Design Document

This assignment will involve implementing a distributed task execution framework on Amazon EC2 using the SQS.
This assignment is implemented in Python programming language.

Client

1. Client takes the input through command line arguments in the form of
client –s QNAME w <WORKLOAD_FILE> , where QNAME is QUEUE name, WORKLOAD_FILE is name of tasks file.
2. Create two AWS SQS queues, one for submitting Workload file tasks to Queue, Other for submitting executed tasks code to queue.
3. Once client starts, Client program reads tasks from workload file and submits tasks to queue. Workload file contains tasks like sleep 0 , sleep 10, sleep 1.
4. After submitting tasks, Client program waits for the tasks execution code status from another queue.
5. Calculate elapsed time from submission of first task to queue and receiving of last task execution code status from other queue.

Local back-end workers

1. Create thread pool in Local back-end worker and takes input from user such as number threads in thread pool using command line. Command line is as below
client –s LOCAL t N w <WORKLOAD_FILE>, where LOCAL is in-memory queue name for submitting tasks, N is number of thread pool threads, Workload_file is tasks file.
2. Local back-end workers are threads number of thread pool thread, where each thread reads task from LOCAL tasks queue and executes it. Back end workers inserts the execution code of task into other queue. Client receives the execution code from that queue.

Remote back-end workers

1. Remote back-end worker accepts input from user through command line as below format worker –s QNAME –t N, where QNAME is AWS SQS already created name for receiving tasks to execute by back-end worker , N is number threads to execute tasks.
2. Create service variable for each AWS SQS queue by using boto3 interface.
3. Remote back-end worker uses dynamoDB to remove duplicate tasks.
4. Remote back-end worker receives message from AWS SQS and checks the taskid in dynamodb table. If tasks id exists in dynamoDb then remote backend skips the job and rceives other task from AWS SQS. If taskid does not exists in dynamodb then worker inserts taskid into dynamodb table and excutes the task. The worker inserts excution code status and task id to AWS queue.

Manual

This Assignment consists of 3 files mentioned as below.

1. client_local.py: This program contains both local client and worker. Client part submits tasks to queue and worker part executes tasks from queue, inserts execute code to other queue.

2. clinet_remote.py: This program submits tasks to AWS SQS queue and wauts for tasks execution status code from other SQS queue

3.worker.py: This program receives tasks AWS SQS and checks dynamodb for task duplicity and executes. Inserts tasks execution status code other AWS SQS queue.

Pre-requistes

Install updates using command sudo apt-get update

Install Python pip using command sudo apt-get install python-pip

---

client_local.py:

1. Open amazon AWS management console in web

2. Create t2.micro instance

3. Open terminal in ubuntu

4. Copy client_local.py file to t2.micro instance using command.
Scp -i “vishwanath” /home/file_path root@ec2-54-200-133-116.us-west-2.compute.amazonaws.com:.

4. Connect ec2 t2.micro instance using command as below
ssh -i "vishwanath.pem" root@ec2-54-200-133-116.us-west-2.compute.amazonaws.com

5.Install updates using command sudo apt-get update

6. Install Python pip using command sudo apt-get install python-pip

7. Run client_local.py as below command
python client_loacl.py -s LOCAL -t N -w WorkloadFile
LOCAL is local queue name.
N is number of threads
WorkLoadfile is tasks file

8. Elapsed time will be displayed in sec as output

---

client_remote.py

Manual

1. Open amazon AWS management console in web

2. Create t2.micro instance

3. Open terminal in ubuntu

4. Copy client_remote.py file to t2.micro instance using command.
Scp -i “vishwanath” /home/file_path root@ec2-54-200-133-116.us-west-2.compute.amazonaws.com:.

4. Connect ec2 t2.micro instance using command as below
ssh -i "vishwanath.pem" root@ec2-54-200-133-116.us-west-2.compute.amazonaws.com

5.Install updates using command sudo apt-get update

6. Install Python pip using command sudo apt-get install python-pip

7. Run client_remote.py as below command

python client_loacl.py -s REMOTE -t N -w WorkloadFile
REMOTE is amazon AWS SQS name.
N is number of threads
WorkLoadfile is tasks file

8. Elapsed time will be displayed in sec as output
---

3. **worker.py**

1. Open amazon AWS management console in web

2. Create t2.micro instance

3. Open terminal in ubuntu

4. Copy client_remote.py file to t2.micro instance using command.
Scp -i “vishwanath” /home/file_path root@ec2-54-200-133-116.us-west-2.compute.amazonaws.com:.

4. Connect ec2 t2.micro instance using command as below
ssh -i "vishwanath.pem" root@ec2-54-200-133-116.us-west-2.compute.amazonaws.com

5.Install updates using command sudo apt-get update

6. Install Python pip using command sudo apt-get install python-pip

7. Run worker.py as below command
python worker.py -s REMOTE -t N

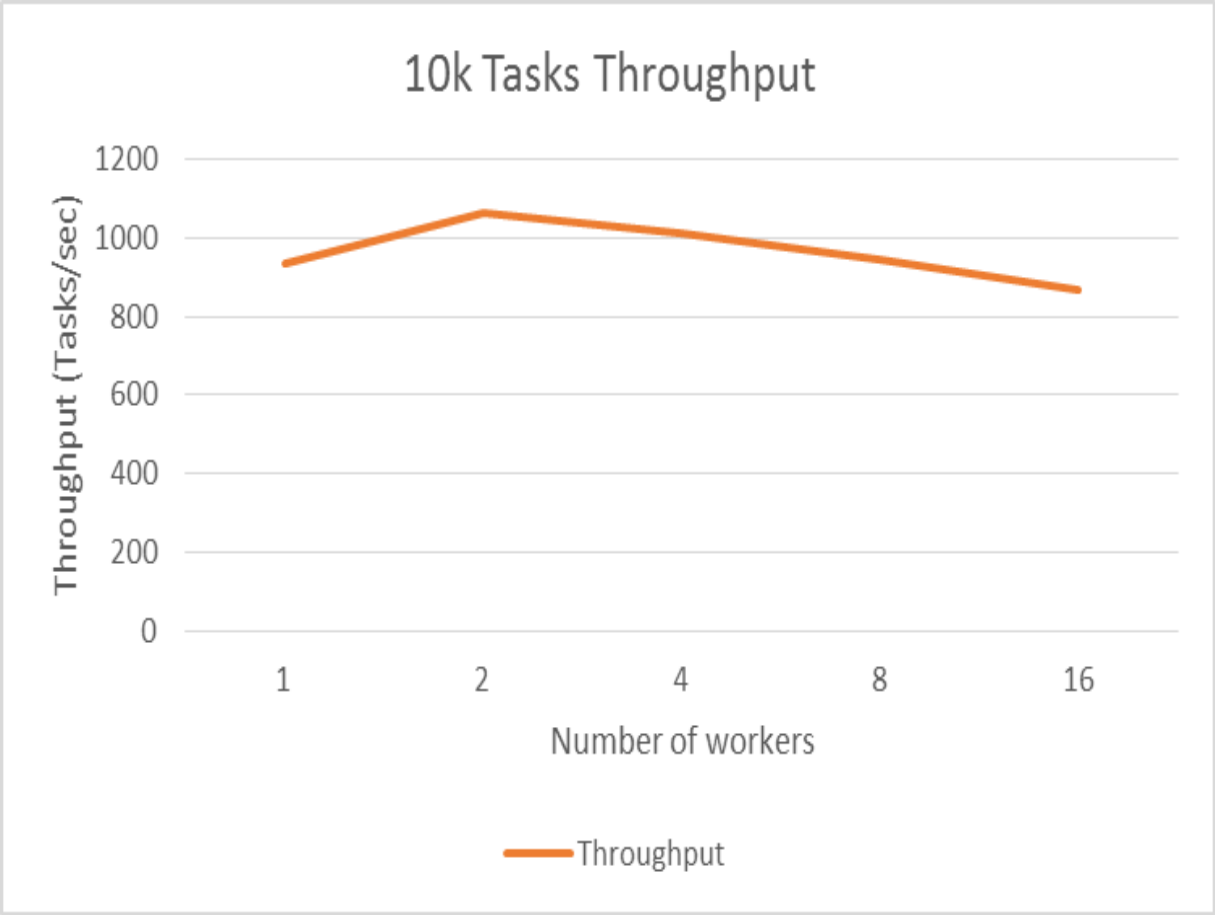
REMOTE: Amazon AWS SQS queue name
N is number of threads

Performance Evaluation

Throughput Graphs of Local workers

Throughput for 10k tasks of sleep 0

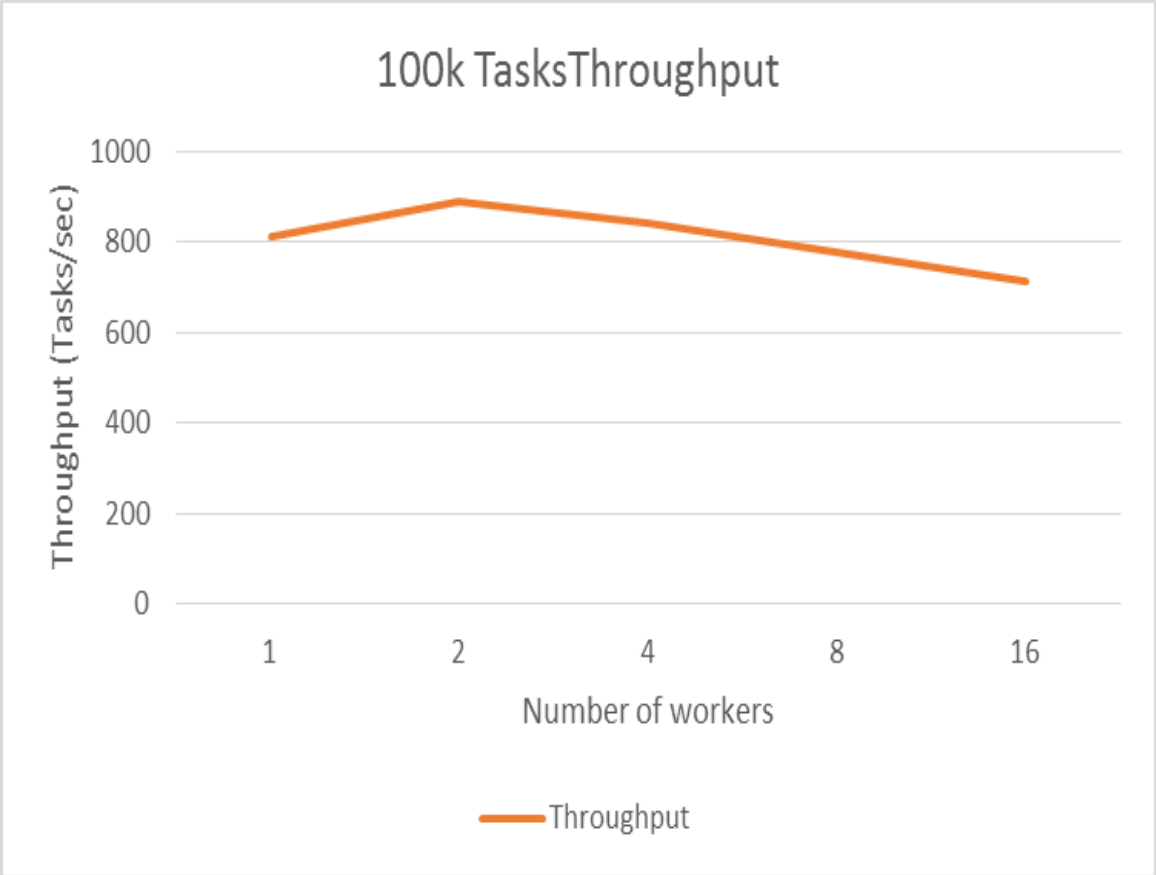
#Workers	Throughput
1	935.7524
2	1065.582
4	1010.826
8	943.6741
16	868.8562



Performance Evaluation

100 k Tasks of sleep 0

#Workers	Throughput
1	810.6482
2	890.7734
4	843.4548
8	776.6729
16	712.4465

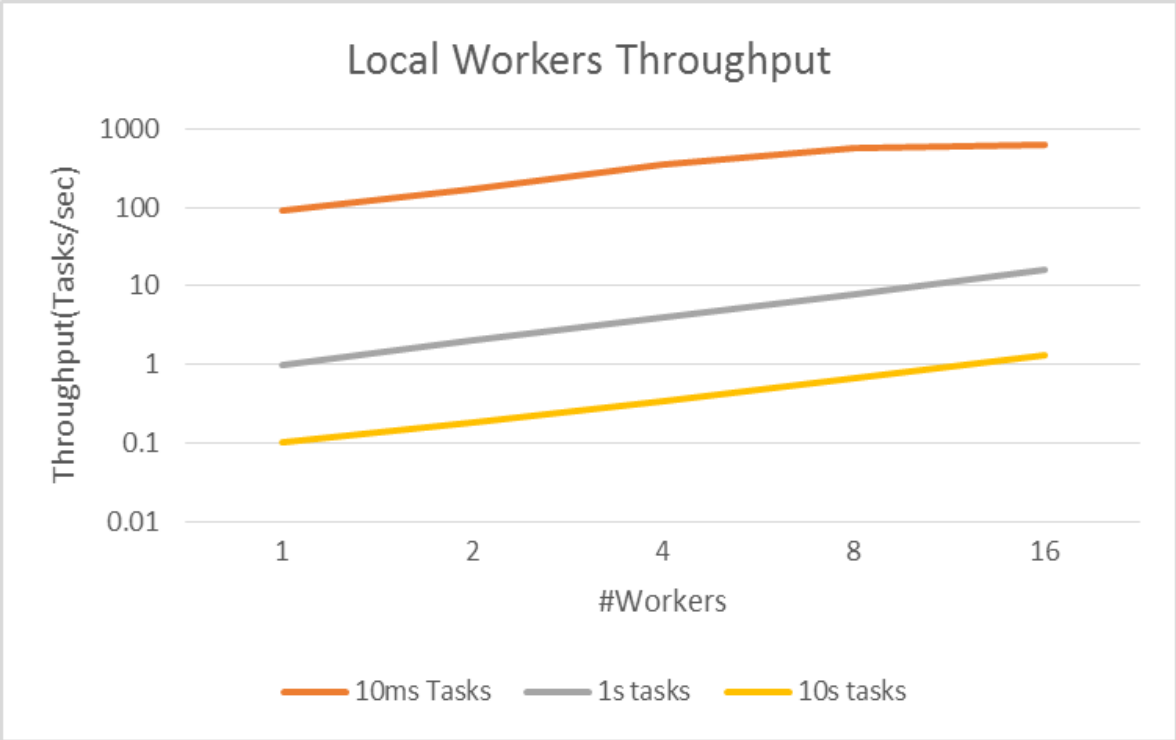


In the above two graphs, number of tasks are fixed for various number of workers such as 1, 2, 4, 8 and 16 workers. Throughput is increasing from 1 to 2 workers because this experiment is conducted in t2.micro. Since t2.micro has 1 Vcpu as number of threads increases throughput will be decreased.

Performance Evaluation

Local Worker throughput

Workers	10ms Tasks	1s tasks	10s tasks
1	89.79991	0.99755	0.099951
2	170.3022	1.993104	0.181698
4	346.1337	3.986123	0.333048
8	555.5452	7.961816	0.666042
16	614.3716	15.73386	1.330854

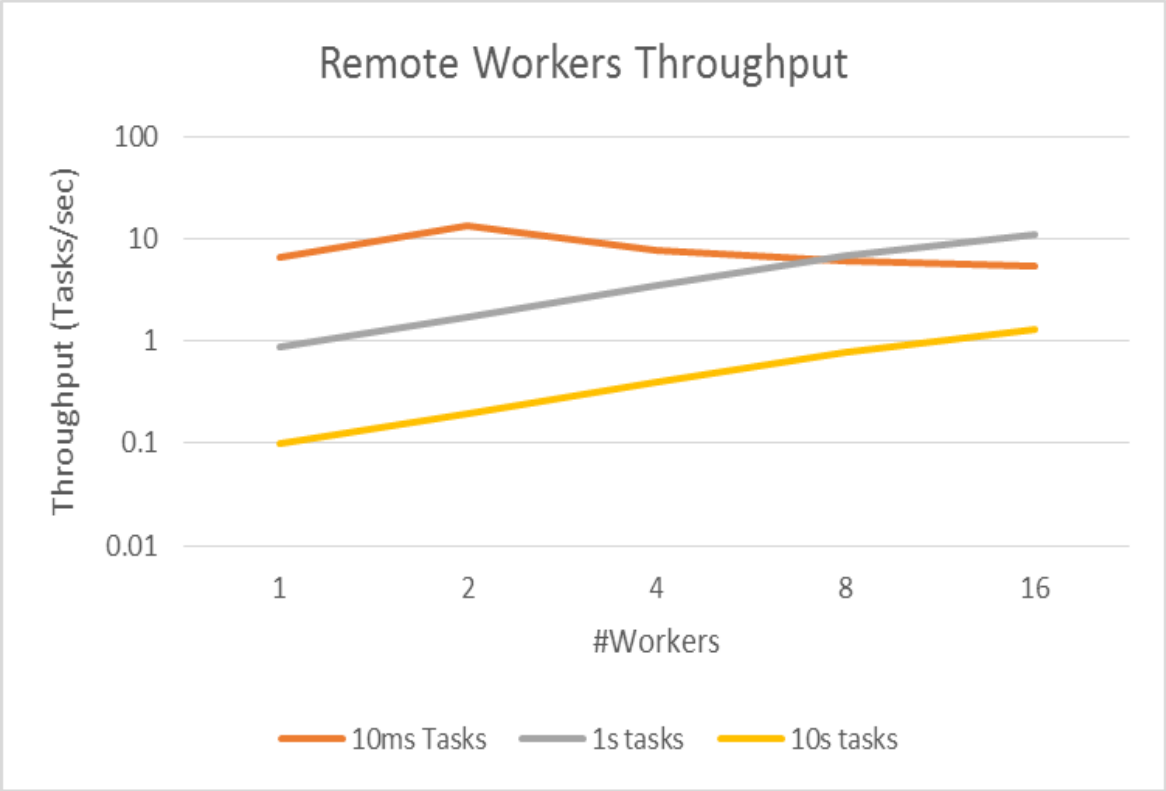


The above graph shows throughput of local workers for different task lengths such as 10ms, 1s, 10s with 1000 tasks per workers for 10ms task length, 100 tasks per worker for 1s task length, 10 tasks per worker for 10s tasks length. Observation is that throughput is increasing for 10ms smaller length tasks compared to 1s and 10s tasks length.

Performance Evaluation

Remote Workers Throughput

Workers	10ms Tasks	1s tasks	10s tasks
1	6.619702	0.867078	0.098341
2	13.34916	1.741159	0.196197
4	7.930948	3.461244	0.392258
8	6.114008	6.964599	0.782854
16	5.499188	10.97179	1.306102



Performance Evaluation

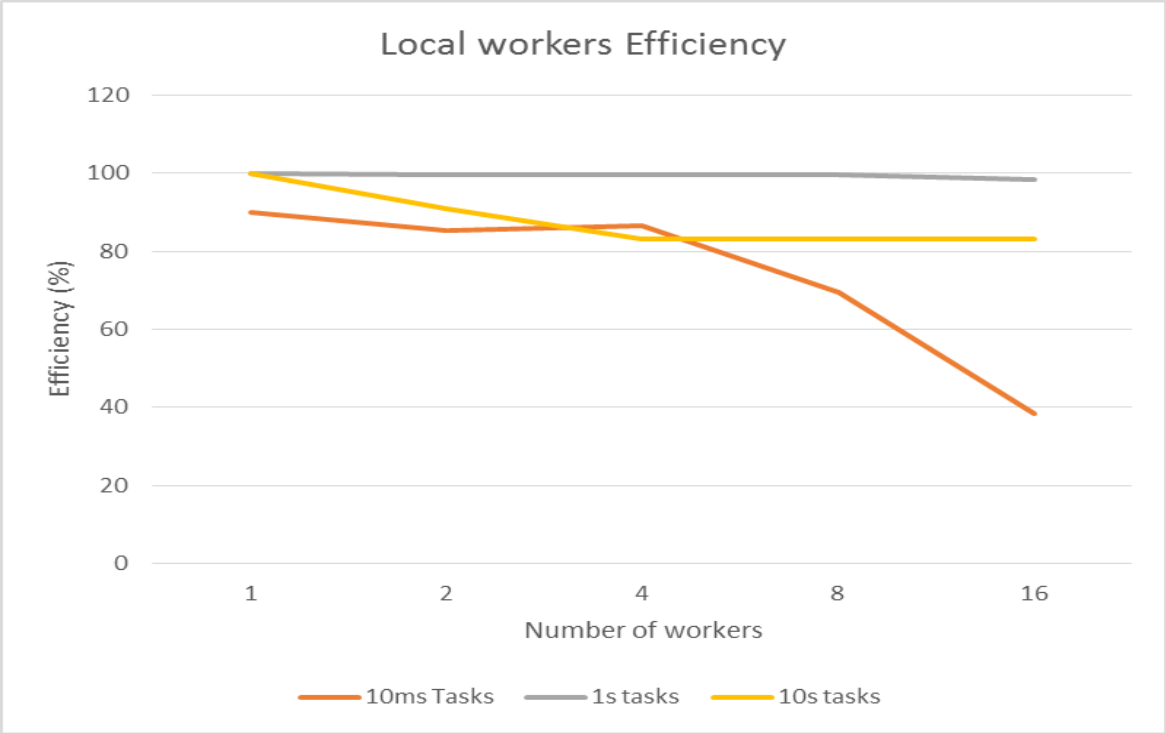
The above graph shows throughput of remote workers for different task lengths such as 10ms, 1s, 10s with 1000 tasks per workers for 10ms task length, 100 tasks per worker for 1s task length, 10 tasks per worker for 10s tasks length. Observation is that throughput is increasing for 10ms smaller length tasks compared to 1s and 10s tasks length for 1 to 8 workers. 10ms tasks throughput is decreasing for 16 workers compared 1s task length because task length is smaller compared latency in case of 16 remote workers.

Efficiency Graphs

Local workers

Workers	10ms Tasks	1s tasks	10s tasks
1	89.79991	99.75505	99.95136
2	85.15111	99.65522	90.84902
4	86.53343	99.65307	83.26203
8	69.44315	99.5227	83.25527
16	38.39823	98.33664	83.17839

Performance Evaluation



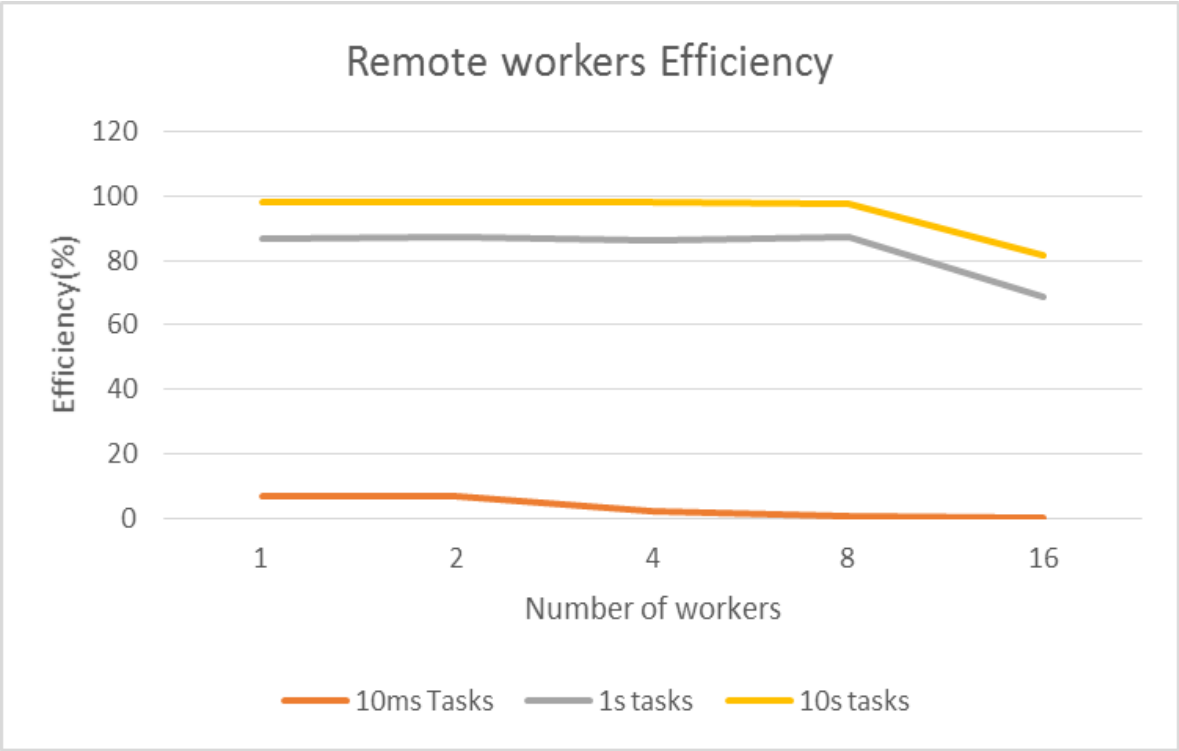
Efficiency is ratio of ideal time of task execution to actual time of task execution. 1s task length is taking almost same time as ideal time .i.e. minimum time taken to complete the task because task length is not too small to have synchronization overhead or too large to take more execution time. 10s task length is little far from ideal time in case of 4, 8 16 workers. 10ms tasks are far from ideal time compared to 1s and 10s task because significant latency in case of 8 and 16 workers to synchronize workers i.e. threads in case of local workers.

Remote workers Efficiency

Workers	10ms Tasks	1s tasks	10s tasks
1	6.619702	86.70783	98.34119
2	6.67458	87.05793	98.0984
4	1.982737	86.53109	98.06458
8	0.764251	87.05748	97.85676

Performance Evaluation

1 6	0. 343699	6 8.57371	8 1.63136



The above graph shows efficiency for remote workers. 10s task workers are taking same time as ideal time to execute task because there will no bottleneck of accessing SQS and Dynamodb, Since each task execution time is 10s. In case of 1s tasks length, taking little more time than ideal time. 10ms length tasks are far away from ideal time to execute tasks because tasks length is too small, so there will be bottleneck problem for accessing SQS queue and DynamoDB table.