CS553: Cloud Computing

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

VICKYBEN PATEL (CWID: A20370450)

The goal of this programming assignment is to enable to gain experience programming with:
● Amazon Web Services, specifically the EC2 cloud , the SQS queuing service , S3 and DynamoDB.
●Learn about distributed load balancing and how to distribute work between clients and workers
This assignment involved implementing a distributed task execution framework on Amazon EC2 using the SQS.

Design:

The Client:

The client is command line tool, which can submit tasks to the SQS.
Client has following roles:
1. The client reads the file from the local and sends task to the server one by one.
2. Following commend is used to run the client:
java -cp <<JarFIle>> Client <<Queue name>> <<Filename>>
Where,
Queue name- Name of the queue
File Name- Name of the file in which all the workloads are specified.

Local Back-End Workers:

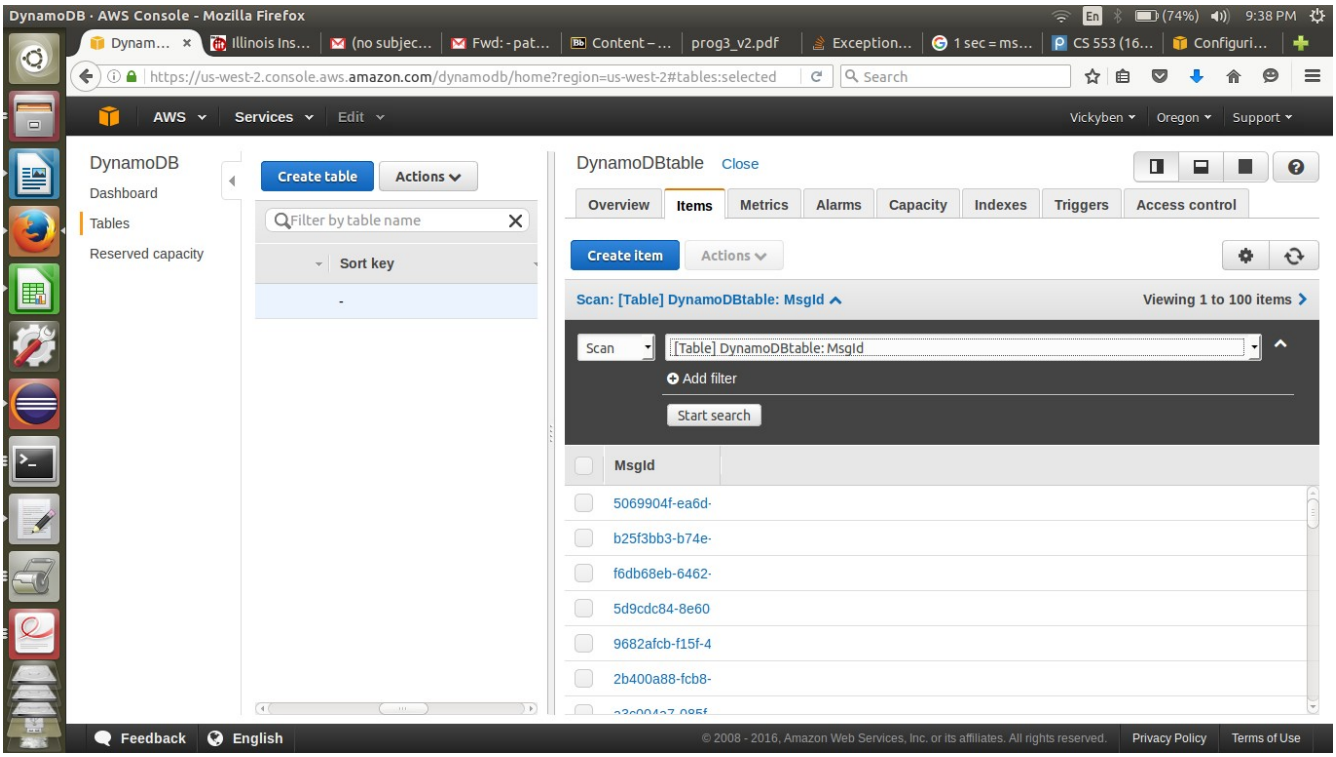
In this we implemented a pool of threads that will process tasks from the request queue, and when complete will put results on the response queue.
The functionality as below:
1. The numbers of local workers are defined by the number of threads.
2. The results are written in another queue as soon as the thread completes running the tasks.
3. This method handle the sleep functionality by taking tasks sleeps for a specified length in milliseconds.
4. An exception is raised on the failure of the thread and the return value is stored in the result queue.

Remote Back-End Workers :

In this some back end workers are implemented that run on different machines to facilitate the scalability.
The functionality is as follows:
1. The tasks are stored into SQS queue on AWS.
2. The remote worker polls the SQS queue. If it finds any tasks, they are retrieved and executed. Otherwise it polls again after some time.
3. If the poll remains idle for too long, the remote worker is terminated.

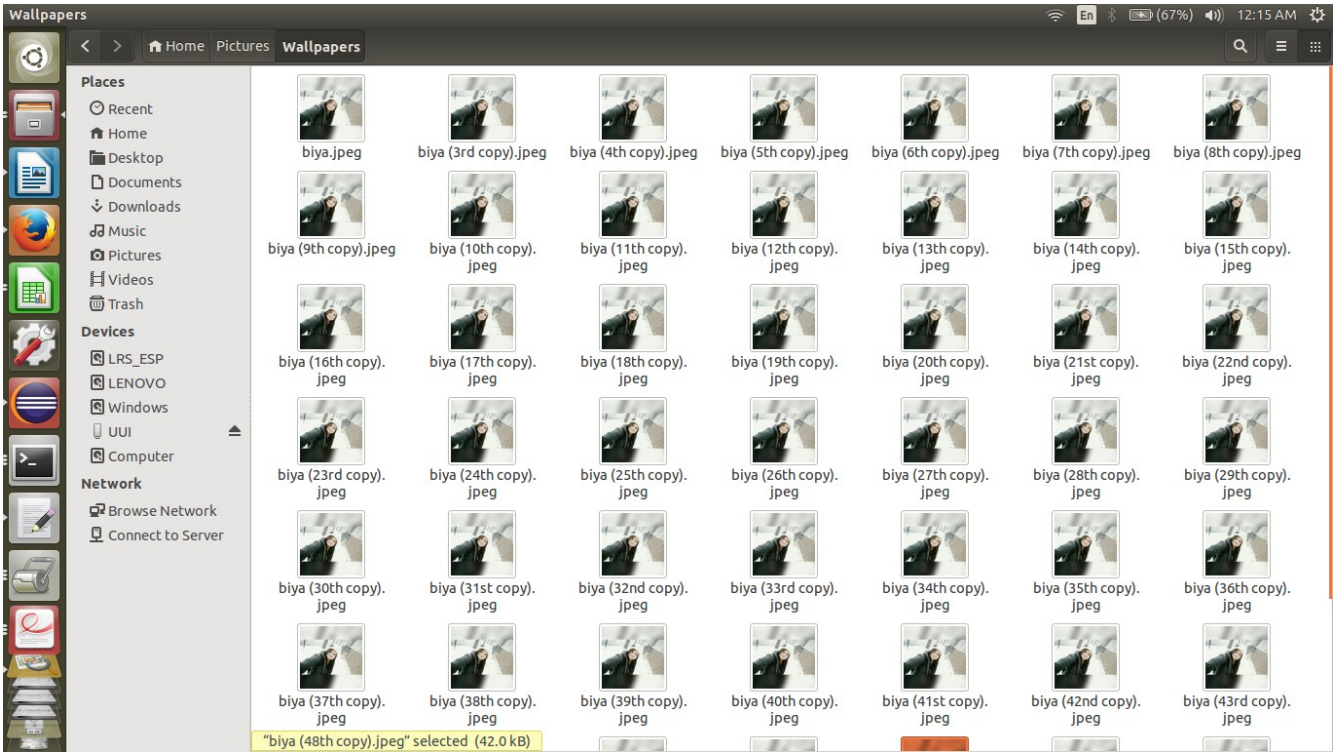
Duplicate Tasks

Here we are using DynamoDB. Each worker checks DynamoDB first if any worker working on same task. If MsgID not present in DynamoDB, that means no other worker working on that task.In that case add entry of MsgID and execute the task.



Animoto

For the animoto clone I had to use the FFMEPG where you needed to stitch a list of photos to form video. So instead of finding 60 HD photos I would convert a HD video to a list of photos and put them to drop box account and make them accessible publicly . These set of links will be sent by the client for the S3 to download using wget and then use ffmpeg to stitch the photos to for the videos. The video would be made shareable to the public and link would be sent back to the user



Code Design:

The detailed description of source code files as follows:

Local Back End Workers

Client.java : Create 2 queues Reqest queue and Response queue, add tasks to the request queue, send task one by one to threadpool(WorkerThread.java)
WorkerThread.java : Perform task and add to response queue

Remote Back-End Workers

Client.java : Create 2 SQS queues Request queue and Response queue, create DynamoDB table, send tasks to queue
Worker.java : Receive tasks from Request queue check the message Id of task in DynamoDB table if not present then perform task and add task message Id to DynamoDB table then add task in Response queue and delete from Request queue.

Manual:

Local Back End Workers

LocalClientWorker.jar

java -cp LocalClientWorker.jar Client local 16 /home/ec2-user/task10s16t.txt

local : local worker version
16 : the number of threads in the thread pool
task10s16t.txt : Workload file

Remote BackEnd Workers

Remote.jar

$aws configure

[ec2-user@ip-172-31-11-199 ~]$ aws configure
AWS Access Key ID [None]: AKIAIKRB5CM2Y5XOPHWA
AWS Secret Access Key [None]: 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
Default region name [None]: US_WEST_2
Default output format [None]:

One instance is created for Client and run using the following command

java -cp Remote.jar Client ReqQueue /home/ec2-user/task.txt

ReqQueue : Create queue of this name
task.txt : Workload file

All other instances are made remote workers and run using following command

#pssh in to 16 instances and the host ip's are configured in the hosts.txt
For 1 instance:
pssh -i -h host.txt -l ubuntu -x "-oStrictHostKeyChecking=no -i /home/biya/pa3.pem" 'java -cp Remote.jar worker –s ReqQueue –t 1'
For 2 instance:
pssh -i -h host.txt -l ubuntu -x "-oStrictHostKeyChecking=no -i /home/biya/pa3.pem" 'java -cp Remote.jar worker –s ReqQueue –t 2
For 4 instance:
pssh -i -h host.txt -l ubuntu -x "-oStrictHostKeyChecking=no -i /home/biya/pa3.pem" 'java -cp Remote.jar worker –s ReqQueue –t 4'
For 8 instance:
pssh -i -h host.txt -l ubuntu -x "-oStrictHostKeyChecking=no -i /home/biya/pa3.pem" 'java -cp Remote.jar worker –s ReqQueue –t 8'
For 16 instance:
pssh -i -h host.txt -l ubuntu -x "-oStrictHostKeyChecking=no -i /home/biya/pa3.pem" 'java -cp

Remote.jar worker –s ReqQueue –t 16"

ReqQueue : Fetch tasks from this queue
1,2,4,8,16 : No. of workers

ANIMOTO

The ffmpeg decoder is installed.
The following command takes the images from the local disk and converts it into the video.
Ffmpeg –framerate 1/1 –i Images/Img%0d.jpg –c:v libx264 r 25 –pix_fmt yuv420p imagevideo.mp4

Performance:

Throughput :

Execution time table :

No. of workers	Local (10k)	Local (100k)	Remote
1	969	15020	2398856.54
2	875	12714	341288.47
4	626	12152	128725.65
8	641	12382	65899.23
16	674	13139	70478.12

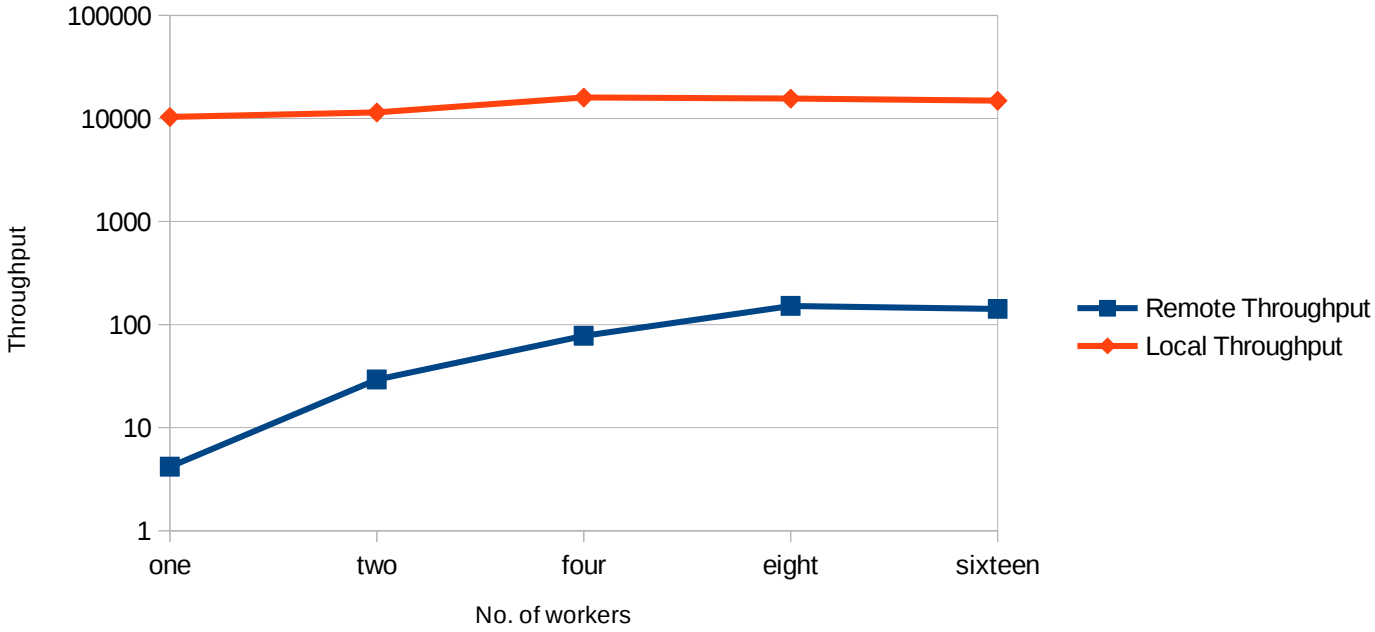
Throughput (tasks/sec) table :

The throughput is determined by dividing the number of tasks processed by the total time from submission of the first task to the completion of the last task. Below table demonstrates the throughput values of 10K tasks and then tasks are increased to 100K since the time taken for the execution of 10k sleep job is less than 10 sec.

Throughput = $\frac{\text{Number of tasks}}{\text{total time}}$

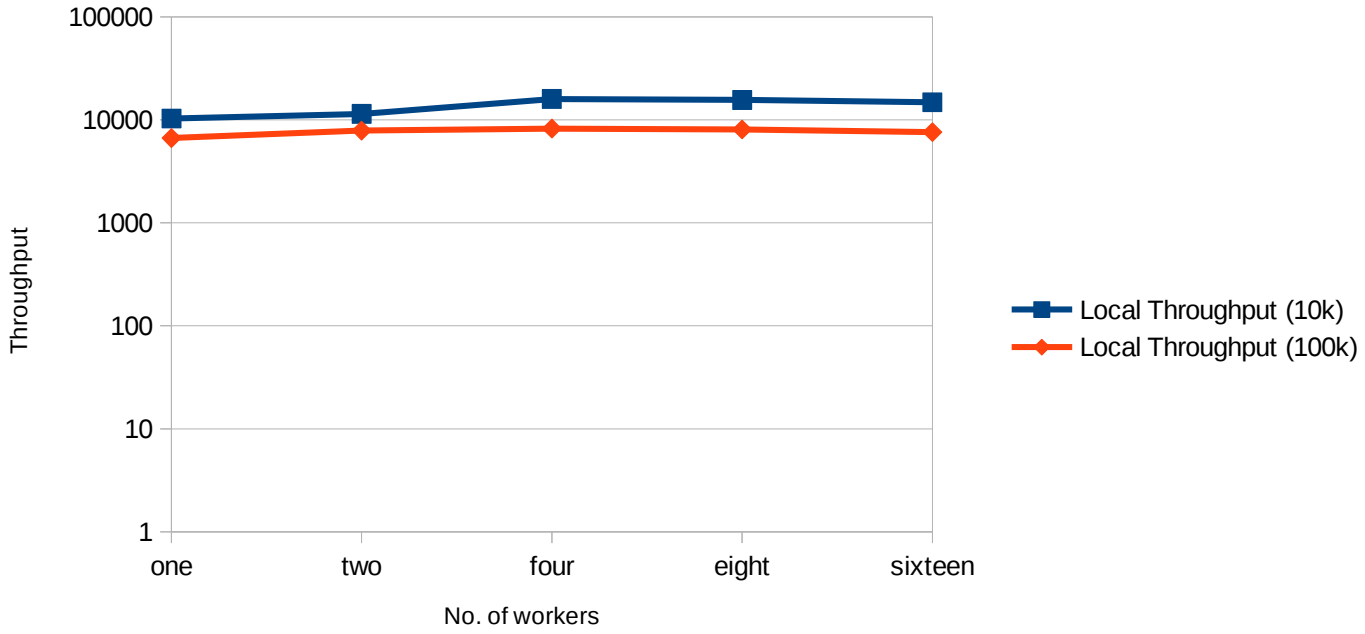
No. of workers	Local Throughput (10k)	Local Throughput (100k)	Remote Throughput (10k)
1	10319.9174	6657.789	4.1686527
2	11428.5714	7865.345	29.30072615
4	15974.44089	8229.098	77.8845951
8	15600.624	8076.223	151.7468413
16	14836.7952	7610.929	141.8880072

Throughput comparison graph for remote and local:



Increase in throughput as no. of workers increase. Throughput of local is high compare to remote. Increase in throughput of Local from 1 worker to 16 workers is very small and increase in throughput of remote from 1 worker to 16 workers is high compare to Local.

Local throughput graph for 10k and 100k tasks:



Efficiency :

Execution time table :

no.of worker	Local (10ms)	Local (1s)	Local (10s)	Remote (10ms)	Remote (1s)	Remote (10s)
1	10135	100038	100011	419284.47	212412.23	196812.23
2	10154	100035	100010	109145.54	177523.12	153985.59
4	10165	100033	100013	77458.12	147212.45	138099.41
8	10217	100036	100012	71845.25	135420.85	127641.21
16	10164	100039	100016	72214.56	127053.48	127586.58

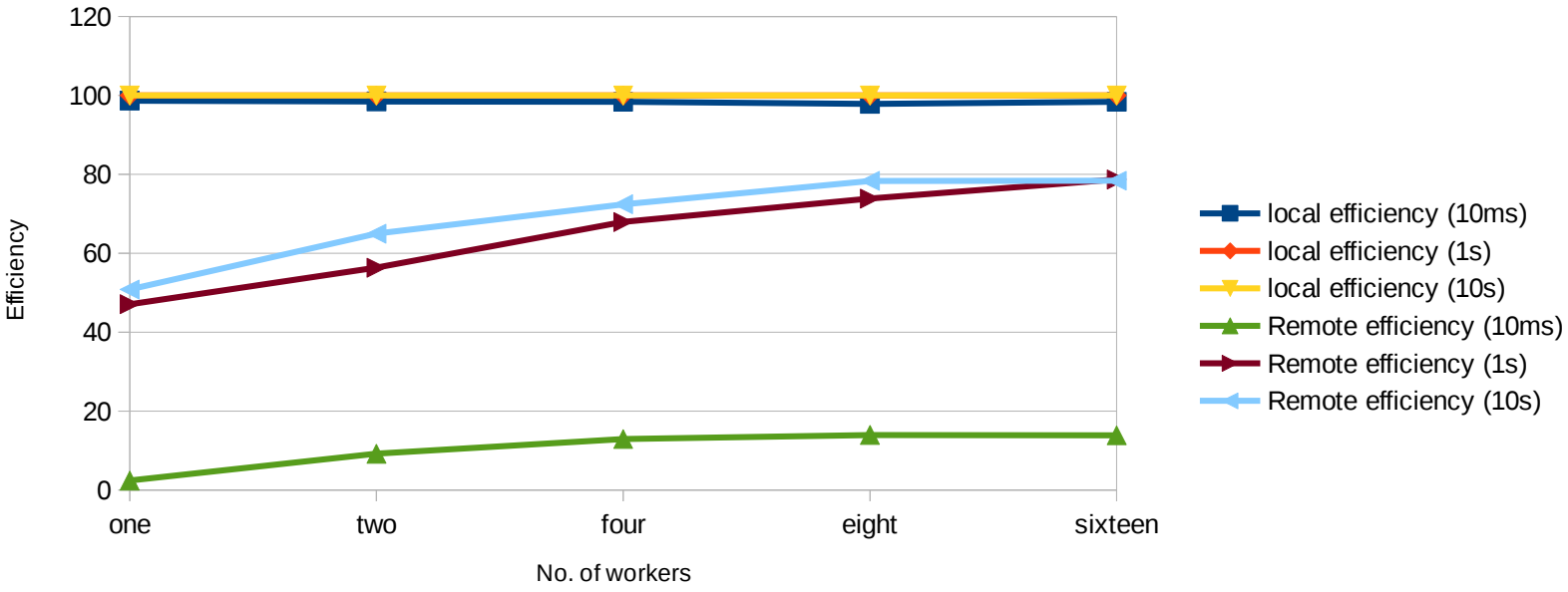
Efficiency table :

efficiency is the ideal time divided by the measured time.

Efficiency = $\frac{\text{Ideal time}}{\text{measured time}}$

No. of workers	local efficiency (10ms)	local efficiency (1s)	local efficiency (10s)	Remote efficiency (10ms)	Remote efficiency (1s)	Remote efficiency (10s)
one	98.668	99.962	99.989	2.385	47.0782	50.8099
two	98.483	99.965	99.99	9.162	56.3307	64.9411
four	98.377	99.967	99.987	12.9102	67.929	72.4116
eight	97.876	99.964	99.988	13.9188	73.8439	78.3446
sixteen	98.386	99.961	99.984	13.8476	78.707	78.3781

Efficiency comparison graph for remote and local :



Local efficiency for 10ms sleep time, 1s sleep time and 10s sleep time is near by 100.
and for remote as sleep time increase efficiency is also increases and also as workers increase efficiency increase.

References:

http://www.journaldev.com/1069/java-thread-pool-example-using-executors-and-threadpoolexecutor

http://www.javacodex.com/Concurrency/ConcurrentLinkedQueue-Example