CSE 546 CLOUD COMPUTING

Project 2 Report

Sudarshan Reddy Mallipalli - *Darshin Shah* - *Venkata Pavana Kaushik Nandula*

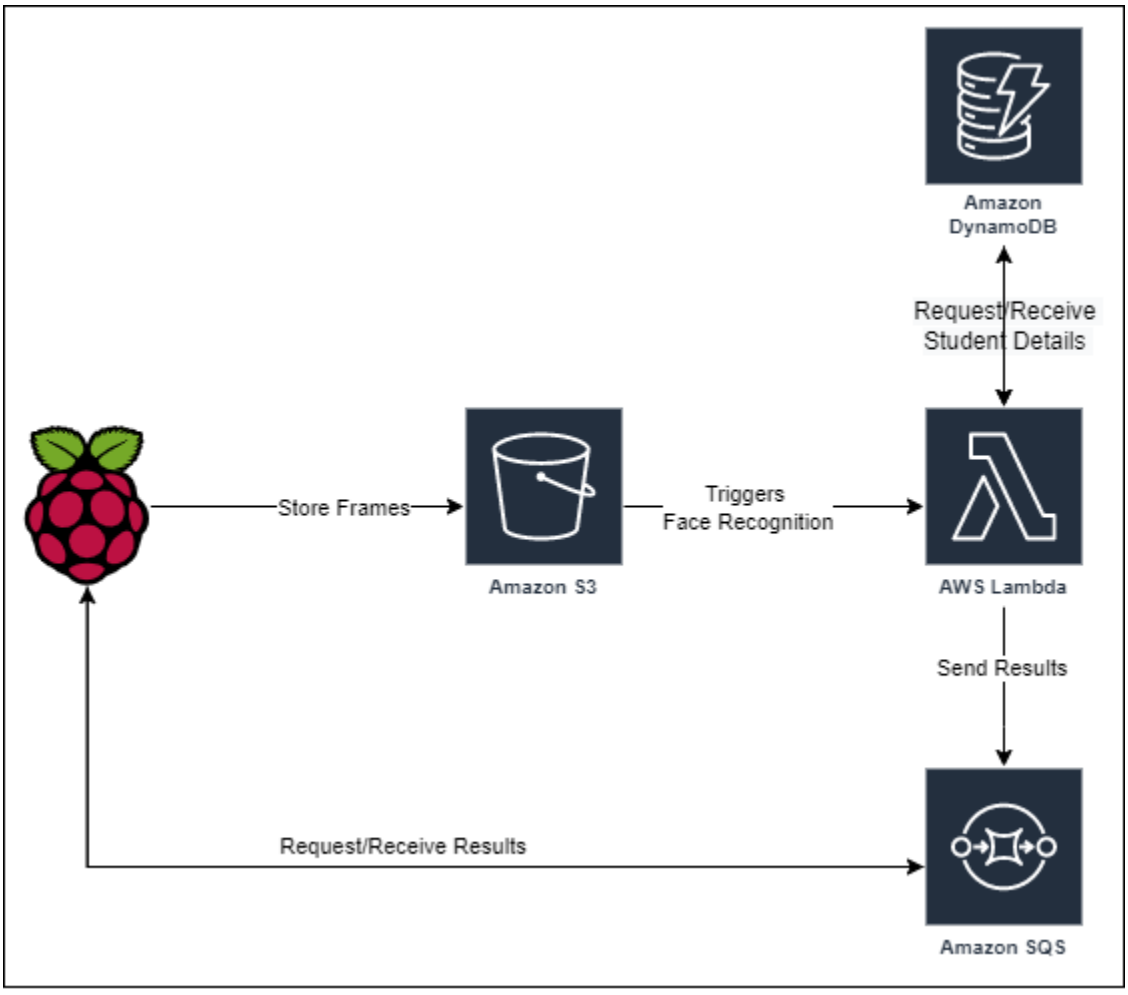
1. Problem statement

The aim is to build a distributed application that utilizes PaaS services and IoT devices to perform real-time face recognition on real videos recorded by the devices. Specifically, we developed this application using Amazon Lambda based PaaS and Raspberry Pi based IoT. Other AWS elements include a DynamoDB for storing academic details, an S3 for storing real time images coming from Raspberry Pi, SQS for queueing the results incoming from lambda function.

2. Design and implementation

2.1 Architecture

The following architectural diagram includes all major components of the system.



Components and the flow:

1) Raspberry Pi - The Raspberry Pi records video and sends 0.5 second frames to the S3 bucket.
2) AWS S3 - The S3 bucket receives 0.5 second frames and triggers the lambda on PUT operations.
3) AWS Lambda - The lambda function is created using a containerized image of the AMI that the professor provided for face recognition.
4) AWS SQS - The SQS is used to store the response coming from Lambda function about facial recognition. Raspberry Pi will take results from this queue and calculate the round trip time.
5) AWS DynamoDB - This is used to store academic related information of students. The lambda, after recognising the student will fetch details from here and upload the results to the SQS.

2.2 Concurrency and Latency

For Project 2, we use AWS Lambda which automatically scales depending on the load. We do not need to manually do any configuration to auto scale this lambda. We decoupled the system by using an SQS that stores the results and Raspberry Pi would poll for those results. We reduced the number of I/O operations on the Lambda and tried to do everything over the fly so that processing is quick per request.

3. Testing and evaluation

With the given AMI which holds the machine learning code for face recognition is dumped into an EC2 Instance which is used to train the model. Initially, we took pictures of the team members (individually) using the pi camera. Then we wrote a python code which does the resizing and format changes for the input images. Using the Python Image Library (PIL), we took the image "160 x 160" width and height. This 160 x 160 image which is .jpeg format is then converted into .png which is acceptable format for the machine learning model provided.Developed another python code using the shitil and random libraries is written to pick 5 random images to test the model accuracy.

Using 40 images of each team member in the "real_images/train" folder and 5 images in the real_images/val" folder are used to train and test the model. By executing the command : "python3 train_face_recognition.py --data_dir "data/real_images" --num_epochs 100" the model is trained with the given training data. We observed the accuracy and the loss for the training and val data. Finally, our team ended up with 66% accuracy on the training data. Using the image in the "real_images/val" model is tested. Face recognition model gave us 66% accuracy on test cases for each user.

Tested the latencies for invocations and most of the results have latency of under 2 seconds. Downloaded some of the frames randomly from S3 bucket and verified the results given by the model and it has an acceptable accuracy of more than 60 percent.

4. Code

On the Pi:

Pi can stream the video as per the requirements, in this case it is given as 5 minutes. So, the pi streams the video and also parallely uploads frames for every 0.5 seconds to the S3 bucket and also

consumes the result given by the model from an SQS. There are three main files on the Pi which are responsible for all the activities.

The first file is a requirements.txt file which has all the required python libraries for the code to execute. The second file is the frames.py file which is responsible for extracting frames of 0.5 seconds and also invokes another file called *upload_to_s3.py* which uploads the frames to S3 bucket. While invoking the *upload_to_s3.py* the number of frames that will be totally extracted is sent as a command-line argument to the program.

I implemented a class called *Frames* which has the method *recored_video*. In this method all the camera configurations are set and this extracts the frames for every 0.5 seconds and saves them in the local folder called *videos* in the *.h264* format. This method also invokes *upload_to_s3.py* as a subprocess.

We implemented a class called *UploadVideo* which has the following methods:

- *video_upload :* This is the main function of the class that is called which is responsible for generating the filename and path of the frame that has to be uploaded to the S3 bucket. The path of the frame is sent to the *upload_video_to_s3* function. The time when the frame was uploaded is stored in a dictionary which is later used to calculate the latency.
- *upload_video_to_s3:* This function takes the path of the file that has to be uploaded to S3 as input. This function reads the object into a buffer and then uploads it to S3 using the boto3 library of python.
- *fetch_result_from_sqs:* This function consumes messages from the SQS which are pushed by the AWS Lambda

Steps to execute:

- First install all the modules required for the code by running
    - *pip install -r requirements.txt*
- Create the credentials and config file for AWS access in the ~/.aws folder. This is required as we are using the boto3 library.
- Create a directory named *videos* in the home directory. This directory is used to store frames.
- In the *frames.py* file give the number of seconds we need to stream the video to *self.time_to_execute* variable.
- Give the bucket name and SQS name to *self.s3_input_bucket_name* and *self.sqs_response_queue_name* respectively.
- Run the following command to execute the program:
    - *python3 frames.py*

On AWS:

Dockerfile: This file is used to containerize the face recognition system that was provided by the professor. We follow a staged process wherein:

Stage 1: We set up the python environment. Install any ubuntu related libraries we want for the functioning of the code.

Stage 2: We then setup the working directory structure, install required python libraries such as boto3, matplotlib etc.

Stage 3: Install lambda specific entry points for the function, and run the specific part of the face recognition system that will give us results when given an input video.

eval_face_recognition.py: We added a handler function in this file to work with the trigger AWS S3 sets. First we connect to clients of S3, SQS, dynamodb for the initial setup. We then extract which uploaded file that fired this trigger and pass it to the face recognition model. We also implement the logic to extract images out of the 0.5 second frame. Eventually after getting the result from the model, we push it to a response queue from which the raspberry pi will retrieve the results.

Once you place dockerfile and the edited eval_face_recognition.py code on the AMI professor gave, we can go ahead and create the image.

I assume the system has Docker installed.

Follow command:

1) docker build -t cse546project2 .

2) aws ecr get-login-password --region us-east-1 | docker login --username AWS --password-stdin 472509191780.dkr.ecr.us-east-1.amazonaws.com

3) docker tag cse546project2:latest 472509191780.dkr.ecr.us-east-1.amazonaws.com/cse546project2:latest

4) docker push 472509191780.dkr.ecr.us-east-1.amazonaws.com/cse546project2:latest

Once all this is done the image is not uploaded/pushed to the Elastic Container Registry. You need to create the lambda using this image to get everything running. Set Lambda iam role that can access S3, DynamoDB, and SQS with full access.

Lambda to be configured to timeout at 30 seconds and 1024 Memory size.

5. Individual Contribution

Sudarshan Reddy Mallipalli - MSCS - ASU ID: 1222600444

The aim is to build a distributed application that utilizes PaaS services and IoT devices to perform real-time face recognition on real videos recorded by the devices. Specifically, we developed this application using Amazon Lambda based PaaS and Raspberry Pi based IoT.

Design:

Participated in all the group meetings in coming up with the design for the whole project. I took the whole responsibility of the RaspberryPi and got involved in setting up the Pi and accessing it using the USB Serial console and also SSH. I came up with the idea of implementing a Simple Queue Service which can be used to push the result from the model from which the result can be fetched in RaspberryPi. This will help us to give the real time latency for each and every invocation. I came up with the solution on how to implement concurrency to send the frames to S3 while live streaming. So starting a new sub-process to upload all the frames that are being extracted makes it very effective as both the programs work independently and parallely which helps in reducing the latency by very much.

Implementation:

I implemented the whole code on RaspberryPi which is also called edge. The code is responsible for extracting the frames while live streaming and uploading it to the S3. Also the code is responsible for keeping the timestamps for each and every frame that has been uploaded to S3 and also fetches the messages from Simple Queue Service then calculates the latency and gives the output. The code has been implemented very well so that there is no frame at all and this helps in face detection for every 0.5 seconds. frames.py extracts frames of 0.5 seconds and also invokes another file called *upload_to_s3.py* which uploads the frames to S3 bucket. While invoking the *upload_to_s3.py* the number of frames that will be totally extracted is sent as an command-line argument to the program.I implemented a class called *Frames* which has the method *recored_video*. In this method all the camera configurations are set and this extracts the frames for every 0.5 seconds and saves them in the local folder called *videos* in the *.h264* format. This method also invokes *upload_to_s3.py* as a subprocess.

Testing:

All the results were verified and we are getting an acceptable accuracy of 66%. Initially the local folder has been tested to check if frames are extracted for every 0.5 seconds. Then the S3 bucket has been verified to see if there is any loss of frames and it is working perfectly. Then all the results were being uploaded to SQS and consuming the messages is also working perfectly. Then finally the output results were checked randomly and the accuracy of the model is found to be more than 60% every time.

5. Individual Contribution

Venkata Pavana Kaushik Nandula - MCS - ASU ID: 1223627444.

The aim is to build a distributed application that utilizes PaaS services and IoT devices to perform real-time face recognition on real videos recorded by the devices. Specifically, we developed this application using Amazon Lambda based PaaS and Raspberry Pi based IoT.

Design:

Collaborated with the team in the weekly meeting to update the work which is assigned to me. I worked on training and testing the machine learning model. I prepared the training and testing data set for the given machine learning face recognition code. I designed the code for lambda which takes the events from S3 bucket and triggers the eval function which is given by the professor to process the images and send the results to the response queue. I also helped my teammates in building the rest of the work like setting up dynamodb. I came up with a solution of triggering the lambda by writing the handler function.

Implementation:

I implemented the training of the machine learning model on the EC2 instance with a different set of images. Training the dataset requires dividing the entire dataset into two parts, out of which one is used to train the machine learning model that was given by the professor on the AMI and the other is used to test the models. Apart from this I created the lambda function (FAAS) on AWS. In the lambda function, I developed the handler function which takes events and context as the inputs. This lambda function is triggered upon the incoming event (input from S3) which initiates the connection between the dynamodb, s3 and responseSQS. After initialization the variable declared stores the image name and the bucket name and the paths. This code is also responsible for pickling the face evaluation model file from the EC2 instance and downloading the video from the s3 bucket. Once the video is downloaded from the s3 bucket frames are extracted from the video to evaluate the face in the particular frame. The code copies the path of the frame and runs the transformation on the image and evaluates the face of the person in the current frame using the trained machine learning model. Upon the completion of face evaluation this code sends the result to dynamodb to fetch the details of the person from the information table. At last, this code pushes the results that were fetched from dynamodb.

Testing:

Initially, I tested the model by giving different images of each team member to see whether the model is giving us accurate results. Initially we ended up with 44% accuracy which was later improved to 68% by training the model with the images where they were of better quality. The handler function is tested by giving different lengths of videos to check whether the frames are getting extracted correctly. Frame extraction and sending that to eval function is rapidly tested for auto scalability and accuracy. Lambda is also tested for missing frames and data fetching from dynamodb.