CS553: Cloud Computing
Report
Programming Assignment-3

Problem:

The goal of this programming assignment is to get to know the Amazon Web Services like EC2, SQS queuing service, S3 and Dynamo DB. In this assignment will learn about distributed load balancing and how to distribute the loads/tasks between client and worker.

Overall program design

This assignment has 2 parts

1. Local task execution framework (In-memory)
2. Remote task execution framework (Using SQS)

1. Local task execution framework
    - In this part of assignment, we have put the client and worker on same machine and then they will communicate via in-memory. We considered this design tradeoff to avoid latency and request response time between
    - We have created one client who will internally give calls to the number of workers threads depending on the inputs provided via command line interface.
    - Client will read the file called workload, this workload file act as input of tasks.
    - Then client part will create an in-memory queue to store the jobs/tasks read from workload file.
    - At the same time client will gives call/create to worker threads depending upon input commands.
    - Worker threads will read the source queue and will execute the task, as this is in-memory queue, reading will be fast and synchronized (as we are dequeening) so there are no chances of submitting same task to multiple thread.
    - So here don't have to do any validation for duplicate execution of same task.
    - Once task is completed by the working thread they will report the result to client via another in-memory queue.
    - When all task get completed client will get terminated.

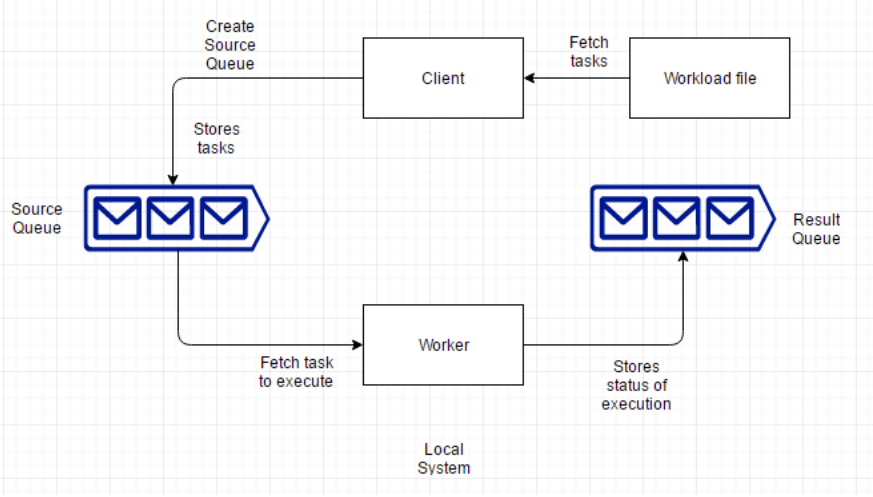


Fig. Execution flow for Local task execution framework

2. Remote task execution framework
    - In this part of assignment, we have to put the client and worker on different machine and then they will communicate via SQS.
    - On worker side, worker will create source and result queue, and will wait for client to submit the job to source queue.
    - Once client submit the job worker will fetch them from source queue (which is SQS in this case) and will process them
    - Here on worker side, we have to handle the one drawback of SQS.
    - SQS is guarantees to give the response back but there is not guarantee that same task will not send twice. (SQS sends same task multiple times- i.e. Duplicate Tasks)
    - So to handle this we have used Dynamo DB, in Dynamo Db we have inserted the task ID of each task processed by worker.
    - Each time a new task received by any worker first thing is to check whether that task has been processed by any other worker or not. If it is already picked up by other worker, worker simply moves forward and request for next task.
    - Once processing done by remote worker, they put their feedback (status of task) 0 incase successful and 1 in case of unsuccessful in result SQS queue.
    - On client side end user has to provide the workload file which client will read and start pushing job to Source SQS queue.
    - When client submits the task at same time client start checking for result SQS queue for results. And once all task gets processed client get shutdown.

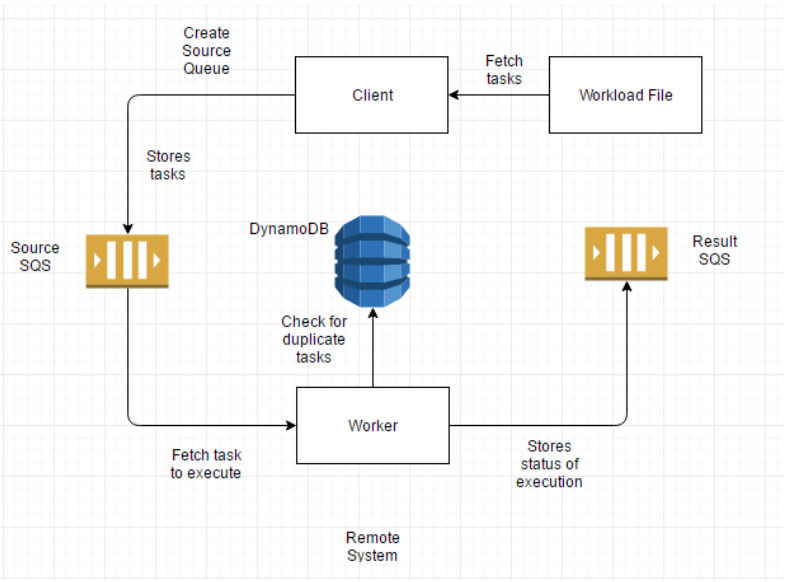


Fig. Execution flow for Remote task execution framework

Manual

Local Worker Experiment

1. Added .pem file to identity first
   cd /home/abhijeet/workspace
   chmod 400 abhijeetkedari.pem
   ssh-add abhijeetkedari.pem

2. Then copy all required Programming codes and scripts to Amazon EC2 instance by SCP commands
   scp -i abhijeetkedari.pem -r /home/abhijeet/workspace/LocalWorkerCloud ubuntu@52.23.166.253:/home/ubuntu
   scp -i abhijeetkedari.pem -r /home/abhijeet/workspace/Utility ubuntu@52.23.166.253:/home/ubuntu
   scp -i abhijeetkedari.pem /home/abhijeet/workspace/localworkerscript.sh ubuntu@52.23.166.253:/home/ubuntu
3. Then connect to Amazon EC2 instance by SSH command
   ssh ubuntu@54.164.214.73
4. On Amazon EC2 instance execute the script **./localworkerscript.sh**

Script **localworkerscript.sh** contains

- Update required software
- Run the Local experiment for all cases sequentially, like for 0ms, 10ms, 1 second, 10 seconds and for different number of threads like 1,2,4,8 and 16.

Remote Worker Experiment

For Remote Client part

1. Added .pem file to identity first
   cd /home/abhijeet/workspace
   chmod 400 abhijeetkedari.pem
   ssh-add abhijeetkedari.pem
2. Then copy all required Programming codes and scripts to Amazon EC2 instance by SCP commands
   scp -i abhijeetkedari.pem -r /home/abhijeet/workspace/Utility ubuntu@54.164.214.73:/home/ubuntu
   scp -i abhijeetkedari.pem -r /home/abhijeet/workspace/CloudKonClient
   ubuntu@54.164.214.73:/home/ubuntu
3. Generate workload first
   A. Update required software
      sudo apt-get update
      sudo apt-get install oracle-java8-installer
      sudo apt-get install ssh
      sudo apt-get install ant

   B. Go to the directory /home/ubuntu/Utility
   C. Use build.xml to build the code and generate the jar file by command called **ANT**
   D. Use below command to generate required workload as per experiment
      java -cp /home/ubuntu/Utility/build/jar/Utility.jar TaskCreation 10000 0
4. Go to the /home/ubuntu/CloudKonClient, where your local client code is saved
5. Generate the jar file by using below commands
   cd /home/ubuntu/CloudKonClient
   ant
6. Wait for Remote worker to get setup and ready state, once they are in ready state run below command
   java -cp /home/ubuntu/CloudKonClient/build/jar/ClientSide.jar SQSClient client -s SOURCEQUEUE -w workload

   This command will run fill the sourcequeue and will validate against the resultqueue and will print required time for execution. This park you can run from User laptop too. As this is just to submit the task to remote worker.
   This part of code is required the credentials for the AWS, so if you have to run this code or submit the task to SQS you have to configure the file called **AwsCredentials.properties.**

For Remote Worker part

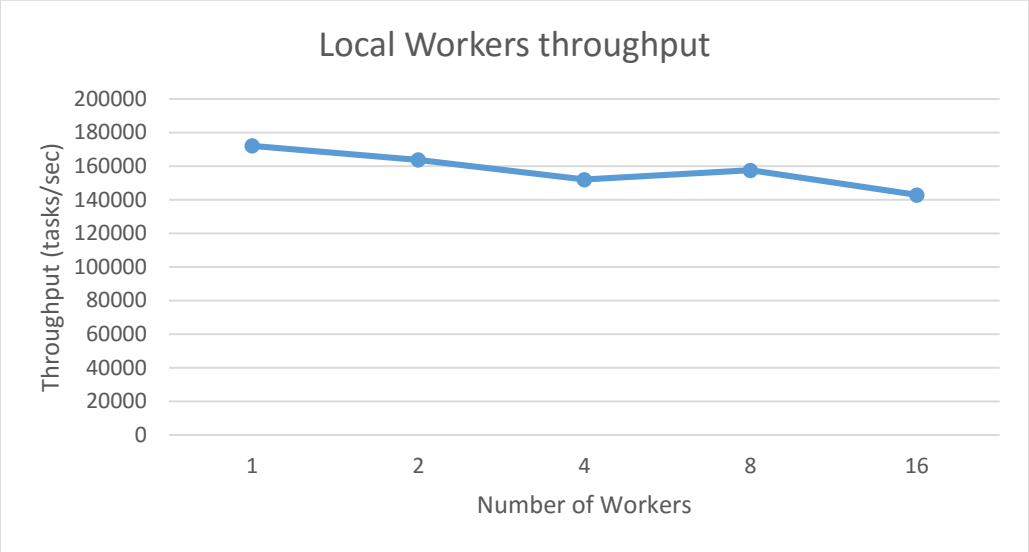
1. Added .pem file to identity first
   cd /home/abhijeet/workspace
   chmod 400 abhijeetkedari.pem
   ssh-add abhijeetkedari.pem
2. Create 16 instances and use cluster ssh to make it easy for operating
   sudo apt-get install openssh-serve
   sudo apt-get install clusterssh
   sudo nano /etc/clusters
   clusters = testcluster
   testcluster = ubuntu@552.164.251.923 ubuntu@52.164.344.325
   cssh -l ubuntu -o "-i aditya.pem" testcluste
3. Copy required code to Amazon EC2 instance
   scp -i abhijeetkedari.pem -r /home/abhijeet/workspace/CloudKonRemoteWorker ubuntu@52.87.183.106:/home/ubuntu
4. On Remote worker build the code
   cd /home/ubuntu/CloudKonRemoteWorker
   ant
5. Run the below command on Cluster ssh window
   java -cp /home/ubuntu/CloudKonRemoteWorker/build/jar/RemoteWorker.jar SQSRemoteWorker client -s SOURCEQUEUE -t 1
6. This will put the all worker in ready state.

   This part of code will always have to be run on cloud as this is worker side not on end user's laptop. This task also requires to consume the AWS services so you have to mentioned the AWS credentials for the same. To provide the credentials you have to update the **AwsCredentials.properties file.**

Performance Evaluation

1. Throughput for Local task execution framework

Workers	Number Of Tasks	Time in sec	Throughput (tasks/sec)
1	100000	0.581	172117.0396
2	100000	0.611	163666.1211
4	100000	0.658	151975.6839
8	100000	0.635	157480.315
16	100000	0.7	142857.1429

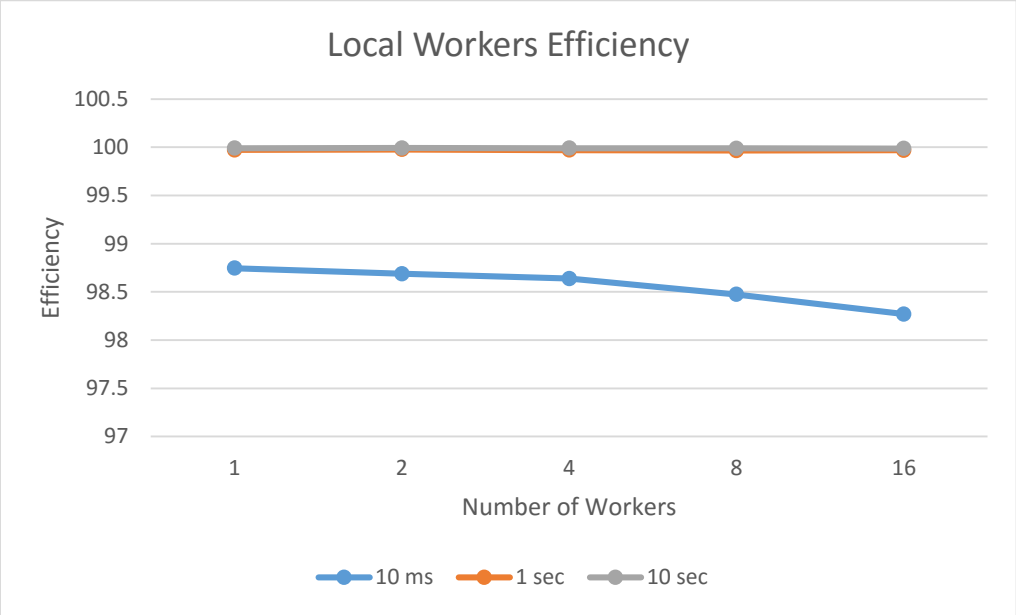


Observation-

- Here we have used t2.micro instance for this experiment, and t2.micro is single core processor instance.
- Here we have fixed number of task irrespective of number of workers.
- So increase in the number of threads is not directly improving the performance (Throughput) of the system.
- Single core processor will get shared between the number of threads.
- Will get maximum performance on single thread only.
- In contrast, increase in number of threads requires more thread handling which results in more time.
- So, here we got such trend of decrease in throughput with more on threads.
- Drawback for Local task execution framework is we have single client listing for resultant queue and many threads. So it is going to take linear time for validation.

2. Efficiency for Local task execution framework

	Task time		
Workers	10 ms	1 second	10 second
1	98.74592673	99.97200784	99.990001
2	98.68745682	99.97700529	99.99200064
4	98.63878477	99.970009	99.99100081
8	98.4736583	99.96501225	99.98800144
16	98.27044025	99.96900961	99.98700169

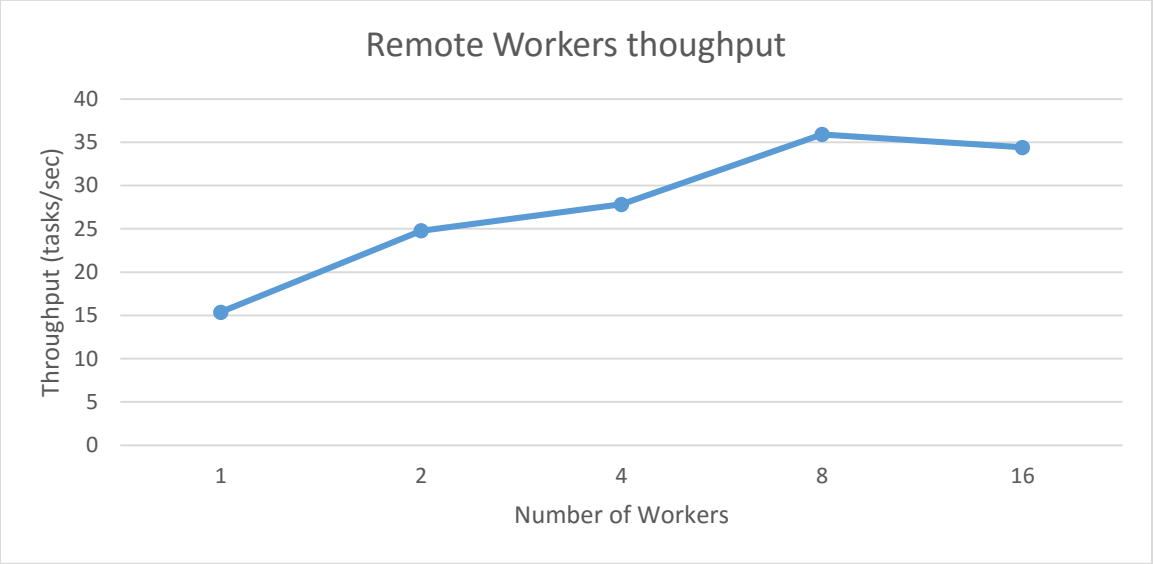


Observation-

- This graph is for efficiency of local worker for task of size 10 ms, 1 second, 10 second.
- Efficiency is calculated by dividing the ideal time by actual time.
- In our case we have got nearly same efficiency for all tasks. Because we ideal time and actual time taken by task are close to each other.
- Efficiency has not much impact of the size of task, because it is going to execute the task on single core so taking same amount time.
- Small amount of extra time taken with compared to ideal time, because we have threads and they need to handle.
- 10 ms task have lowest efficiency compared to other task because we have highest number of task with task length 10 ms. i.e. 16000. So this is putting extra effort on scheduler to distribute the task and resulting lower the efficiency.

3. Throughput for Remote task execution framework

Workers	Number Of Tasks	Time in seconds	Throughput
1	10000	650.459	15.37375915
2	10000	403.368	24.79125761
4	10000	359.315	27.83073348
8	10000	278.485	35.90857676
16	10000	290.672	34.40303848



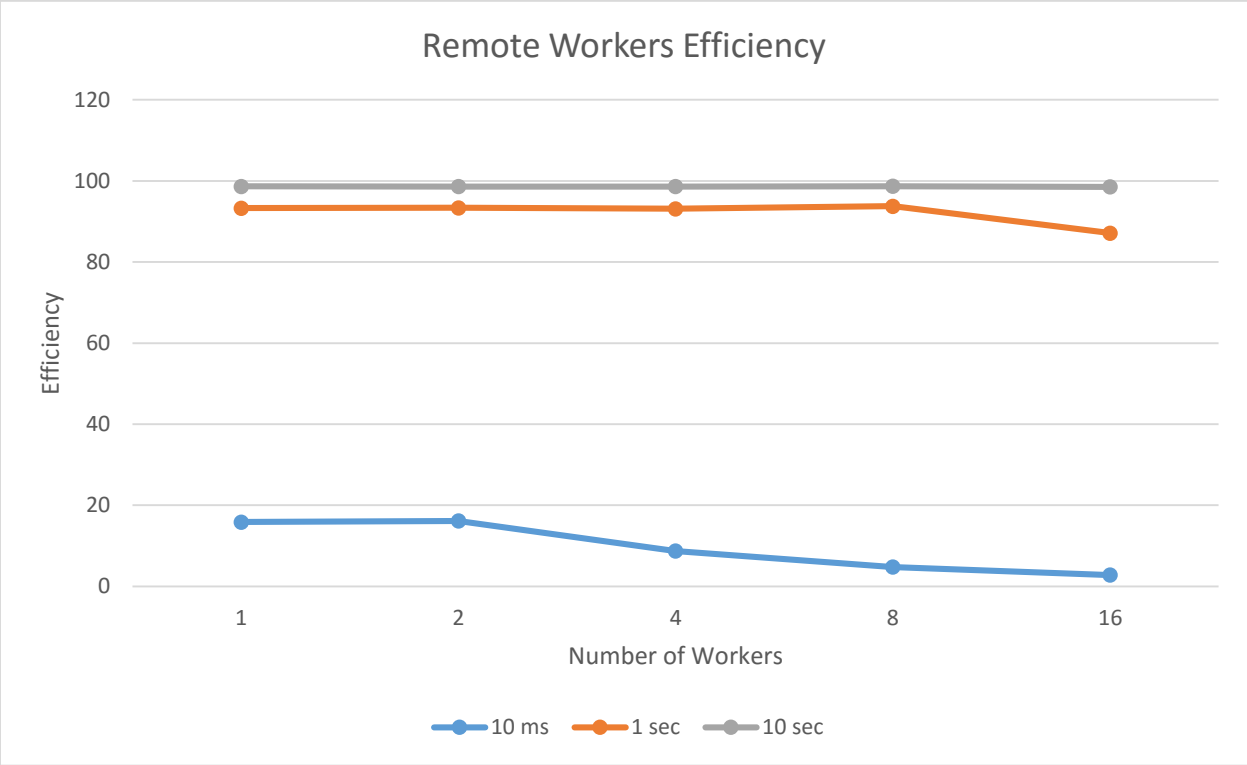
Observation-

- In this case we have kept the number of task constant for all experiment example for 1 worker, 2 workers and 4 workers up to 16 workers.
- All task in this experiment are of fixed sixe that is 0 ms sleep
- So here have fixed length of task and fixed number of tasks, so increase in number of workers are directly impacting the execution time i.e. whole time to execute all task.
- So we are getting higher throughput with increase in number of workers.
- Though with increase in number of worker we are getting higher throughout it is limited after 8 workers.
- This is because even we have more number of workers to work on given task, we have single client to validate the task execution and their status.
- So in case workers finished with their task execution, our experiment keeps adding time for client to validate, this is bottle neck for this design i.e. In remote task execution framework

- We can increase the performance of this designed framework by improving the client side to validate the task completion process.

4. Efficiency for Remote task execution framework

	Task time		
Workers	10 ms	1 sec	10 sec
1	15.88411	93.31405	98.65143
2	16.14466	93.37678	98.61058
4	8.702993	93.1541	98.61544
8	4.794025	93.78048	98.70986
16	2.816386	87.14293	98.53576



Observation-

- In this case we are changing the number of task according to task length.
- 10 ms task have lowest efficiency compared to other task because we have highest number of task with task length 10 ms. i.e. 16000. So this is putting extra effort on scheduler to distribute the task and resulting lower the efficiency

- Similarly, for 1 second task we have lowest number of tasks so it is giving highest efficiency.