# CS553 Cloud Computing
# Programming Assignment 3

## Evaluation

**submitted by:**
Chiranjeevi Ankamreddy
A20359837

The assignment is on distributed load balancing and how to distribute work between clients and workers It involves implementing a distributed task execution framework on Amazon EC2 using the SQS and implement a task execution framework similar to Cloudko.I used EC2 to run your framework; the framework should be separated into two components, client (e.g. a command line tool which submits tasks to SQS) and workers (e.g. which retrieve tasks from SQS and executes them). The SQS service is used to handle the queue of requests to load balance across multiple workers and DynamoDb is to Eliminate Duplicate Task Execution.

The Task Execution Framework consists of two parts.Local System and Remote System.

**1.Local System:**

The Local System Framework code has been written in java. It consists of two parts.one is client side and another is Local BackEnd Workers.The client is a command line tool, which can submit tasks to the SQS.I implemented using java, as well as the TCP communication protocol between the client and the SQS. The client should follow this interface: client –s QNAME w The QNAME is the name of the SQS queue.I started time when client starts to send tasks. if QNAME does not exist, the client should terminate. The WORKLOAD_FILE is the local file ( client side ) that will store the tasks that need to be submitted to the SQS.workload file consists of sleep 1000 sleep 10000 sleep 0 sleep 500 Each task is separated by a new line character. Each line is a task description. The Task performs Sleep operation like "sleep 1000" denotes that the task is to invoke the sleep library call for 1000 milliseconds. Since the client is simply reading these task descriptions and send to SQS. And here I Submitting each task one by one.and To be able to keep track of task completion, each task should be given a unique task ID at submission time. Once all tasks had been sent, the client should wait for results to be received at the client, and every received task ID result should be matched with the corresponding submitted task ID.

Local System framework will only support sleep tasks, and therefore you can in run a large number of sleep tasks on the same resource where the client runs, as the tasks are extremely lightweight and do not require significant amount of resources. I implemented configurable size pool of threads that will process tasks from the submit queue, and when complete will put results on the result queue. The pool of threads can be extremely simple, and only have to handle the sleep functionality, of a specified length in milliseconds. When the sleep task is completed, a success returns value 0 should be returned and failure returns returns Value 1. And t the communication between the client and the workers is via the inmemory queue like insert tasks ,remove tasks and retrieve tasks. with a client, and local workers, and you should be able to run sleep workloads.I started the local workers by: client –s LOCAL t N w Where LOCAL denotes that it is the local worker version, N denotes the number of threads in the thread pool.

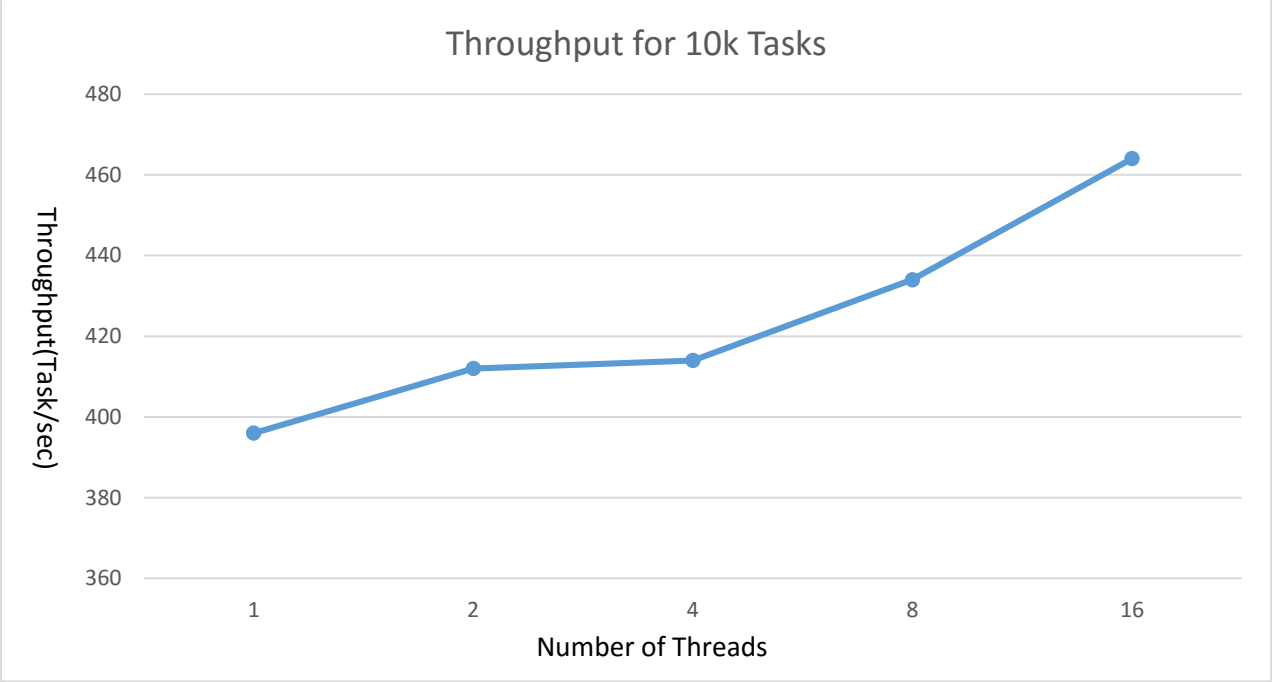
## Performance Evaluation:

In performance evaluation, I have Evaluated Throughput and Efficiency for local System . To determine maximum throughput, I have measured performance running "sleep 0" under a statically provisioned system. Initially I have measured throughput of 10K tasks, where each task is "sleep 0". Preformed benchmark by varying the number of workers from 1, 2, 4, 8, and 16

Here Each Task is Sleep 0 for **10k Tasks:**
Throughput= number of tasks processed divided / total time from submission of the first task to the completion of the last task

| | 10k Tasks | | | | |
|---|---|---|---|---|---|
| Number of Threads | 1 Thread | 2 Threads | 4 Threads | 8 Threads | 16 Threads |
| Throughput(Task/sec) | 396 | 412 | 414 | 434 | 464 |

**Graph:**



The above Graph takes Shows throughput in Tasks/sec with varying 1,2,4,8,16 threads. Initially, Sleep 0 takes 25.4 sec to run on single thread for 10k Tasks. As number of threads increases, latency decreases and Throughput increases. But Throughput constantly increasing as number of threads increases,since all the tasks were shared by other workers . Sleep 0(10k Tasks) takes more than 10 seconds for every run. I performed Same experiment with increasing number of tasks to 100K.

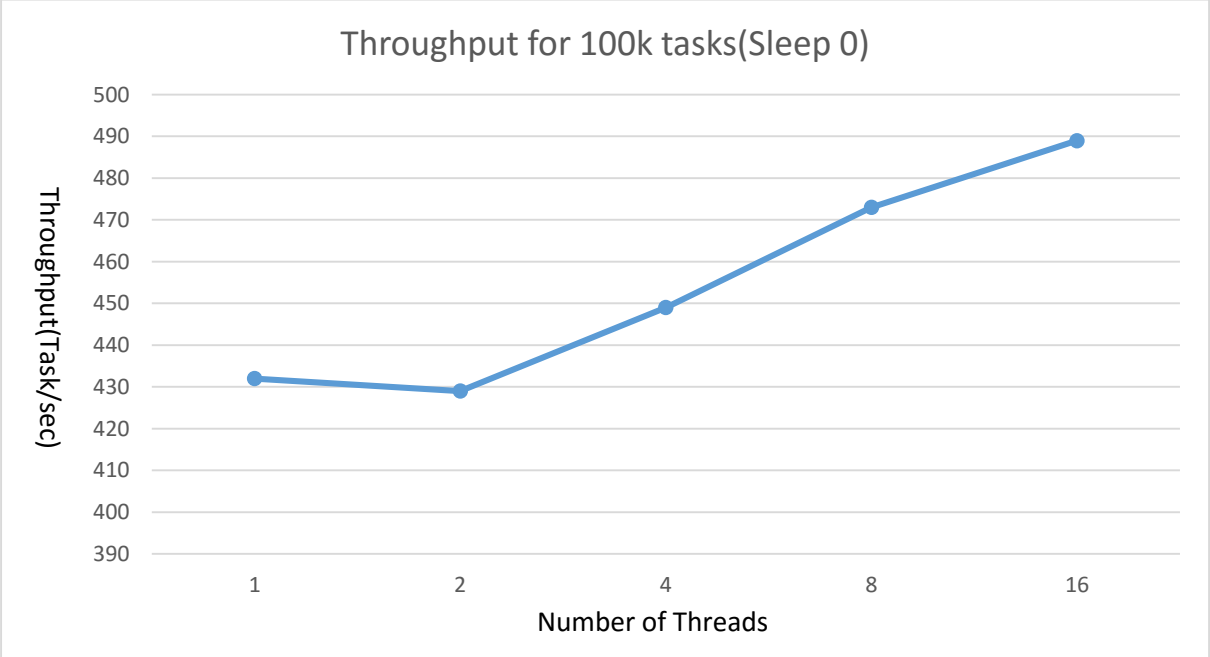
**For 100kTasks,**

I have Performed Similar benchmark for 100k Tasks by varying the number of workers from 1, 2, 4, 8, and 16; and Number of clients is Fixed(Only 1 Client).

Throughput= number of tasks processed divided / total time from submission of the first task to the completion of the last task

| | 100k Tasks | | | | |
|---|---|---|---|---|---|
| Number of Threads | 1 Thread | 2 Threads | 4 Threads | 8 Threads | 16 Threads |
| Throughput(Task/sec) | 432 | 429 | 449 | 473 | 489 |

**Graph:**



The above Graph takes Shows throughput in Tasks/sec with varying 1,2,4,8,16 threads. Initially, Sleep 0 takes 28.6 sec to run on single thread for 100k Tasks.It achieves 446 task/sec. As number of threads increases, latency decreases and Throughput increases same as 10k Tasks. But Throughput constantly increasing as number of threads increases. , since all the tasks were shared by other workers Sleep 0(10k Tasks) takes more than 10 seconds for every run. I performed Same experiment with increasing number of tasks to 100K.

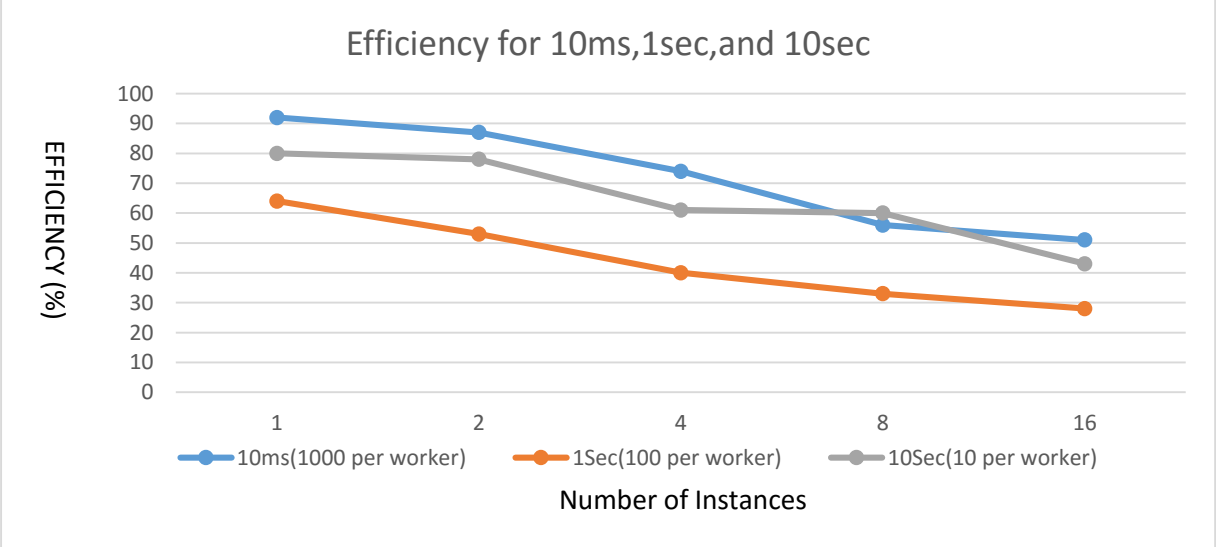
**Efficiency:**

To measure efficiency, in addition to varying the number of workers from 1 to 16 , I varied the sleep time, from 10 ms, 1 second, and 10 sec. The number of tasks should be 1000 per worker for 10 ms tasks, 100 per worker for 1 sec tasks, 10 per worker for the 10 sec tasks. As you increase the number of workers, the aggregate number of tasks will also increase. For example, at 16 workers and 1 second tasks, you will have 100*16 = 1600 tasks. However, at 2 workers and 10 second tasks, you would have 2*10 = 20 tasks. And Performed computation the ideal time to run these experiments assuming zero cost to communicate and distribute the tasks. The following Plot shows the efficiency of different task lengths against the number of workers.

Efficiency isthe ideal time divided by the measured time.

| | Efficiency(%) | | | | |
|---|---|---|---|---|---|
| Number of Instances | 1 Thread | 2 Threads | 4 Threads | 8 Threads | 16 Threads |
| 10ms(1000 per worker) | 92 | 87 | 74 | 56 | 51 |
| 1Sec(100 per worker) | 64 | 53 | 40 | 33 | 28 |
| 10Sec(10 per worker) | 80 | 78 | 61 | 60 | 43 |

**Graph:**



The above Graph takes Shows Efficiency in %with varying 1,2,4,8,16 threads and varying the number of tasks for worker froe Defined Sleep Tasks. Initially, Sleep 10 achieves 92% to run on single thread for 1000 Tasks. and Efficiency slightly Decreases due to Duplicate tasks were executed. And Different workers receives same tasks.It leads High Latency to perform these tasks. And Sleep 1000 achieves 64% to run on single thread for 100 Tasks. and Efficiency slightly Decreases due to Duplicate tasks were executed.but efficiency of 1sec is less than compared to 10ms Since as time of the task increases And due to Different workers receives same tasks.It leads High Latency to perform these tasks.

Sleep 10000 achieves 80% sec to run on single thread for 10 Tasks. Due to less number of tasks ,performance might be increases and Efficiency slightly Decreases due to Duplicate tasks were executed as number of tasks were increased for workers.. And Different workers receives same tasks.It leads High Latency to perform these tasks.

## 2. Remote System

Remote Consists of various Parts. They are client, work reader. I Installed Amazon AWS on Eclipse and Made all the configurationsSqs,DyanamoDb,EC2Instances and S3. The client is a command line tool, which can submit tasks to the SQS.I implemented using java, as well as the TCP communication protocol between the client and the SQS. The client should follow this interface: client –s QNAME w The QNAME is the name of the SQS queue.I started time when client starts to send tasks. if QNAME does not exist, the client should terminate. The WORKLOAD_FILE is the local file ( client side ) that will store the tasks that need to be submitted to the SQS.workload file consists of sleep 1000 sleep 10000 sleep 0 sleep 500 Each task is separated by a new line character. Each line is a task description. The Task performs Sleep operation like “sleep 1000” denotes that the task is to invoke the sleep library call for 1000 milliseconds. Since the client is simply reading these task descriptions and send to SQS. And here I Submitting each task one by one.and To be able to keep track of task completion, each task should be given a unique task ID at submission time. Once all tasks had been sent, the client should wait for results to be received at the client, and every received task ID result should be matched with the corresponding submitted task ID. When running the client, it is important that the client also run in the cloud, in a VM.

And I implemented backend workers using Java and Runs on Amazon Ec2 Instances which have the ability to run on different machines to allow large amounts of computing to scale and leverages Amazon AWS services. In order to allow a general purpose cloudkon enabled solution, i used the SQS service to communicate between the client and the backend workers, and vice versa. I assumed only 1 client, and N backend workers, where N could be 0 to 16; I useed spot instance types to run Workers. The worker should only receive 1 task at a time, and process 1 task at a time. And it allow smultiple tasks to run concurrently at the same and allow completed tasks to be returned back to the client (via SQS) as soon as they are complete. The interface to start workers with worker –s QNAME –t N and the remote workers run on different VMs than the client and other workers.

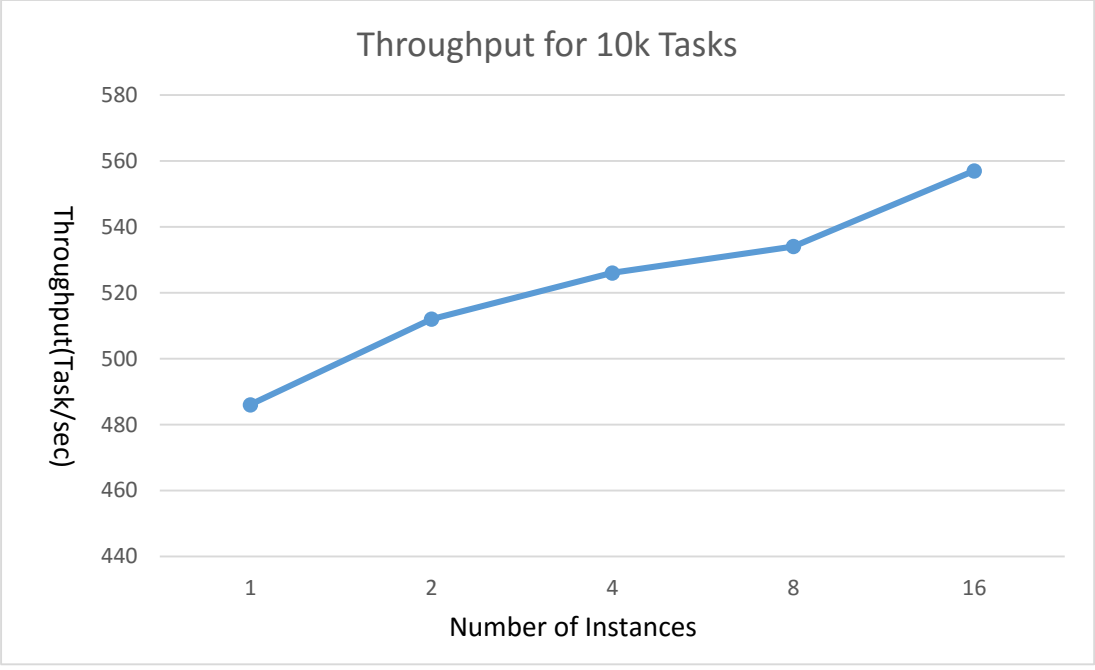
## Performance Evaluation:

In performance evaluation, I have Evaluated Throughput and Efficiency for Remotel System . To determine maximum throughput, I have measured performance running “sleep 0” under a statically provisioned system. Initially I have measured throughput of 10K tasks, where each task is “sleep 0”. Preformed benchmark by varying the number of workers from 1, 2, 4, 8, and 16

Here Each Task is Sleep 0 for **10k Tasks:**
Throughput= number of tasks processed divided / total time from submission of the first task to the completion of the last task

| | 10 Tasks | | | | |
|---|---|---|---|---|---|
| Number of Instances | 1 | 2 | 4 | 8 | 16 |
| Throughput(Task/sec) | 486 | 512 | 526 | 534 | 557 |

**Graph:**



The above Graph takes Shows throughput in Tasks/sec with varying 1,2,4,8,16 threads. Initially, Sleep 0 takes 31.2 sec to run on single thread and throughput is 486 tasks/sec for 10k Tasks. As number of threads increases, latency decreases and Throughput increases. But Throughput constantly increasing as number of threads increases,since all the tasks were shared by other workers .Amazon DynamoDb is to check Duplicate Task execution,it will lead to increase the Throughput.

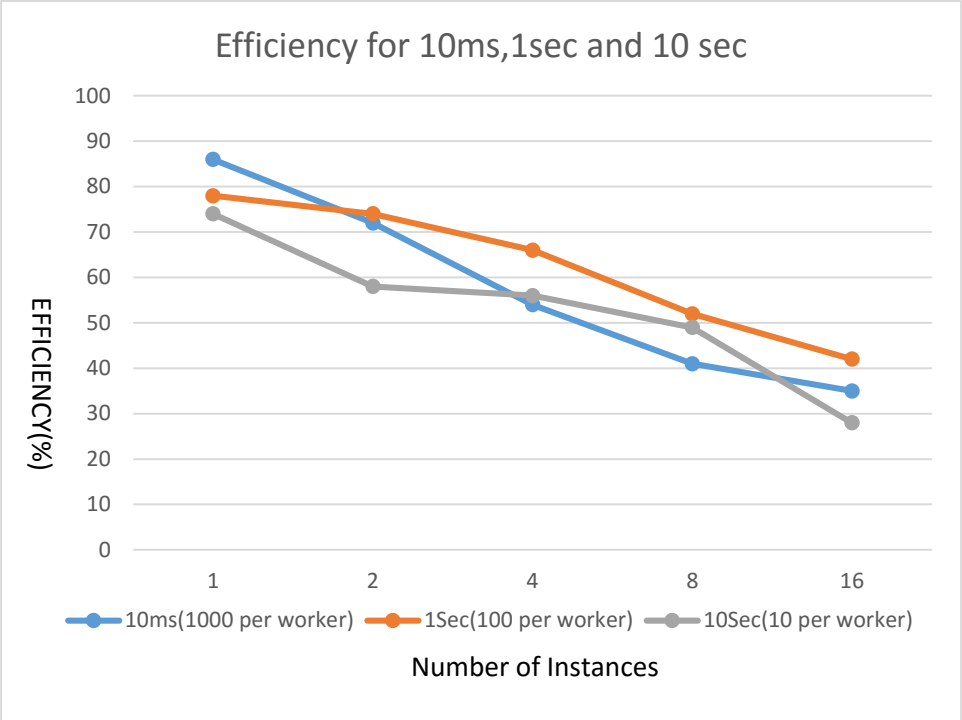
**Efficiency:**

To measure efficiency, in addition to varying the number of workers from 1 to 16 , I varied the sleep time, from 10 ms, 1 second, and 10 sec. The number of tasks should be 1000 per worker for 10 ms tasks, 100 per worker for 1 sec tasks, 10 per worker for the 10 sec tasks. As you increase the number of workers, the aggregate number of tasks will also increase. For example, at 16 workers and 1 second tasks, you will have 100*16 = 1600 tasks. However, at 2 workers and 10 second tasks, you would have 2*10 = 20 tasks. And Performed computation the ideal time to run these experiments assuming zero cost to communicate and distribute the tasks. The following Plot shows the efficiency of different task lengths against the number of workers.

Efficiency isthe ideal time divided by the measured time.

| | Efficiency(%) | | | | |
|---|---|---|---|---|---|
| Number of Instances | 1 | 2 | 4 | 8 | 16 |
| 10ms(1000 per worker) | 86 | 72 | 54 | 41 | 35 |
| 1Sec(100 per worker) | 78 | 74 | 66 | 52 | 42 |
| 10Sec(10 per worker) | 74 | 58 | 56 | 49 | 28 |

**Graph:**




The above Graph takes Shows Efficiency in % with varying 1,2,4,8,16 threads and varying the number of tasks for worker froe Defined Sleep Tasks. Initially, Sleep 10 achieves 86% to run on single thread for 1000 Tasks. and Efficiency slightly Decreases due to Duplicate tasks were executed. And Different workers receives same tasks.It leads High Latency to perform these tasks. And Sleep 1000 achieves 78% to run on single thread for 100 Tasks. and Efficiency slightly Decreases due to Duplicate tasks were executed.but efficiency of 1sec is less than compared to 10ms Since as time of the task increases And due to Different workers receives same tasks.It leads High Latency to perform these tasks.

“Sleep 10000” achieves 74% sec to run on single thread for 10 Tasks. Due to less number of tasks ,performance might increases and Efficiency slightly Decreases due to Duplicate tasks were executed as number of tasks were increased for workers.. And Different workers receives same tasks.It leads High Latency to perform these tasks.

The performance of remote System is better than Local System. Since Dynamo DB plays a important role in remote System.When workers completes the tasks,it will store the task id and results. Duplicate tasks will be eliminated by using Dynamo DB.

3.**Animato Clone:**

I have implemented a web application that converts pictures to video, which will involve both significant network I/O and significant amounts of processing. I have used all the functionalities of Remote System, but

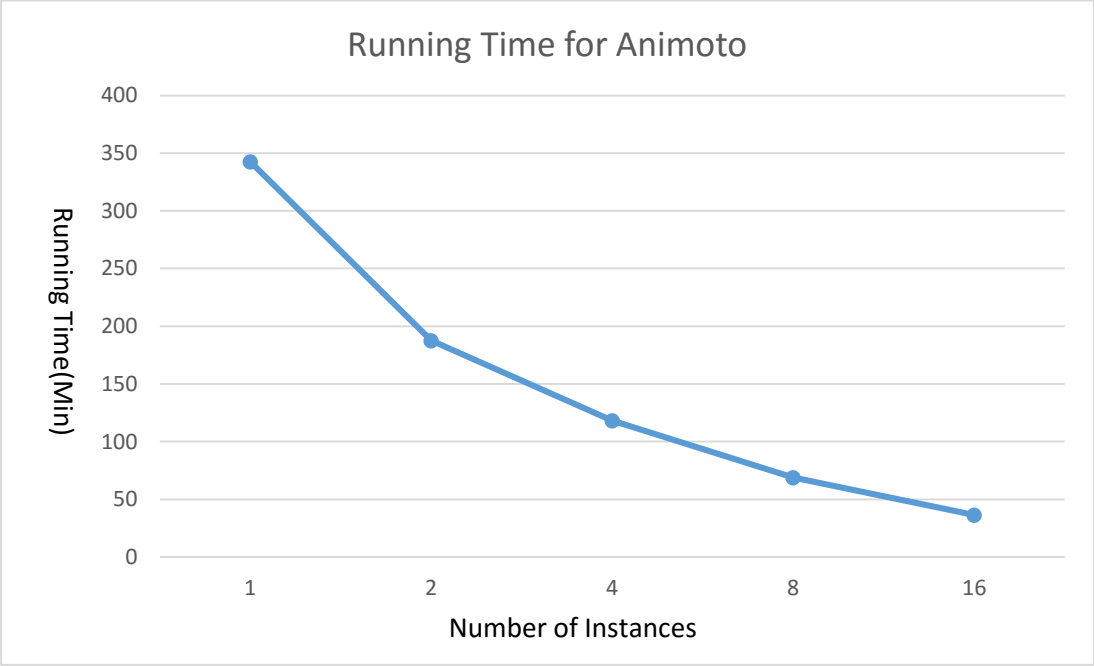
instead of running sleep task, I have used the real time app i.e S3 is used to store the video file. I have used images on the web.and workload will then consist simply of URLs to the images .Using wget is to Download all the images and all the URLs were listed. ffmpeg is used to convert images. Once the video has been created to the local disk, it should be written to S3, and the client notifies the location of the video.

Here Workload is fixed ,Number of tasks are 160 jobs, where each job is a list of 60 pictures (1920*1080 resolution). And generate a 60 second video clip in HD 1080P resolution and store it on S3 and run this workload on 1 node, 2 nodes, 4, 8, and 16 nodes.

Total number of tasks; **160 Tasks:**

| | 160 Tasks | | | | |
|---|---|---|---|---|---|
| Number of Instances | 1 | 2 | 4 | 8 | 16 |
| Running Time(min) | 342.6 | 187.5 | 118.2 | 68.8 | 36.4 |

## Graph:



**Conclusion**:In this Experiment, I Downloaded Images of each size 600-800Kb,Since Each task consists of 60 images. Total time to download 60 images is 120 Secs(approx.).Using wget command is used to download images.Ffmpeg is process images to make video.Inorder to make 60 sec video by using 60 images,it takes 40-50sec.

Each task takes approx. 2min. Total time to complete 160 tasks is 300-350 min. For one thread it takes 342 min ,for two threads it takes 187 min, four threads it takes 118 min, for Eight threads it takes 68 min and for 16 threads  it takes 36min.