sqs.createQueue(requestQueueName).getQueueUrl();

//System.out.println("Creating a new SQS queue called MyResponseQueue.\n");
String myResponseQueueUrl = sqs.createQueue(responseQueueName).getQueueUrl();
```

Two SQS queues are created as request and response queue.

```
fileReader = new FileReader(fileName);
bufferedReader = new BufferedReader(fileReader);
String eachTaskLine = null;
mySubmittedTasks = new ConcurrentHashMap<Integer, String>();

while ((eachTaskLine = bufferedReader.readLine()) != null) {
    sqs.sendMessage(new SendMessageRequest(myRequestQueueUrl, taskID + " " + eachTaskLine));
    mySubmittedTasks.put(taskID, eachTaskLine);
    taskID++;
}
```

Messages are sent to requestQueue created from reading the file and the same task is put in Hashmap to verify the every task has been executed or no.

```
while (!mySubmittedTasks.isEmpty()) {
    List<Message> messages = sqs.receiveMessage(receiveMessageRequest).getMessages();
    for (Message message : messages) {
        if (mySubmittedTasks.containsKey(Integer.parseInt(message.getBody().toString()))) {
            mySubmittedTasks.remove(Integer.parseInt(message.getBody().toString()));
            String messageReceiptHandle = message.getReceiptHandle();
            sqs.deleteMessage(new DeleteMessageRequest(myResponseQueueUrl, messageReceiptHandle));
        }
    }
}
```

Once the messages are retrieved back in response queue it is again verified with the task sent. And the time taken to perform the same is obtained.

Executable name: RunClient.jar

2. **Class name:** `RemoteWorkerSQS`, which implements the worker part of Remote experiment. It reads every tasks from request queue and puts it into DYNAMODB and executes it.

```
PutItemRequest putItemRequest = new PutItemRequest(tableName, item)
        .withConditionExpression("attribute not exists(taskID)");
```

Duplicacy check has been done for task execution using **DYNAMODB.** Used conditionexpression, which throws `ConditionalCheckFailedException` when duplicacy is found while inserting.

```
ReceiveMessageRequest receiveMessageRequest = new ReceiveMessageRequest(myRequestQueueUrl);
```

And then messages are received

```
List<Message> messages = sqs.receiveMessage(receiveMessageRequest).getMessages();
```

And after executing the sleep tasks, tasks are deleted from the request queue and taskID is put into response queue.

```
sqs.sendMessage(
        new SendMessageRequest(myResponseQueueUrl,splitTask[0]));
// Delete the message
String messageReceiptHandle = message.getReceiptHandle();
sqs.deleteMessage(new DeleteMessageRequest(myRequestQueueUrl, messageReceiptHandle));
```

Executable name: RunWorker.jar

3. ANIMOTO implementation

1. **Class name:** `AnimatoClient`, Same implementation as REMOTE CLIENT part. Instead of tasks, URLs of the images are given.
   URL used: http://wallpapercave.com/wp/Qsbbfie.png
   And used Google URL shortner and used : http://goo.gl/uoBq8r
   For every workers, urls are found and listed here.

```
for(String url:urlToViewVideo){
    if(!url.equalsIgnoreCase("URLs")){
        System.out.println("Location of the video on S3: " + url);
    }
}
```

2. **Class name:** `AnimatoWorker`, Same implementation as REMOTE SERVER part. Instead of sleep tasks, URLs of the images are downloaded using **wget**. And all those images are clubbed together and a video is made using **ffmpeg. Ffmpeg** should be installed in your instance.

```
url = "wget -O /home/ubuntu/pics/pshetty4pic"+i+".png "+urlList[i];
//System.out.println("URL each worker::>>"+url);
fetchImageFromURL = Runtime.getRuntime().exec(url);
fetchImageFromURL.waitFor();
```

```
Process downloadVideo = Runtime.getRuntime().exec(
        "ffmpeg -t 60 -r 1/5 -i /home/ubuntu/pics/pshetty4p
downloadVideo.waitFor();
```

And after these, using AmazonS3Client, buckets are created with key for file.

```
String bucketName = "animoto75f2b1df-d46f-40d2-bad9-40b326d5027a";
String key = "MyVideoFile" + UUID.randomUUID();
```

And once the task is executed, video is uploaded to s3.

```
s3.putObject(new PutObjectRequest(bucketName, key, fileToBeSent));
// System.err.println("after creating and putting it");

String urlOfS3 = "http://" + bucketName + ".s3-website-us-west-2.amazonaws.com/" + key;
```

And once it is put, the url is sent back to client side.

```
sqs.sendMessage(new SendMessageRequest(myResponseQueueUrl, ("17000" + "~" + urlOfS3)));
```

Performance Evaluation

CloudKon clone with Amazon EC2, S3, SQS, and DynamoDB

Pavankumar Shetty
CWID-A20354961
CS-553 Spring 2016
Main Campus

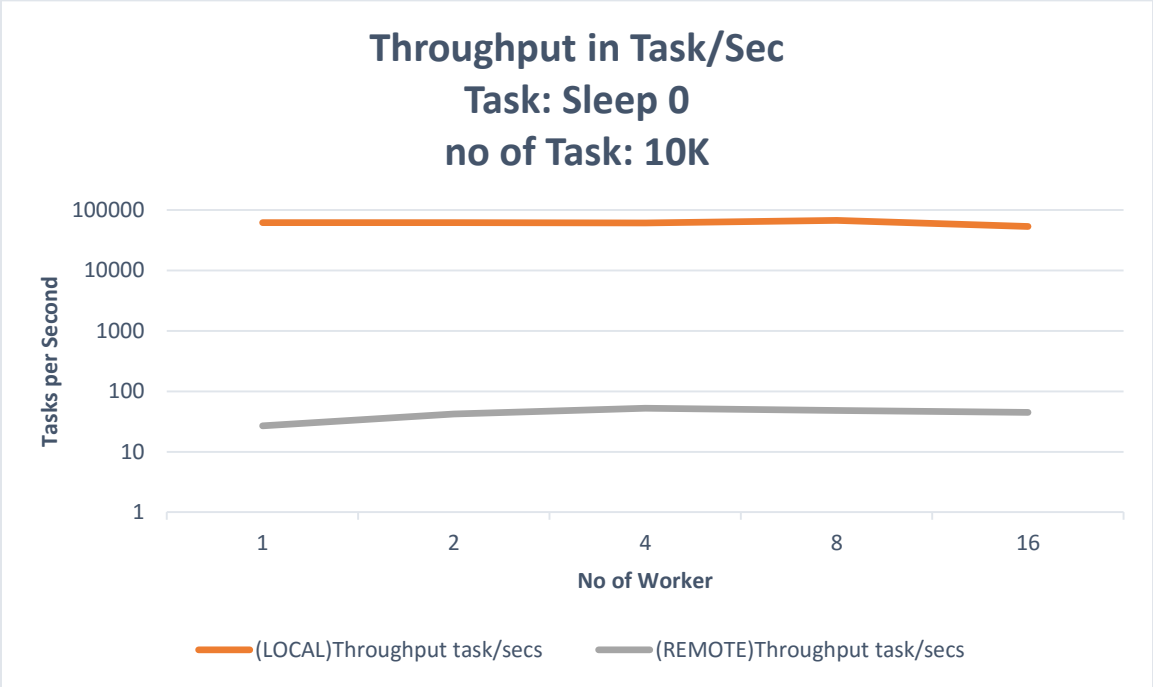
Experiment no 1.

Time taken:

No of threads	(LOCAL)Time in secs	(REMOTE)Time in secs
1	0.1619	372.33
2	0.1621	237.69
4	0.1634	189.36
8	0.1489	206.49
16	0.1887	223.886

Throughput:

No of threads	(LOCAL)Throughput task/secs	(REMOTE)Throughput task/secs
1	**61766.52254**	**26.85789488**
2	**61690.31462**	**42.07160587**
4	**61199.5104**	**52.80946346**
8	**67159.16723**	**48.42849533**
16	**52994.17064**	**44.66558874**



Conclusion:

Local queue seems to be very high efficient because there won't be any communication overhead as in case of remote execution. Throughput is increasing gradually as more number of processors are added to the execution. As it is t2 micro instances, we cannot guarantee the performance every time.

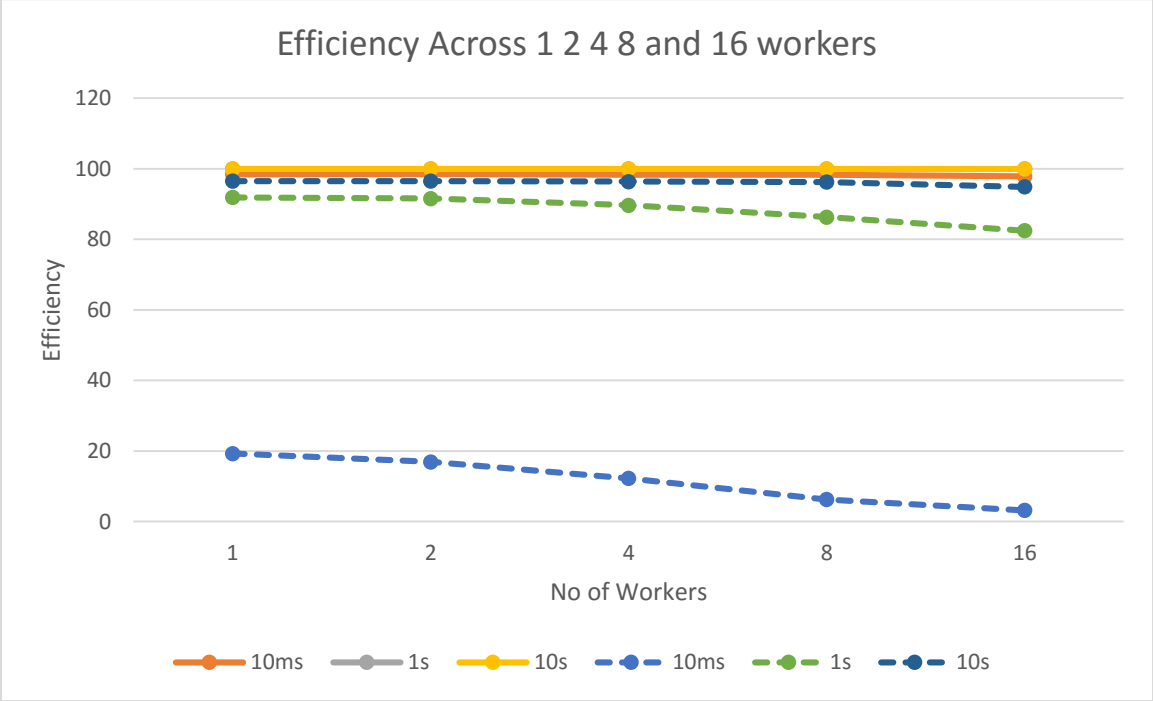
Experiment no 2.

Time taken:

	Time ideally			Time locally			Time Remotely		
No of threads	10ms	1s	10s	10ms	1s	10s	10ms	1s	10s
1	10	100	100	10.16	100.039	100.0213	51.85	108.83	103.63
2	10	100	100	10.15	100.042	100.0284	59.13	109.23	103.62
4	10	100	100	10.17	100.038	100.0242	81.75	111.52	103.76
8	10	100	100	10.17	100.038	100.0232	160.44	115.88	103.9
16	10	100	100	10.22	100.041	100.0265	316.77	121.31	105.39

Efficiency:

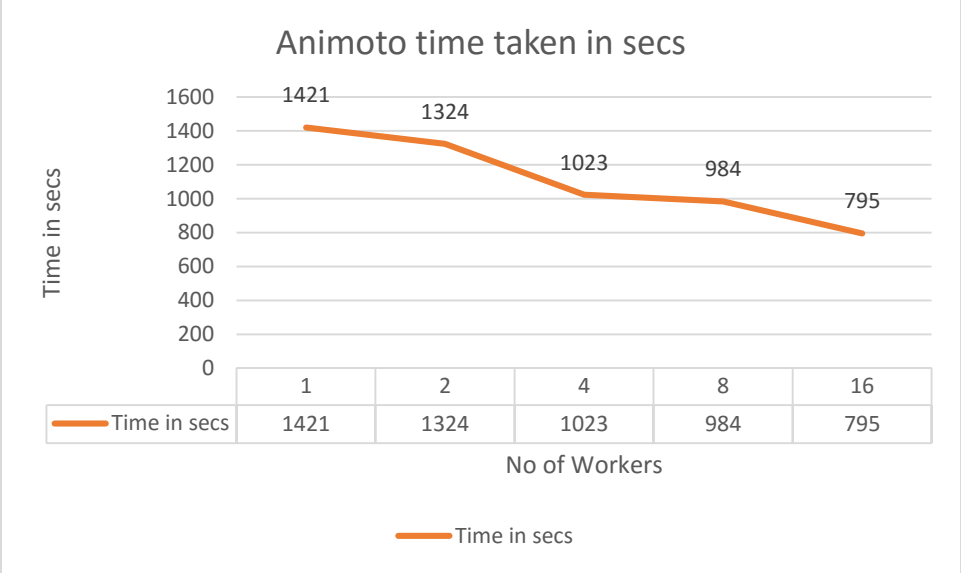
	Efficiency LOCAL			Efficiency REMOTE		
No of threads	**10ms**	**1s**	**10s**	**10ms**	**1s**	**10s**
1	98.4252	99.9610152	99.9787	19.2864	91.88642837	96.49715
2	98.52217	99.95801763	99.97161	16.91189	91.54994049	96.50647
4	98.32842	99.96201443	99.97581	12.23242	89.67001435	96.37625
8	98.32842	99.96201443	99.97681	6.23286	86.29616845	96.24639
16	97.84736	99.9590168	99.97351	3.156865	82.433435	94.88566



Conclusion:

From the above graph, it can be inferred that local queue has almost 100% efficiency as it has zero communication amongst them. But, remote instances prove to be less efficient, as it has communication overhead on it. As more number of workers, the efficiency decreases.

Plot of time taken for Animoto Clone for 1 2 4 8 16 Workers:



Conclusion:

And it takes more time for convert into VIDEO. Uploading to S3 is quick and persistent.

Implemented a Web application and static website hosted has been captured and screenshot is provided.
And as suggested by TAs, Screen frames of the video has been captured and the same has been submitted.

Screenshots are attached separately in a folder named Screenshot for every execution.

REFERENCES:

- AWS SQS, DynamoDB and S3 sample projects from AWS SDK installed in Eclipse.
- http://www.javatpoint.com/java-thread-pool
- http://docs.aws.amazon.com/AmazonS3/latest/dev/HostingWebsiteOnS3Setup.html
- http://docs.aws.amazon.com/AmazonS3/latest/dev/WebsiteAccessPermissionsReqd.html
- http://docs.aws.amazon.com/AmazonS3/latest/dev/UsingTheMPDotJavaAPI.html#TestingJavaSamples
- https://goo.gl/
- http://wallpapercave.com/wp/Qsbbfie.png
- http://stackoverflow.com/questions/6896490/how-to-set-a-videos-duration-in-ffmpeg
- http://www.ffmpeg.org/ffmpeg.html