Performance Evaluation Document

Vinit Bharat Shah

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

Problem Statement:

The performance evaluation is about gaining an experience with Amazon web services such as Ec2, the SQS queueing service, S3 and DynamoDB. The evaluation is about distributed workload balancing, how job tasks are scheduled between client and multiple workers in distributed workload environment. This assignment includes implementation of distributed execution framework on amazon ec2 using the SQS which is similar to CloudKon. The SQS service is used to handle the queue of requests to load balance across multiple workers.

The entire implementation framework is divided into two components

1) Client – Command line tool which submits tasks to SQS
2) Workers- Retrieves tasks from SQS and executes them.

Evaluation consists of following:

Throughput an Efficiency evaluation using Local Back end workers (In-memory queue)

Throughput and Efficiency evaluation with SQS using Remote Back end workers

Handling of Duplicate tasks using DynamoDB

Server's used:

Amazon Ec2 Instance- t2.micro

Instance Configuration:

Vcpu- 1

Mem GiB- 1

Amazon Services used:

SQS

DynamoDB

Ec2

1. Throughput an Efficiency evaluation using Local Back end workers (In-memory queue)

- Local backend workers evolution consists of pool of threads which are responsible for execution of sleep tasks.
- The evaluation makes use of Pool of Threads, varying from 1 to 16, with different number of sleep tasks, along with varying number in sleep duration
- For calculation of throughput, the sleep tasks is kept at 0ms and number of tasks are being varied from 10 K to 100K with varying the number of local workers to execute the task.
- For % efficiency, the sleep duration is varied to 10 ms, 1sec and 10 sec with varying number of sleep tasks 10000, 100, 10 respectively per worker.
- The time calculated is the roundtrip time, and it's a time taken to send 1st task in queue, until the last task is available in response queue.
- The queue used in this implementation is in memory queue, so there is no need of network connection required to communicate between client and workers.
- Client simply puts data in memory queue, and a worker which repeatedly polls queue for the task, takes the task and executes sleep tasks and put the response as 0 or 1 in to the response queue.

The interface used to execute is:

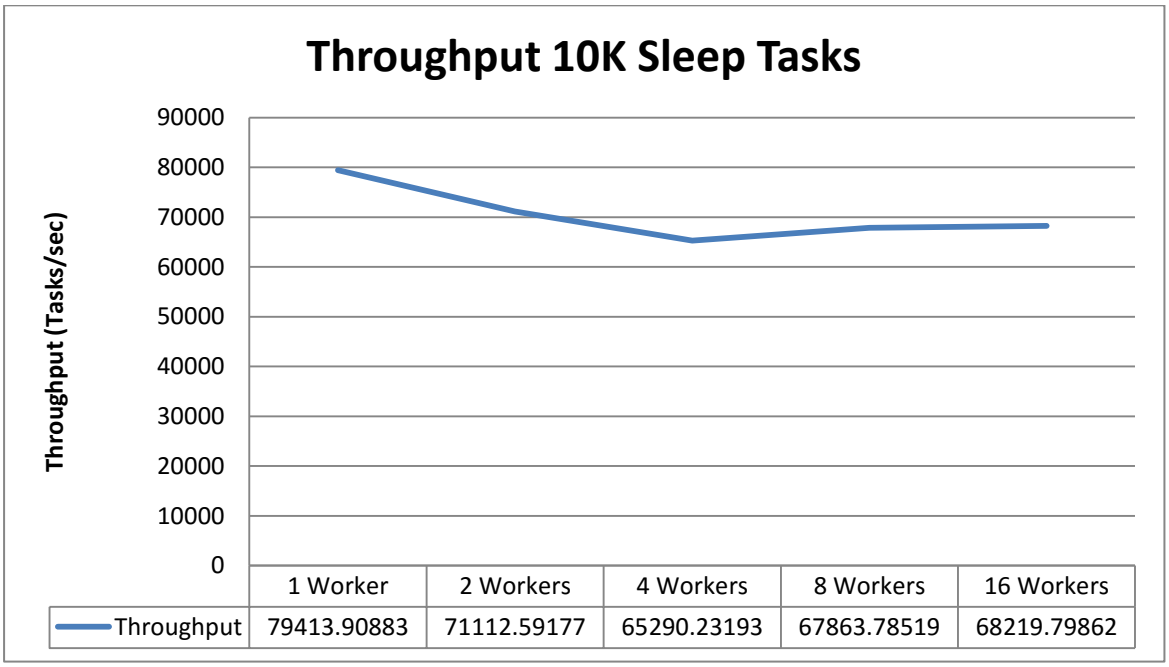
client –s LOCAL t N w <WORKLOAD_FILE>

Throughput:

For throughput evaluation of local backend workers the sleep tasks is varied from 10K to 100K while the number of local thread workers is also varied from 1 to 16.

Sleep Task 10K:

Number of Worker Threads	Throughput (Tasks/sec)
1	79413.91
2	71112.59
4	65290.23
8	67863.79
16	68219.8

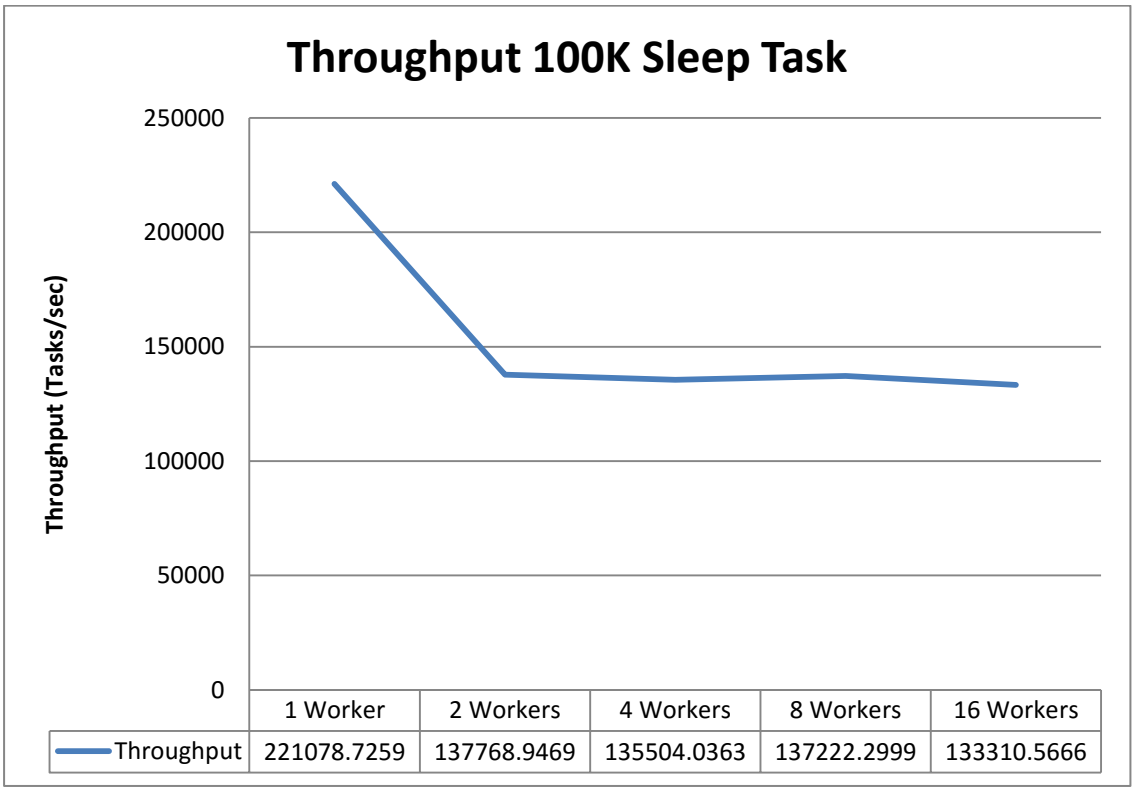


Observation:

Through above graph we can see that as we increase the number of workers for constant amount of sleep tasks, the throughput is decreasing, it's happening because of opportunistic nature of t2.micro for CPU credits.

Sleep Task 100K:

Number of Worker Threads	Throughput (Tasks/sec)
1	221078.7
2	137768.9
4	135504
8	137222.3
16	133310.6



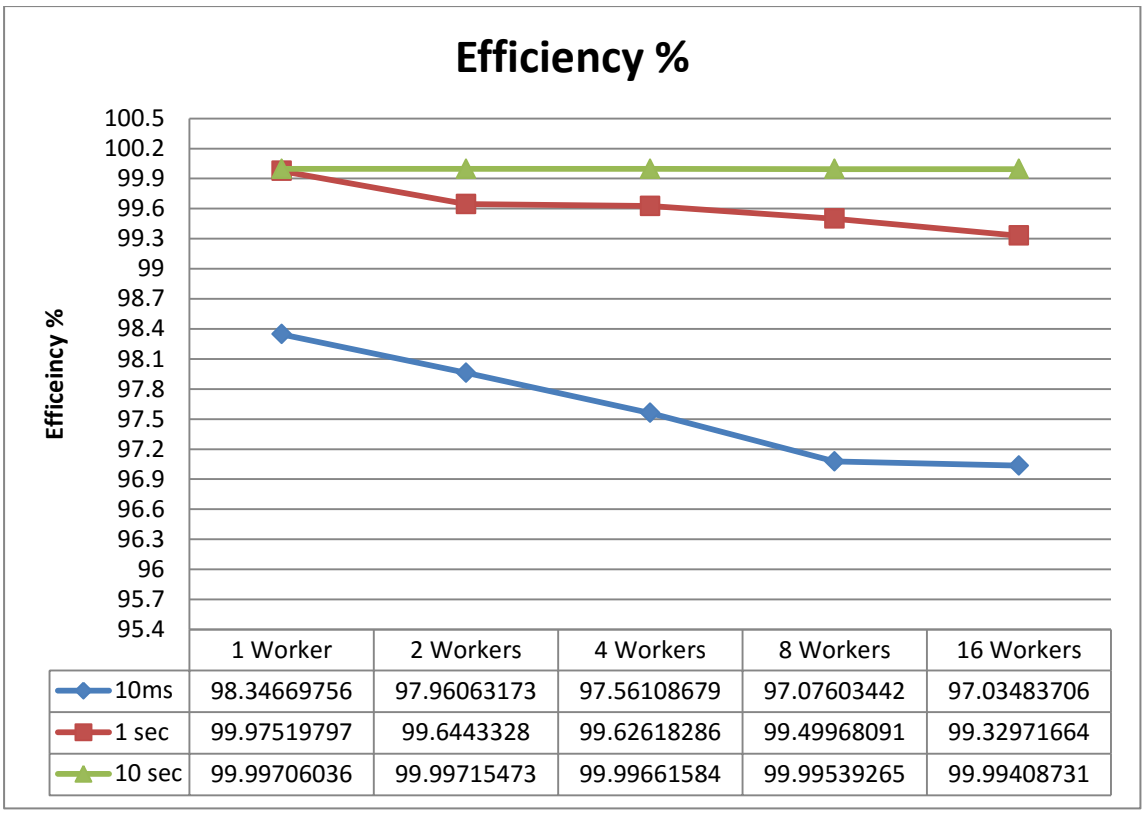
Observation:

As we can observe from the graph that increasing the number of sleep tasks increases the throughput but the time required to execute the task is also increasing. The decrease in curve is because of happening because of opportunistic nature of t2.micro for CPU credits.

Efficiency %:

For efficiency calculation the sleep task duration is varied to 10 ms, 1 Sec, 10 Sec with number of tasks per worker also varied to 10000, 100, and 10 respectively:

	Efficiency %		
Number of Workers	10ms	1 sec	10 Sec
1	98.3467	99.9752	99.99706
2	97.96063	99.64433	99.99715
4	97.56109	99.62618	99.99662
8	97.07603	99.49968	99.99539
16	97.03484	99.32972	99.99409



Observation:

Efficiency is decreasing as for fine task with greater number of task, but it will increase with smaller bigger task with small in numbers. Also as we scale up the workers from 1 to 16, tasks per worker is also increasing resulting in a liner graph.

Throughput and Efficiency evaluation with SQS using Remote Back end workers (SQS and DynamoDB):

- Remote backend workers evaluation consists of workers which are remote from client.
- In order to communicate among workers and client network level queue SQS is used which helps for distributed job scheduling.
- Client who is responsible to generate the tasks for workers put the task in SQS.
- Workers which continuously poll the SQS using long polling mechanism will take the task from SQS and execute it.
- SQS doesn't guarantee the unique deliver of tasks as tasks can be duplicated multiple times
- To avoid duplication DynamoDB is used which helps in storing the message which has already been executed by the worker.
- So whenever, the new task is fetched from SQS, the DynamoDB is checked with message id
- If message ID is already there then worker simply avoids the task, if it's not present then worker will execute the task and make a new entry in DynamoDb with message ID.
- Once the task execution is done, the worker will send the response as a result in response SQS queue, which will then be read by client to make sure that the task is executed successfully.

Interface used for Client:

client –s QNAME w <WORKLOAD_FILE>

Interface used for workers:

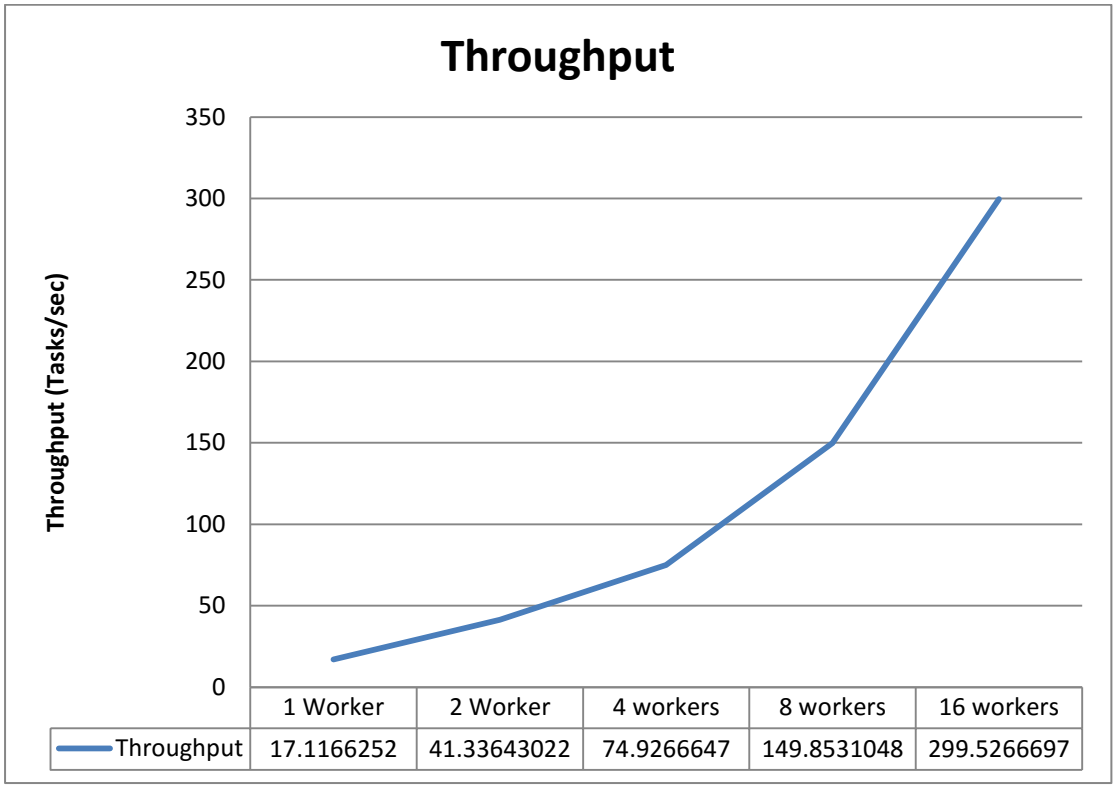
worker –s QNAME –t N

Throughput:

For throughput evaluation of remote backend workers the sleep tasks is 10K while the number of remote workers is also varied from 1 to 16.

Number of tasks: 10K Sleep Duration: 0 ms

Number of Worker Threads	Throughput (Tasks/sec)
1	17.1166252
2	41.33643022
4	74.9266647
8	149.8531048
16	299.5266697



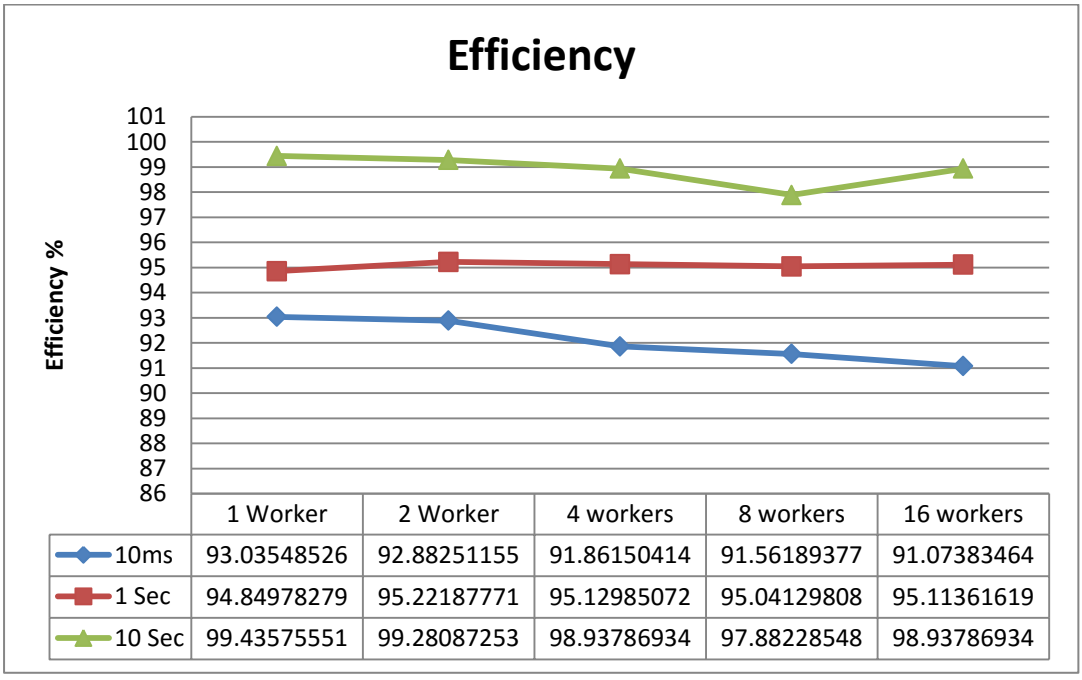
Observation:

Through the observation of above graph, we can make a conclusion that throughput increases as we increase the number of workers while number of tasks to be executed remains constant.

Efficiency %:

For efficiency calculation the sleep task duration is varied to 10 ms, 1 Sec, 10 Sec with number of tasks per worker also varied to 10000, 100, and 10 respectively

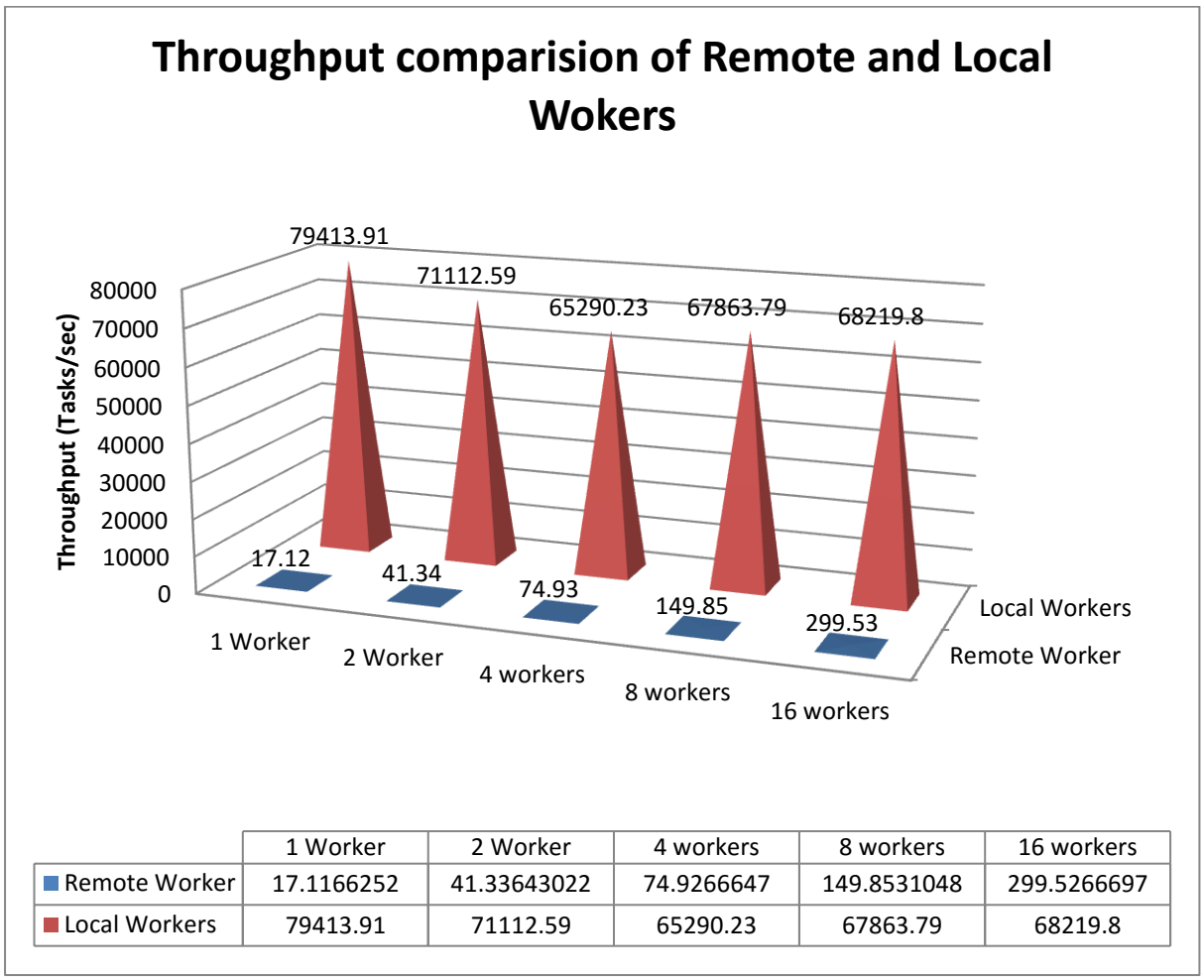
	Efficiency (%)		
Number of Workers	10ms	1 sec	10 Sec
1	93.03548526	94.84978279	99.43575551
2	92.88251155	95.22187771	99.28087253
4	91.86150414	95.12985072	98.93786934
8	91.56189377	95.04129808	97.88228548
16	91.07383464	95.11361619	98.93786934



Observation:

Through the observation of above graph, we can certainly conclude that efficiency will remain almost constant with number of increased remote workers. Such a graph is obtained because workers are running on different instances and each will utilize the core of CPU fully resulting in a trend.

Trade-off of Throughput for Local and Remote Backend workers:



	1 Worker	2 Worker	4 workers	8 workers	16 workers
Remote Worker	17.1166252	41.33643022	74.9266647	149.8531048	299.5266697
Local Workers	79413.91	71112.59	65290.23	67863.79	68219.8

Observation:

As we can see through the comparison of remote and local workers graph trend, we can conclude that the throughput of local workers is way high then throughput of remote workers. Reason being for local workers we have in-memory queue which so for local workers the data are available in memory, where as in case of remote workers the throughput is less, as data is available in SQS which is remote so there will be a network overhead plus to avoid duplicate execution of task DynamoDB related process is also required resulting in a hampering of performance.

Future Scope:

- In future more better instance can be used to test this scenarios which has many cores available
- Notification feature can be implemented to notify the worker about the task in queue.
- Better Load balancing algorithm can be implemented to improve the performance of remote back end workers.

Conclusion:

Through this experiment and observation of graphs for local and remote backend workers execution, we can conclude that the local worker execution has a better performance compared to the remote workers execution.

References:

- Amazon AWS examples used for SQS and DynamoDB code
- https://aws.amazon.com/sqs/
- https://aws.amazon.com/dynamodb/