Kunal Dhande

Programming Assignmrnt 3 rEPORT

INDEX

Design

* Local Client-Worker Experiment
* Remote Client-Worker Experiment

Installation Manual

* Installation Prerequisites
* Run instructions

Performance Evaluation

* Local Client-Worker Experiment
* Remote Client-Worker Experiment

Animoto Clone

# Design

In this experiment, we are building a cloud-con clone, a distributed task execution framework that has two components - a client (command line tool which submits tasks to Amazon SQS to local queue) and worker (command line tool which retrieve task from SQS or local queue and execute it). There can be multiple client and multiple workers. In this experiment, we are using SQS, Amazon’s Queueing service to handle remote executions.

We are running scaling experiments for throughput and efficiency.

The throughput is calculated as:

*Throughput =*

The efficiency of the system is calculated as follows:

*Efficiency =*

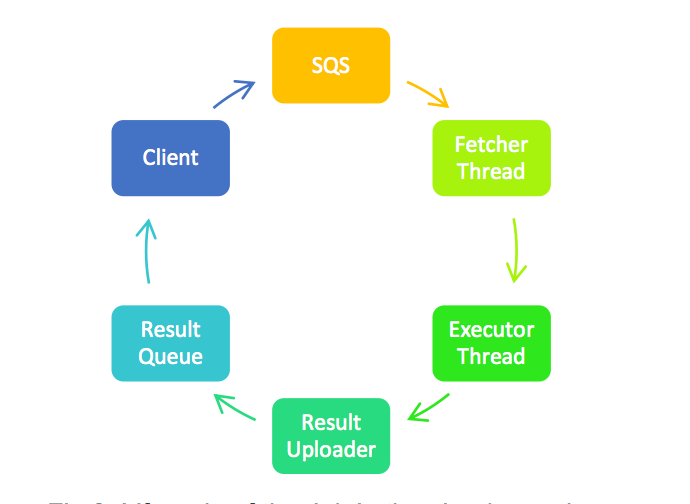
To run this experiment t2.micro instance is used.

## Local Client-Worker Experiment

In local client-worker experiment, distributed execution framework is modeled on a single machine and tasks are shared using in-memory Queue. This should be the fastest way of sharing tasks within single process. To find the scalability of the setup, number of threads are varied from 1, 2, 4, 8 and 16.

In this experiment, client and worker runs within same process.

* Program accepts workload file and number of threads as command line arguments.
* Once programs receive file name, it starts reading file and pushes number of task to in-memory queue.
* After completing this, program one-by-one pop each task and execute it and stores result in into an external file.
* These tasks are executed in number of threads specified by user.



Life Cycle of Remote Client-Worker

## Remote Client-Worker Experiment

In remote client-worker experiment, distributed execution framework is modeled on number of machines and tasks are shared using SQS. This experiment has two different components – client and worker. Unlike, local client-worker, they work independently. To find the scalability of the setup, number of node/ machines are varied from 1, 2, 4, 8 and 16.

Client:

* Client reads workload file and pushes each task to SQS with some queue-name.
* Then, messge\_id from SQS is being stored in Dynamo DB to avoid execution of duplicate task. (SQS guarantees that it will not lose any task, but does not guarantee that it will not repeat task again)

Worker:

* Worker has the same queue-name in which client had pushed task.
* Worker always checks for the new task in SQS as soon as new task added to SQS, worker starts polling them.
* Then, worker check the same task is being executed or not by using Dynamo DB. As soon as task completes, worker deletes the task id from dynamo DB.

# Installation Manual

Here all the steps are mentioned to be followed. All the commands below are run from the scripts folder.

## Installation Prerequisites

### Parallel-ssh

sudo apt-get install pssh

### Ruby (v 2.0 and above)

To install Ruby on Ubuntu run the script in executables folder named ruby-setup.sh. This will install all necessary software packages needed to run cloud-con clone.

./ruby-setup.sh

### Install ffmpeg

Run following command in terminal to install ffmpeg (For animoto clone)

sudo add-apt-repository ppa:kirillshkrogalev/ffmpeg-next

sudo apt-get update

sudo apt-get install ffmpeg

### Assess key for AWS

Get AWS credentials from AWS web interface under menu Credentials and change it in each client and worker files present in executable folder.

## Run instructions

### Local Client-Worker Experiment:

To run cloud-con locally execute

client –s LOCAL -t N -w <WORKLOAD\_FILE>

LOCAL : Queue name indicates that client will run locally

N : Number of worker/ threads to run

WORKLOAD\_FILE : File with number of operations to execute

### Remote Client-Worker Experiment:

In this experiment, we have to run client and worker separately. In this experiment worker is considered as node. This experiment has to be run on 1, 2, 4 , 8 and 16 nodes.

To run client:

client –s QNAME -w <WORKLOAD\_FILE>

QNAME : Queue name for SQS

To run worker:

worker –s QNAME –t N

You can run number of worker parallel with pssh

parallel-ssh -i -v -t 0 -p <NO\_OF\_WORKERS> -l ubuntu -h hosts

-x "-t -t -oStrictHostKeyChecking=no" 'source /home/ubuntu/.bash\_profile; ruby worker < QNAME > 1'

# Performance Evaluation

## Local Client-Worker Experiment

1. Workload file with 100000 Sleep 0 second tasks

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Type of worker** | **Workers** | **Threads** | **Total tasks** | **Time taken (sec)** | **Ideal Time (sec)** | **Efficiency (%)** | **Throughput (task/sec)** |
| LOCAL | 1 | 1 | 100000 | 58.11064649 | 0 | 0 | 1720.855059 |
| LOCAL | 1 | 2 | 100000 | 44.2324121 | 0 | 0 | 2260.785593 |
| LOCAL | 1 | 4 | 100000 | 43.92974377 | 0 | 0 | 2276.362014 |
| LOCAL | 1 | 8 | 100000 | 44.48091555 | 0 | 0 | 2248.155164 |
| LOCAL | 1 | 16 | 100000 | 44.1926558 | 0 | 0 | 2262.819425 |

1. Workload file with 10000 Sleep 10 ms tasks

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Type of worker** | **Workers** | **Threads** | **Total tasks** | **Time taken (sec)** | **Ideal Time (sec)** | **Efficiency (%)** | **Throughput (task/sec)** |
| LOCAL | 1 | 1 | 10000 | 107.1493869 | 100 | 93.32764555 | 93.32764555 |
| LOCAL | 1 | 2 | 10000 | 53.79144645 | 50 | 92.95158116 | 185.9031623 |
| LOCAL | 1 | 4 | 10000 | 27.09604931 | 25 | 92.26437299 | 369.057492 |
| LOCAL | 1 | 8 | 10000 | 13.47669554 | 12.5 | 92.75270755 | 742.0216604 |
| LOCAL | 1 | 16 | 10000 | 6.700074911 | 6.25 | 93.28253912 | 1492.520626 |

1. Workload file with 100 Sleep 1 second tasks

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Type of worker** | **Workers** | **Threads** | **Total tasks** | **Time taken (sec)** | **Ideal Time (sec)** | **Efficiency (%)** | **Throughput (task/sec)** |
| LOCAL | 1 | 1 | 100 | 100.0848587 | 100 | 99.91521329 | 0.999152133 |
| LOCAL | 1 | 2 | 100 | 50.04937458 | 50 | 99.90134826 | 1.998026965 |
| LOCAL | 1 | 4 | 100 | 25.02543879 | 25 | 99.89834829 | 3.995933932 |
| LOCAL | 1 | 8 | 100 | 13.01436567 | 12.5 | 96.04770846 | 7.683816677 |
| LOCAL | 1 | 16 | 100 | 7.009860516 | 6.25 | 89.16011932 | 14.26561909 |

1. Workload file with 10 Sleep 10 seconds tasks

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Type of worker** | **Workers** | **Threads** | **Total tasks** | **Time taken (sec)** | **Ideal Time (sec)** | **Efficiency (%)** | **Throughput (task/sec)** |
| LOCAL | 1 | 1 | 10 | 100.0084007 | 100 | 99.99160003 | 0.0999916 |
| LOCAL | 1 | 2 | 10 | 50.00522733 | 50 | 99.98954644 | 0.199979093 |
| LOCAL | 1 | 4 | 10 | 30.00370336 | 25 | 83.3230475 | 0.33329219 |
| LOCAL | 1 | 8 | 10 | 20.00264573 | 12.5 | 62.49173318 | 0.499933865 |
| LOCAL | 1 | 16 | 10 | 10.00642538 | 6.25 | 62.45986716 | 0.999357875 |

**Efficiency:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Threads \ Time** | **0 sec** | **10 ms** | **1 sec** | **10 sec** |
| 1 | 0 | 93.32764555 | 99.91521329 | 99.99160003 |
| 2 | 0 | 92.95158116 | 99.90134826 | 99.98954644 |
| 4 | 0 | 92.26437299 | 99.89834829 | 83.3230475 |
| 8 | 0 | 92.75270755 | 96.04770846 | 62.49173318 |
| 16 | 0 | 93.28253912 | 89.16011932 | 62.45986716 |

It is observed that efficiency is reducing as increase in number of threads / workers. This is expected as t2.micro instance has only one core and it is spending more time in thread scheduling. Thus, more threads result into more time in scheduling than computing.

Here, task with 10sec sleeping time with single worker thread has highest efficiency.

**Throughput:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Threads \ Time** | **0 sec** | **10 ms** | **1 sec** | **10 sec** |
| 1 | 1720.855059 | 93.32764555 | 0.999152133 | 0.0999916 |
| 2 | 2260.785593 | 185.9031623 | 1.998026965 | 0.199979093 |
| 4 | 2276.362014 | 369.057492 | 3.995933932 | 0.33329219 |
| 8 | 2248.155164 | 742.0216604 | 7.683816677 | 0.499933865 |
| 16 | 2262.819425 | 1492.520626 | 14.26561909 | 0.999357875 |

It is observed that the throughput is increasing with increase in the number of workers. This is expected as there will be more number of threads to do the job with increase in number of threads resulting in comparatively more work done in same time.

Here, task with 0 sec has highest throughput followed by 10 ms task.

## Remote Client-Worker Experiment

1. Workload file with 100000 Sleep 0 second tasks

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Type of worker** | **Workers** | **Threads** | **Total tasks** | **Time taken (sec)** | **Ideal Time (sec)** | **Efficiency (%)** | **Throughput (task/sec)** |
| REMOTE | 1 | 1 | 100000 | 14321.97076 | 0 | 0 | 6.9822793 |
| REMOTE | 1 | 2 | 100000 | 8104.540102 | 0 | 0 | 12.33876306 |
| REMOTE | 1 | 4 | 100000 | 4847.563846 | 0 | 0 | 20.6289186 |
| REMOTE | 1 | 8 | 100000 | 1960.570333 | 0 | 0 | 51.00556624 |
| REMOTE | 1 | 16 | 100000 | 1032.151345 | 0 | 0 | 96.88501644 |

1. Workload file with 10000 Sleep 10ms tasks

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Type of worker** | **Workers** | **Threads** | **Total tasks** | **Time taken (sec)** | **Ideal Time (sec)** | **Efficiency (%)** | **Throughput (task/sec)** |
| REMOTE | 1 | 1 | 10000 | 2269.848737 | 100 | 4.405579912 | 4.405579912 |
| REMOTE | 1 | 2 | 10000 | 1006.454642 | 50 | 4.967933765 | 9.935867529 |
| REMOTE | 1 | 4 | 10000 | 452.8329048 | 25 | 5.52080022 | 22.08320088 |
| REMOTE | 1 | 8 | 10000 | 249.4162654 | 12.5 | 5.011702016 | 40.09361612 |
| REMOTE | 1 | 16 | 10000 | 137.4333827 | 6.25 | 4.547657838 | 72.7625254 |

1. Workload file with 100 Sleep 1 second tasks

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Type of worker** | **Workers** | **Threads** | **Total tasks** | **Time taken (sec)** | **Ideal Time (sec)** | **Efficiency (%)** | **Throughput (task/sec)** |
| REMOTE | 1 | 1 | 100 | 134.3876686 | 100 | 74.41158927 | 0.744115893 |
| REMOTE | 1 | 2 | 100 | 78.09346628 | 50 | 64.02584285 | 1.280516857 |
| REMOTE | 1 | 4 | 100 | 50.102157 | 25 | 49.8980513 | 1.995922052 |
| REMOTE | 1 | 8 | 100 | 24.89731884 | 12.5 | 50.20620927 | 4.016496742 |
| REMOTE | 1 | 16 | 100 | 21.62124918 | 6.25 | 28.90674793 | 4.625079668 |

1. Workload file with 10 Sleep 10 seconds tasks

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Type of worker** | **Workers** | **Threads** | **Total tasks** | **Time taken (sec)** | **Ideal Time (sec)** | **Efficiency (%)** | **Throughput (task/sec)** |
| REMOTE | 1 | 1 | 10 | 117.8522172 | 100 | 84.85203111 | 0.084852031 |
| REMOTE | 1 | 2 | 10 | 130.6565192 | 50 | 38.26827801 | 0.076536556 |
| REMOTE | 1 | 4 | 10 | 33.78493875 | 25 | 73.99747024 | 0.295989881 |
| REMOTE | 1 | 8 | 10 | 21.96062392 | 12.5 | 56.92005858 | 0.455360469 |
| REMOTE | 1 | 16 | 10 | 21.79043339 | 6.25 | 28.68231158 | 0.458916985 |

**Efficiency:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Threads \ Time** | **0 sec** | **10 ms** | **1 sec** | **10 sec** |
| 1 | 0 | 4.405579912 | 74.41158927 | 84.85203111 |
| 2 | 0 | 4.967933765 | 64.02584285 | 38.26827801 |
| 4 | 0 | 5.52080022 | 49.8980513 | 73.99747024 |
| 8 | 0 | 5.011702016 | 50.20620927 | 56.92005858 |
| 16 | 0 | 4.547657838 | 28.90674793 | 28.68231158 |

It is observed that the efficiency of the system is not consistent for the task with high sleep time whereas task with sleep time 0 sec and 10ms is consistent, but too low. On other hand, we can say that that increasing number of workers results into reducing efficiency. This might be because of scheduling overhead. E.g., “Sleep 10 ms” is sleeping for same amount of time as 10 secs, but the number of network connections to get tasks are more in case of 10ms. So, it results into low efficiency.

Here, task with 10sec sleeping time with single worker thread has highest efficiency.

**Throughput:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Threads \ Time** | **0 sec** | **10 ms** | **1 sec** | **10 sec** |
| 1 | 6.9822793 | 4.405579912 | 0.744115893 | 0.084852031 |
| 2 | 12.33876306 | 9.935867529 | 1.280516857 | 0.076536556 |
| 4 | 20.6289186 | 22.08320088 | 1.995922052 | 0.295989881 |
| 8 | 51.00556624 | 40.09361612 | 4.016496742 | 0.455360469 |
| 16 | 96.88501644 | 72.7625254 | 4.625079668 | 0.458916985 |

In this experiment, number shows that increase in number of workers ultimately give high throughput as well. This is oblivious as the increase in number of worker would give more computing power than less number of workers

Here, task with 0 sec has highest throughput followed by 10 ms task.

# Animoto Clone

To run animoto clone t2.micro instance is used.

#### Design

Animoto uses the same design as remote client and server. But, as the functionality is changed, client and worker code is modified according to requirement.

**Client:**

* In remote client, client used to read and consider each line in file as different task. In this case, client read full file. File has number of image links.
* Client joins these links separated by comma, and then push it to SQS.

**Worker:**

* Worker also has similar functionality. Instead of just executing the task in SQS, worker split message retrieved from SQS by comma and download images in temp directory for further processing
* After downloading images, worker execute command (ffmpeg – video processing library) to create video from these images.
* Once video generated, worker, save that video to S3 (Simple Storage Service), cloud storage by AWS.

Animoto clone generally takes 3-3.5 minutes to generate video and store it to S3.

#### Run Instructions

To run animoto client just enter this command in terminal. This will push all video creation operations to SQS.

client –s animoto -w <WORKLOAD\_FILE>

QNAME : animoto.

And to run worker,

worker –s animoto –t N N: number of threads

You can run number of worker parallel with pssh

parallel-ssh -i -v -t 0 -p <NO\_OF\_WORKERS> -l ubuntu -h hosts

-x "-t -t -oStrictHostKeyChecking=no" 'source /home/ubuntu/.bash\_profile; ruby worker < QNAME > 1'

#### Results

|  |  |  |
| --- | --- | --- |
| **Workers** | **Time taken (sec)** | **Throughput (task/sec)** |
| 1 | 28355.73217 | 0.005642598 |
| 2 | 16643.36921 | 0.009613438 |
| 4 | 10295.00625 | 0.015541516 |
| 8 | 9829.142758 | 0.016278124 |
| 16 | 4170.473698 | 0.038364946 |

It is observed that as increase in number of worker, throughput of the system is increasing. This is consistent with our previous experiments.

Animoto clone with 16 workers, completes execution faster than other. Because it has got more computing power and resources. System with 16 workers complete execution almost 1/16th time compared to system with single worker.