CSE 546 — Project Report

Team Members:

- ❖ Pavan Kumar Bhalkey (1217204157)
- ❖ Pushparajsinh Zala (1217568222)
- ❖ Rajashekar Reddy Aluka (1217211645)

General requirements:

- Strictly follow this template in terms of both contents and formatting
- Submit it as part of your zip file on Canvas by the deadline.

1.Problem statement

Clearly describe the problem that you are solving and explain why it is important.

The main aim of the project is to detect objects in a video with low latency and by using the computing resources (cloud and raspberry-pi 3) efficiently. Detecting intruders is a time critical job. For edge computing devices there is always less computing power available. So to improve latency for the given task it is important to distribute tasks properly between cloud and edge devices in our case raspberry pi.

Why are we working on this project:

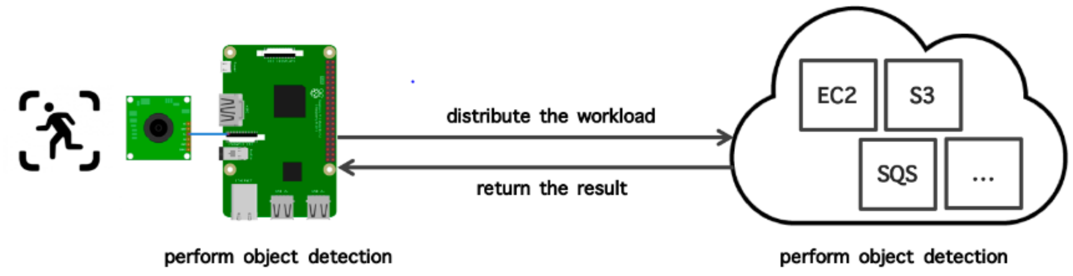
Edge computing is an emerging paradigm for enabling computation on or near the smart devices and Internet of Things (IoT) that produce the input data and provide feedback to users and the physical world.

With the growth of data by various edge devices deployed every day, the cloud is not able to meet all the computational demands, especially when the response time requirement is tight.

The continuous advancement of System-on-Chips has enabled edge devices to conduct more complicated workloads, leading to the emergence of edge computing which extends the computing from the cloud to the edge.

One main advantage of edge computing is the low response time since computation is conducted close to where the input is generated and the response needs to be produced.

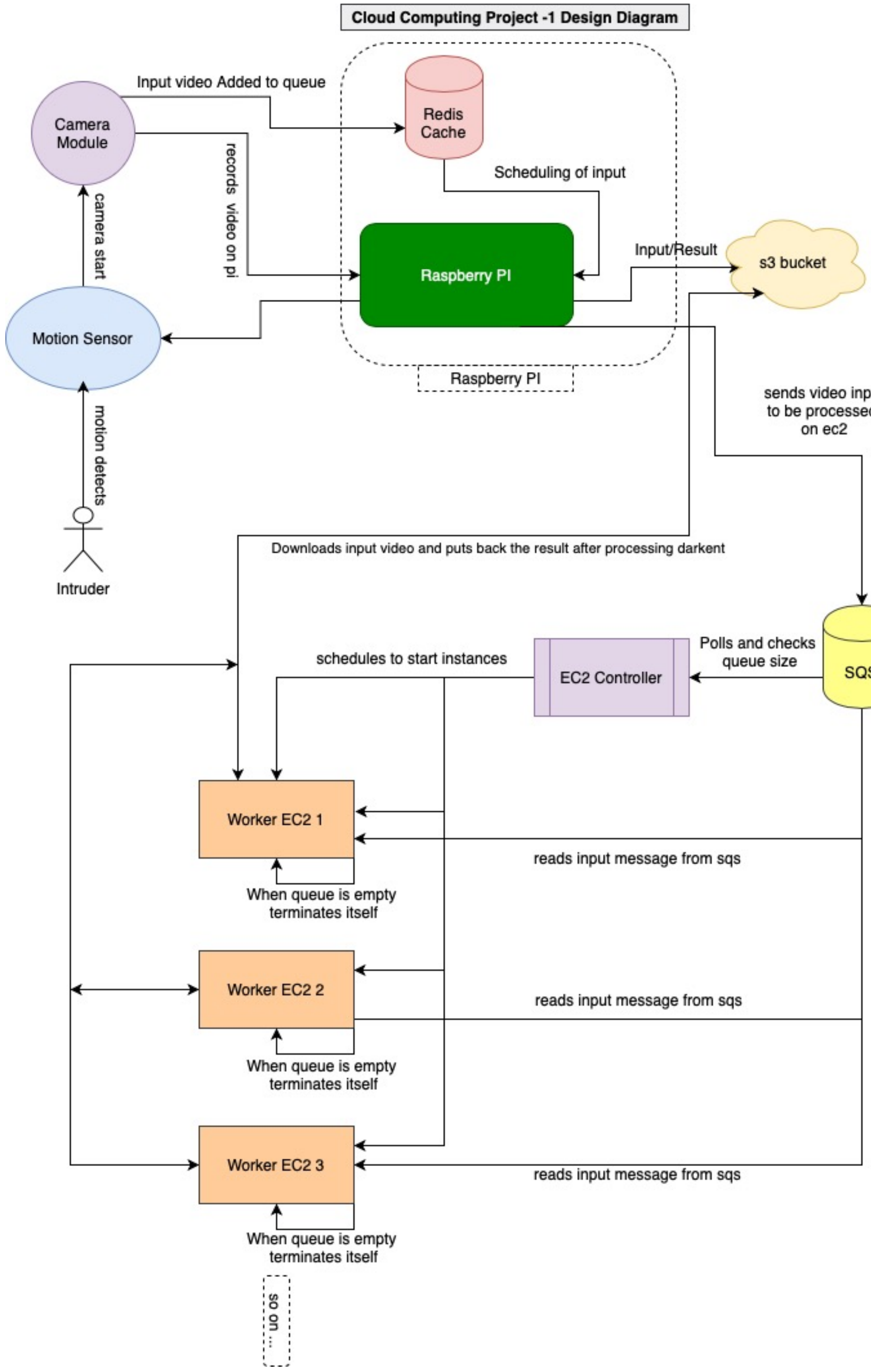
2.Design and implementation



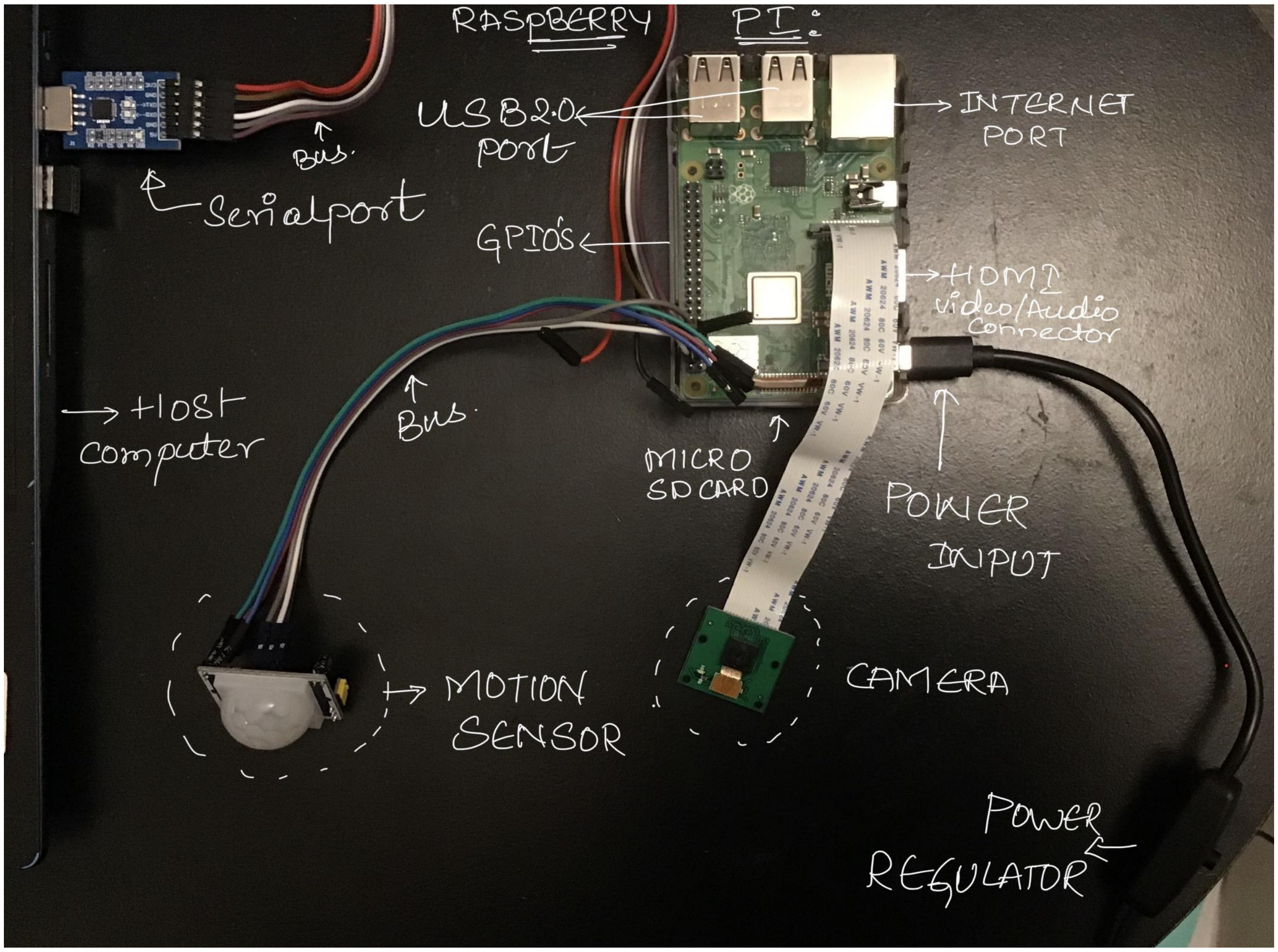
- Raspberry-pi setup
    - OS - setup:
    - Connecting to pi:
        - Serial port connection
        - Directly accessing the Pi
        - Connecting via network
    - Camera setup:
    - Setup the Camera Module
    - Record videos
    - Redis - Queue
- Raspberry-pi to Cloud
    - Redis - Queue
    - Compute input videos
    - S3 Buckets (input & result)
    - SQS Queue
- Aws Cloud
    - Ec2 controller node
    - Ec2 worker nodes
    - S3 buckets (input & results bucket)
    - SQS queue

2.1 Architecture

Following is the architecture diagram for our architecture.


Cloud Computing Project -1 Design Diagram
Input video Added to queue
Camera Module
Redis Cache
records video on pi
camera start
Scheduling of input
Raspberry PI
Input/Result
s3 bucket
Motion Sensor
Raspberry PI
sends video input to be processed on ec2
motion detects
Intruder
Downloads input video and puts back the result after processing darkent
schedules to start instances
EC2 Controller
Polls and checks queue size
SQS
Worker EC2 1
reads input message from sqs
When queue is empty terminates itself
Worker EC2 2
reads input message from sqs
When queue is empty terminates itself
Worker EC2 3
reads input message from sqs
When queue is empty terminates itself
so on ...

Raspberry Pi setup



Pi Controller Logic

Based on the Redis-queue length, it calls an algorithm designed by me to decide the number of videos to be processed on PI and cloud. This algorithm is designed based on the average time taken by the video to process on the cloud. The cloud approximately takes 5 mins to generate the result by processing a 5 seconds video. Similarly, the average time taken by the Pi to process a 5 seconds video is 40 seconds. In total, we can approximately process 7 videos (including the uploading of input) on Pi in 5 minutes.

Now the designed algorithm splits the videos based on redis-queue length (ql) by sequentially allocating the first 7 videos on to Pi and the next (ql-7) on to cloud in a loop. This algorithm optimizes the total number of cloud resources being allocated and results in a decrease in latency time.

2.2 Autoscaling

Explain in detail how your design and implementation support automatic scale-up and -down on demand:

For Autoscaling, we used a controller server for running scheduler code and SQS queue as metric for scaling-up. Controller has a scheduler code which checks no of messages in the queue every 20 seconds to spawn new worker nodes based on the size(no of messages) of the SQS queue with a maximum limit of 20 worker nodes. Controller creates worker nodes from an AMI image created previously. Controller also considers no of worker nodes already running before creating new worker nodes(for 20 worker nodes limit). Each worker node will continuously pull messages from SQS queue & process messages one by one which includes downloading video from s3 process using darknet command & uploading results to s3 bucket. Now for scaling-down when the worker node finds an empty queue it terminates itself.

We are starting worker instances from single stored AMI so we don't need to keep instances in a stopped state so extra storage resources space for all instances. So after the worker instance is done with processing it gets terminated completely thus saving cloud resources.

3.Testing and evaluation

Explain in detail how you tested and evaluated your application.
Discuss in detail the testing and evaluation results:

We have tested our code using 2 ways.

Method 1: (Given test video)

By giving the same type of input videos of length 10. This way we are processing the 7 videos on PI and 3 videos on cloud. Our controller running on PI uploads the videos parallelly and puts a message in SQS that these are the input to be processed on cloud. After this process of the darknet starts on PI.

All the input videos are of the same length and size. Total input videos : 10

This way we are using the 3 EC2 instances and 7 videos we will be processed on PI. To process these 10 videos our code took a total time of 348 seconds i.n 5minutes 48 seconds.

Method 2: (By recording videos using pi camera)

By recording live videos of length 10 using the motion sensor. This way we are processing the 7 videos on PI and 3 videos on cloud. Our controller running on PI uploads the videos parallelly and puts a message in SQS that these are the input to be processed on cloud. After this process of the darknet starts on PI.

All the videos are recorded using a given PI camera. Total number of videos : 10.

This way we are using the 3 EC2 instances and 7 videos we will process on PI. To process these 10 videos our code took a total time of 382 seconds i.n 6minutes 22 seconds.

4.Code

Explain in detail the functionality of every program included in the submission zip file.
Explain in detail how to install your programs and how to run them:

Client(raspberry pi) side:

1) **Motion sensor:** when a motion is sensed by a sensor, a command to record video for 5 seconds starts. After recording the video, a message is sent to redis queue that there is an input video to process. Before running this, create a **cloudProject/inputFiles, cloudProject/resultFiles** folders to store the data.Place the survillence_cc.py file in the darknet folder of PI and then run the following command.
   ```
   python survillence_cc.py
   ```
2) **Redis queue:** We are maintaining this queue to see what all inputs to be processed. This queue is a key value store and it will be easy for retrieval.
   Installation of redis queue on PI: (https://redis.io/download)
   ```
   $ wget http://download.redis.io/releases/redis-5.0.8.tar.gz
   $ tar xzf redis-5.0.8.tar.gz
   $ cd redis-5.0.8
   $ make
   $ src/redis-server ( to start the server )
   ```
3) **Pi Controller:** Now based on the input size of the queue we will decide the number of videos to process on Pi and cloud. Ideally for darkent to process a 5 seconds video takes 30 - 40 seconds to process and get the result. Similarly for darknet on aws ec2 it takes 4-5 minutes to process a single 5 seconds to process. So, if the queue size is greater than 7, then we send the remaining (max 18) videos to the cloud using parallel processing.

   Copy the controller.py to darknet folder. We have to install python3 on PI to run the controller.py because we are using python boto3 to access the results.

   Please follow this link for the installation of python-3 on pi : installation
   Now install virtualenv and activate the virtualenv.
   After that install the dependencies boto3 using the command `pip3 install boto3`
   ```
   python controller.py
   ```
   This will run in a loop and checks if there are any messages in the queue and based on this it will decide whether to process the videos on PI or cloud.

Server side:

There are two main programs: 1) controller 2) worker

1) controller:

   Controller project contains logic for controller code. starup.py is the entry point for the project and running that file starts a scheduler which polls SQS queue size every 20 seconds. We can start controller using following commands:
   ```
   cd controller
   pip3 install -r requirements.txt
   python3 startup.py
   ```

2) worker:

   The worker project contains logic for worker code. Entry point for the project is starup.py and running that file starts a scheduler which reads messages from SQS queue and based on key name it'll download a video file from s3 bucket. After downloading the video it'll run darknet detection command on that video and upload results to s3 bucket. After finishing one video, the scheduler will check for a new message in the SQS queue again and if it finds the message then it'll process that or if the queue is empty then the worker node will terminate itself. We can manually start worker using following commands:

```
cd worker
pip3 install -r requirements.txt
python3 startup.py
```

For creating a worker AMI image first create an EC2 instance from darknet AMI given to us then deploy worker code to the darknet folder and put the startup script(starup.sh) given in the project to the home directory. Add script to crontab: using "crontab -e" add following line to crontab:

```
@reboot starup.sh
```

After that form AWS dashboard select that particular instance and create AMI. Use that AMI id to the controller project code in create_instace.py file.

5.Individual contributions (optional)

For each team member who wishes to include this project in the MCS project portfolio, provide a one-page description of this team member's individual contribution to the design, implementation, and testing of the project. One page per student; no more, no less.

Individual contributions are added in next 3 pages:

Individual Contribution

Name: Pavan Kumar Bhalkey (1217204157)

Raspberry PI and infrastructure set up:

As part of this project, I set up the raspberry pi and the following.

- Raspberry-pi setup
    - OS - setup:
    - Connecting to pi:
        - Serial port connection
        - Directly accessing the Pi
        - Connecting via network
    - Camera setup:
    - Record videos
    - Redis - Queue

I set up the pi based on the instructions given to us in the project document. I downloaded the required softwares and flashed the os to the card and tested the booting of pi.

Installed the pololu for connecting the pi to host os later set up the configurations of the network for the pi connection over the network using ssh by editing the python document to enable remote access.

Camera setup:

Setting up the Surveillance System by using the camera and enabling the pi to access the camera by running the commands.

Record videos:

After making changes to the survellience.py file we were able to start recording the videos using the camera when the intruder was detected near the motion sensor.

Read the motion sensor manual to adjust the sensitivity of the motion sensor for setting the motion sensor sensitivity correctly.

Implementation and Testing:

For testing method2 mentioned below , I wrote the surveillance script, it captures the motion, records the video for 5 seconds and then sends an input (key-value pair) to Redis-queue.

This method gives a latency of 6mins 22 seconds to process 10 videos by using 3 Ec2 instances.

I was responsible for rectifying any configuration related error that used erupt during the testing phase of the project which we used to face due to raspberry pi configurations.

Individual Contribution

Name: Rajashekar Reddy Aluka (**1217211645**)

Design & Implementation:
As part of this project, I designed the method to process the total input videos onto the PI and the cloud. The goal of my individual task is to provide the input to the cloud at the earliest.

To track the input recorded on PI, I introduced Redis-queue (key-value store) to store the input videos recorded on Pi. Key is the filename and value is the comma-separated filename, input path and output path. This key-value is stored in the Redis-queue and is used as input to the controller.

Based on the Redis-queue length, it calls an algorithm designed by me to decide the number of videos to be processed on PI and cloud. This algorithm is designed based on the average time taken by the video to process on the cloud. The cloud approximately takes 5 mins to generate the result by processing a 5 seconds video. Similarly, the average time taken by the Pi to process a 5 seconds video is 40 seconds. In total, we can approximately process 7 videos (including the uploading of input) on Pi in 5 minutes.

Now the designed algorithm splits the videos based on redis-queue length (ql) by sequentially allocating the first 7 videos on to Pi and the next (ql-7) on to cloud in a loop. This algorithm optimizes the total number of cloud resources being allocated and results in a decrease in latency time.

After splitting the number of videos to process on Pi and the cloud, we first upload the cloud videos to S3 bucket and put messages of the corresponding input video in SQS queue (which are later used by the EC2 controller code which is written by Pushparaj) using parallel threads. Using parallel threads, we create a thread for each input file, where the created thread starts uploading files to the S3 and terminates after the uploading is completed.

After the uploading of cloud videos, we start processing the input videos on Pi and create parallel threads to upload the input video and result in the S3 bucket.

Testing the PI Controller:
The design of the Pi controller has been tested in two ways:

1. with an input of 10 videos of the same length and size.
2. with an input of 10 videos recorded by Pi camera, with varying lengths.

For testing the method 1, I wrote a script that directly puts the input messages (key-value pairs) in the Redis-queue. This method gives a latency of 5mins 48 seconds to process 10 videos by using 3 Ec2 instances.

For testing method2, the surveillance script written by Pavan Kumar Bhalkey captures the motion, records the video for 5 seconds and then sends an input (key-value pair) to Redis-queue. This method gives a latency of 6mins 22 seconds to process 10 videos by using 3 Ec2 instances.

Individual Contribution

Name: Pushparajsinh Zala (1217568222)

Design:

I helped in designing architecture for the project. One of the main goals of the project was to utilize cloud resources efficiently. For that I came up with a running worker and controller for detection application. Idea was to dynamically spawn worker nodes which actually does the actual detection from video using darknet program. To efficiently manage cloud resources, I created an AMI image for the worker node with a startup task, so the controller node can create instances from that AMI image. The main advantage of this method is that we can terminate worker nodes when the task is done and results of detection are uploaded to the s3 bucket.

Implementation:

First, I created a python project for controller node. I wrote code for running a scheduler task, for polling the SQS queue and creating worker nodes using AMI image id. Before creating new instances, it also checks for no of running worker instances.

Worker project is also a python project which has a scheduler to check SQS messages from the queue and based on the message it'll download video from s3 bucket. After that it'll process video using darknet command and upload results to s3 bucket. For processing darknet commands I used python subprocess and stored its output and error in temporary output.txt and error file respectively. When subprocess finishes successfully, I process output.txt file to expected result format and upload to s3 bucket. At the end when there is no message in the SQS queue then the worker node will terminate itself.

For creating a worker AMI image, I first created an EC2 instance from darknet AMI given to us then I deployed worker code to the darknet folder and created a startup script(starup.sh) to start the worker python project. I added that startup script in crontab as a reboot script, so it'll fire up whenever a new worker instance is created. I created a new AMI image from that instance. That AMI image id I used in the controller project to start new worker code.

Testing:

To test the worker project, first I created an instance from a darknet AMI image given to us. Then I moved the worker project to the darknet folder to test functionality. I manually ran a worker project and tested if the project is polling from SQS queue correctly or not. I also tested download of video from s3 bucket based on SQS message, detection task and uploading of results to s3 bucket. I also tested if the worker instance is terminating properly when SQS queue is empty.

To test the controller project, I first tested the controller project using my local machine. I tested successful working of the SQS polling mechanism and creation of new worker nodes when there is a new message in the queue. After that I tested end to end working of the controller after deploying it to EC2.

While doing complete end to end project testing, I tested server components and validated working of controller and worker. In the first method when we tested with given videos I validated usage of no of worker EC2 instances. For the second method when we recorded videos from a raspberry pi camera, I monitored controller logs and validated proper creation of worker nodes. I also validated all workers producing proper results and terminating itself after completion of work. We validated that our average latency for all the testing runs were around 5 to 6 minutes.