CSE 546 — Project 1 Report

Shantanu Gupta 1217434439
Yoshitha Gajula 1217127457
Shi Yu Lin 1216287878

1. Problem statement:

The capacity to do real-time on-device processing and analytics is known as edge computing. Think of the edge as the universe of web associated devices and entryways sitting on the field, the partner to the cloud. Edge computing gives additional opportunities in IoT applications, especially for those depending on AI or Machine learning for tasks, for example, object detection, face recognition, and language processing. The ascent of edge computing is an emphasis on a notable innovation cycle that starts with unified preparation and afterward develops into more distributed structures. We have learned in class what the cloud is. Along these lines, cloud and edge computing are two complementary ideas for enabling many demanding applications, for example, AI, AR/VR, and autonomous driving, where cloud resources can provide scalability and edge devices can convey responsiveness.

In this project, we will use both cloud (utilizing AWS) and edge (utilizing Raspberry Pi) to develop an application that gives real-time object detection. AWS is the most common IaaS supplier and offers an assortment of computing, storage, and informing cloud administrations. Raspberry Pi is a minimal effort, credit-card sized IoT advancement platform.

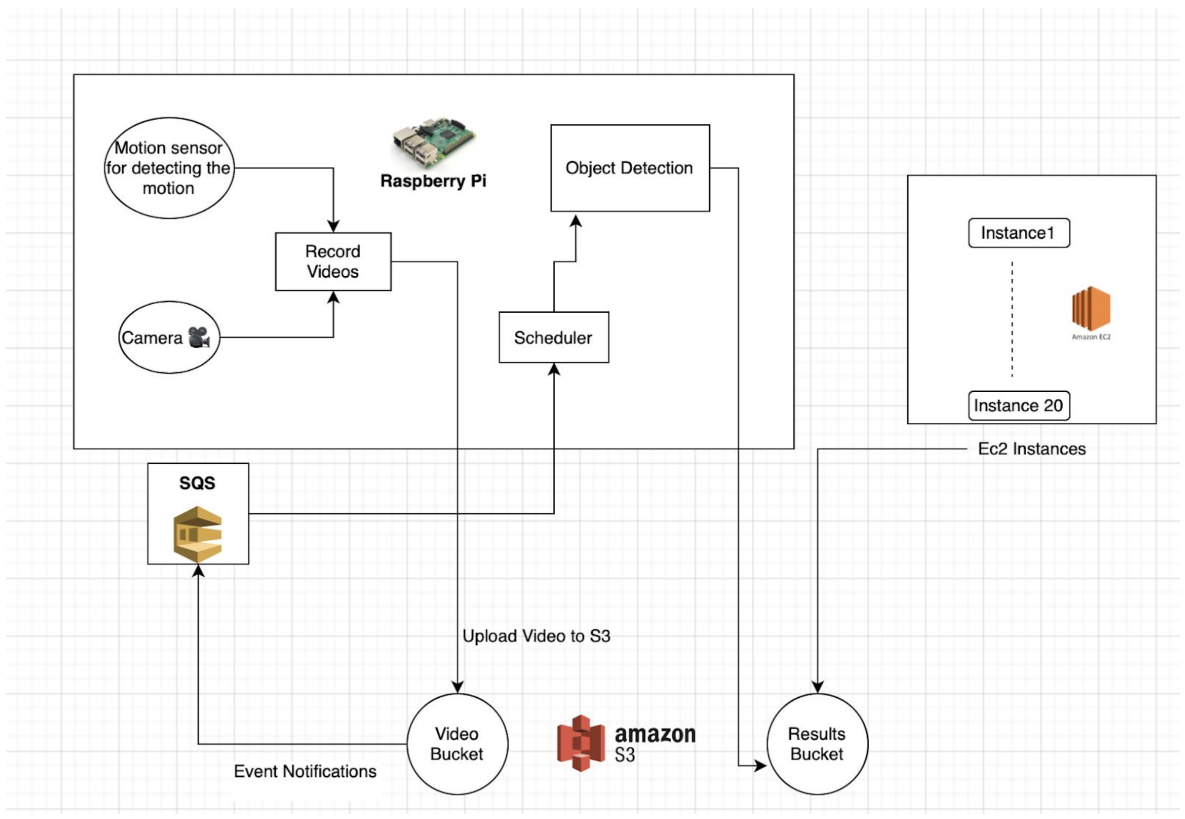
2. Design and implementation

2.1 Architecture

Raspberry Pi is at the core of our surveillance system. The motion sensor attached to the Pi senses motion to trigger the recording and store the recording files locally. Another process working on Pi scans the recording directory and distributes the workload between Pi and cloud. The files are moved from recording directory to ./pi_videos and ./ec2_videos to identify the recordings which are to be analyzed on Pi and cloud respectively. Then Pi starts the analysis first for its assigned recordings; and uploads the files to S3 content bucket and sends messages to the SQS queue for recordings to be processed on the cloud. Once analysis is completed on Pi for its assigned recordings, it uploads the results to S3 result bucket and recordings to S3 content bucket for storage.

Our architecture utilizes an EC2 Master instance that monitors the queue for videos to be processed on the cloud. These requests are pipelined into a queue which will be picked up by EC2 worker instances (19 max, 0 min). The worker instances will be scaled out through the Master instance. The worker instances will scale in by themselves if no messages are available in the queue. When the messages are received by a worker instance, they are made invisible to other workers for a visibility timeout period during which the worker performs object detection on the video. The worker downloads the video recording from S3 based on the 'MessageBody' content of the SQS message. Once recording is downloaded locally, the object identification is performed on that video using the darknet framework. The detection outcomes are processed and returned as a comma-separated ordered list of detected objects, which are put into the S3 content bucket.

We will be working on our application using Raspberry Pi based IoT and Amazon AWS based cloud. Raspberry Pi's camera captures several videos consistently and persistently sends it to the cloud or registers itself relying upon the measure of calculation required, the calculation is object detection on the video utilizing the Darknet model. We have to utilize cloud computing and distributed systems methods to figure as productively as could be allowed and in this way guaranteeing no wastage of processing power. The lesser the wastage of computing and more productive it is, less expensive it will cost our framework to work.



2.1.1 Raspberry Pi:

To use the Pi camera, we need to set up the Raspberry Pi and some local resources. First, we have to figure out the motion detection. Raspberry pi camera will record videos and store them locally.

2.1.2 Amazon services used:

Amazon EC2:

Amazon Elastic Compute Cloud (Amazon EC2) gives an adaptable computing limit in the Amazon Web Services (AWS) cloud. Utilizing Amazon EC2 disposes of your need to put resources into equipment in advance, so you can create and convey applications quicker. You can utilize Amazon EC2 to dispatch the same number of or as barely any virtual servers as you need, design security and organizing, and oversee storage. Amazon EC2 empowers you to scale up or down to deal with changes in necessities or spikes in prevalence, diminishing your need to figure traffic. EC2 allows users to create their own virtual machine in the cloud for running any app. EC2 is utilized to perform object detection on the solicitations that the Raspberry Pi cannot handle in time. We followed the below steps to arrange Darknet on the EC2 instances.

```
git clone https://github.com/pjreddie/darknet.git
sudo apt-get update
sudo apt-get install gcc g++
cd darknet
make
```

Amazon S3:

Amazon S3 is a service provided to store data in the cloud. S3 is used to store the videos(input) that are to be processed by EC2 or Pi. Also, it is used to store the result(output) from EC2 or Pi. In S3 we can store the result in bucket form.

Amazon SQS:

SQS is utilized to couple the different parts in the framework and achieve auto-scaling. When the queue is not empty and messages are available, EC2 will look for the corresponding video in S3. The message body of messages in the queue contains the key of the file in the S3 content bucket, and EC2 will utilize the corresponding key to download the video from the S3 bucket.

2.2 Autoscaling

The Autoscaling logic has been handled in two parts. Firstly, the distribution of workload between Raspberry Pi and cloud and secondly, the number of instances required for the processing on cloud.

I. **Task Distribution logic at Pi**: We have set up the workload distribution logic in such a way, that only Raspberry Pi and a Master EC2 instance is active when there is no processing to be done. When there are less than 7 videos, Pi handles the processing of all videos. If the number of videos pending analysis at run time exceeds 7, new videos are sent to the cloud for processing. We arrived upon this distribution based on the processing time on Pi and EC2 instances. We also took into consideration the time it took to start or stop an EC2 instance to find the optimal balance for best throughput out of our system.

II. **Autoscaling in the cloud**: Scaling out is managed by the Master EC2 instance while the Scale in logic is built into the worker instance themselves.

a. Scaling Out: The Master instance is always polling the queue for an approximate count of visible messages. The first condition is that the number of worker instances must not exceed 19 due to the limitations of free tier usage. The auto scaling algorithm tries to match the number of instances with the number of visible messages for processing. Whenever there are messages visible in the queue, the Master instance polls for the number of active EC2 instances. If any instances are in 'stopping' or 'stopped' states, they are eligible for restarting. The Master instance would attempt to restart these instances to pick up the messages and process them. If no active instances are available to restart, the Master instance triggers the request to create new instances using our custom AMI preinstalled with darknet, scripts and required dependencies.

b. Scaling In: When the worker instances are created or started, the program to receive and process the message from the queue is autostarted. While messages are available in the queue, the worker instance processes it and puts the result object to the S3 result bucket. If the polling request returns empty after a wait time of 20 seconds, the instance initiates the request to stop on its own. This way the worker instance ensures that they are not idle if there are no messages to be processed.

3. Testing and evaluation

When we test the object detection with darknet using Pi and EC2, the execution time is different as the computing power and memory are different for EC2 and Pi. For further evaluations, we noted down the execution times for different videos. In Pi, it took around 40 seconds and in EC2 instance of type t2.micro, it took around 4~5 minutes. Furthermore, starting and stopping an Ec2 instance takes extra time, so it adds up overhead to the processing time. Taking this into consideration, we decided for the case of 10 videos, the first 7 videos are served by Pi and the rest of the videos from SQS will go to EC2 for execution. For deploying our project in the real world, we would keep this ratio as 7:19, meaning at any time if the number of videos to be processed on Pi equals 7, next 19 recordings will be put to SQS for processing on the cloud since we can have 19 worker instances in the free tier limitations.

Our implementation meets the evaluation criteria of the project specification:

1. The Raspberry Pi is able to detect activity using the motion sensor and trigger the recording using the camera provided.
2. The Pi starts processing recordings instantly and scales out to the cloud if recordings count to be processed exceeds 7, which we found ideal through multiple testing rounds.
3. The Master instance in cloud is able to scale out as per the demand creating/starting new instances upto a limit of 19.

4. The recordings are stored in the format "video-%Y-%m-%d_%H-%M-%S.h264" in the S3 content bucket immediately for recordings to be processed on cloud and after analysis for recordings processed on the Pi to minimize latency between analysis and results.
5. We found the best performance for 10 videos to be **7 minutes and 10 seconds**, considering it takes around 1 minute for EC2 instances to start as well.
6. The results are stored in S3 result bucket with key as the recording file name and value as comma separated list of detections made in the same order. For eg:

"video-%Y-%m-%d_%H-%M-%S.h264": "dog,horse,person,cat"

4. Code

For this project, we have used Python language and Boto3 SDK provided by AWS. Here we explain the functionalities of each file in our repository:

1. **survellaince_cfs.py**: This program kick starts the recording on Pi.
2. **record.py**: This program captures the recording for the specified period, which is 5 seconds as per the recommended setting.
3. **upload_script.py**: This program polls the ./record_video directory for *.h264 files and then distributes tasks to Raspberry Pi and cloud according to the specified logic. Once distribution is finalized, it uploads any files to be processed on the cloud to S3 content bucket and sends the message to the queue.
4. **analyze_pi.py**: This program runs the darknet command to do the processing locally for recordings found under the ./pi_videos directory. Once results are parsed for a list of objects detected, it uploads the result to S3 result bucket and the recording to S3 content bucket respectively in that order.
5. **parse.py**: Once the objects are detected, the parse program is called to return the detected objects as a comma-separated list of unique objects detected strictly in that order.
6. **ec2-controller.py**: This program is executed on the Master instance of our cloud system which polls the queue for an approximate count of messages and determines the number of EC2 workers needed based on the Scale Out logic explained in detail later. Once the number of workers to be restarted and created are identified, it calls the methods to restart and create any worker instances respectively. Newly created EC2 instances are created based on the custom AMI created by us which has darknet, code and required modules for our project preinstalled.
7. **ec2-instance.py**: Upon creation or restart, we have configured the rc.local file of our custom AMI to set a virtual display and execute the ec2-instance.py program. The program polls the SQS queue for available messages. Upon receiving a message, it reads the 'MessageBody' for the filename of the video recording and downloads that file onto the local storage under ./ec2_videos. Then it calls the darknet library to perform object detection. If the analysis takes longer than expected, we have embedded logic to change the visibility timeout of the message so that it is not available to other EC2 instances while processing is in progress. Once object detection is complete, it parses

the output of darknet for unique objects, stores the result back to S3 result bucket and deletes the message from the SQS queue. The program looks for available messages in the queue in an infinite loop until the ‘receive message’ request exceeds the wait time of 20 seconds, in which case the instance initiates the request to stop itself.

8. **queue_util.py**: This file contains utility methods for SQS queue like:
    a. creating a queue,
    b. get queue URL based on queue name,
    c. sending message to a queue based on its URL,
    d. receiving an available message based on queue URL, and
    e. deleting a message based on queue URL and receipt handle of the message.
9. **s3_util.py**: This file contains three utility functions related to S3 buckets:
    a. upload list of recordings to S3 bucket ‘content-bucket-546’
    b. upload results to S3 bucket ‘result-bucket-546’ with recording file name as the key and list of detected objects as value.
    c. download recording files to local system at given target directory from S3 bucket ‘content-bucket-546’ for analysis on EC2 instances
10. **ec2_util.py**: This file contains the utility methods for EC2 instances like:
    a. get all instances by ID and state associated with the current AWS account.
    b. create new instances for given custom AMI, instance type etc
    c. start an instance given the instance ID
    d. stop the instance given the instance ID

Installation instructions:

1. Clone below repository on the system

   https://github.com/shantanu-93/pi-eye-py

2. Run below commands under repo directory to install all dependencies
    a. sudo apt-get install python3-pip
    b. pip3 install -r requirements.txt
3. Run the command ‘aws configure’ and enter aws credentials
    a. Access Key as provided
    b. Secret Key as provided
    c. Region: ‘us-east-1’
    d. Output format: ‘json’
4. Booting up project:
    a. For Raspberry Pi, execute below shell file from terminal under repo directory, which will start the surveillance on Pi:

       ```
       ./start_script.sh
       ```

    b. For cloud, on master EC2 instance, run the python program under repo directory with below command:

       ```
       python3 ec2-controller.py
       ```

5. Individual Contribution

Yoshitha Gajula - 1217127457:

Design :

I worked on the initial idea design phase through pitching some research about Raspberry Pi and ideas about maintaining the instances in Ec2 from the start. We came up with the architecture after discussing a lot of models. Contacted AWS for student account approvals. I took the AWS Ec2 part and had worked on running the instances on the Ec2 part mainly for the project.

Implementation :

For the implementation part I implemented adding a message and fetching that message to send data to Ec2. Worked on analyzing the videos in Ec2 and saving the results to S3. Operations in S3 includes sending files to a bucket (videos and results), fetching files from the bucket. In Ec2 subprocess logic for running darknet, getting to EC2 instances through ssh associations, introducing darknet and python in EC2 examples. Executed the initial stage coding of EC2 examples where the code frames various strings to perform darknet object identification on recordings and store the yield to S3, with string synchronization to end an occasion if all the strings have halted and different changes.

Testing:

I reviewed all the test cases to be tried and recorded the equivalent in the report. Played out the test cases identified with S3, SQS and EC2 occasions to test the arrangement of the ideal establishment in EC2. Working off message lines, if adding a message to the line and evacuating is working. Also worked on Transferring and downloading of information to S3. if the occurrences are ready to run darknet and process the outcomes, stringing situations are working, and so forth. I had to do manual testing in some cases to check the implementation. The entire group contributed in all the testing so different perspectives to the test situations can be tried as a group. Likewise contributed towards the final testing.