CSE 546 — Project 2 Report

Harsh Sanjaykumar Nagoriya (1222471185), Dhruval Vinod Patel (1221832001), Kenil Vipulkumar Patel (1223049204)

1. Problem statement

In this global era of advanced computing, it is critical to deliver a cost-effective and reliable solution to the problem at hand. AWS, Microsoft Azure, and other cloud service providers offer similar capabilities for deploying our service (or software) on virtually endless resources. When using cloud IaaS services, we must construct and manage many instances of scalable resources ourselves. However, with Paas, cloud vendors let us just declare the activities or operations we want to do, and they take care of establishing, managing, and destroying resource instances as needed. While the device is simultaneously taking video, this project conducts real-time facial recognition. We'll utilize an AWS service called Lambda for this project, which auto-scales its instances, and an IoT device called a Raspberry Pi (with a linked camera) for live video recording and frame capturing. It will be able to launch and execute several resources at the same time in order to handle a large number of inputs as quickly as feasible. This autoscaling functionality will allow the application to make use of the cloud's capabilities in order to run efficiently and properly.

2. Design and implementation

2.1 Architecture

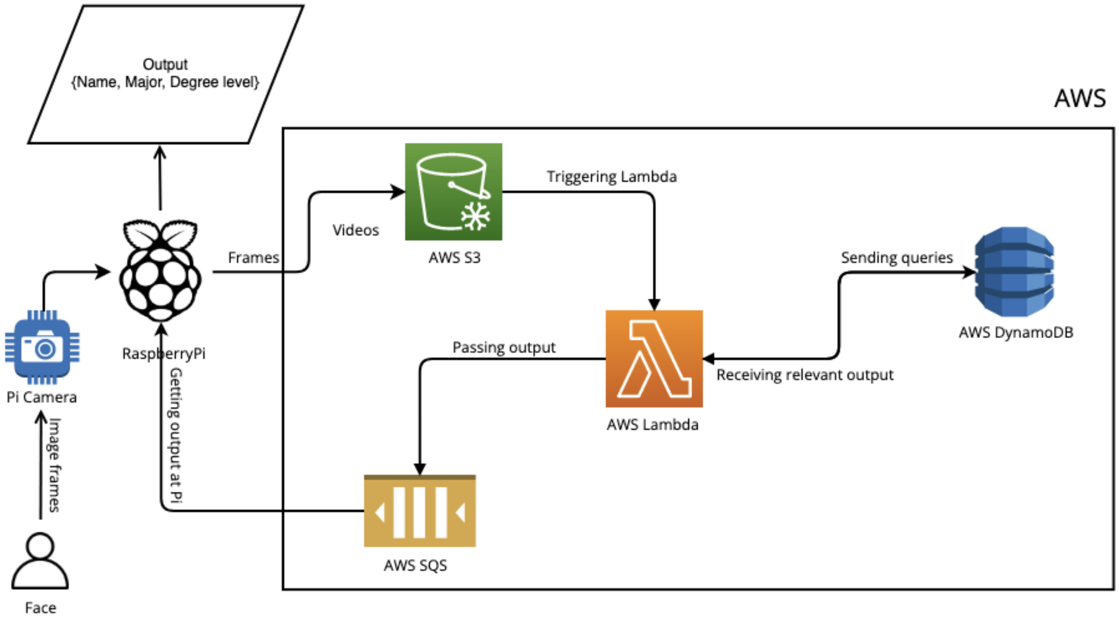

Image 1: System Architecture

2.1.1 Amazon Services

2.1.1.1 Amazon Lambda

AWS offers Amazon Lambda as one of its PaaS services. It's a stateless event-driven service that starts when a specific event happens. It also manages its instances automatically as needed.

2.1.1.2 Amazon SQS (Amazon Simple Queue Service)

Amazon SQS is a distributed messaging queuing service that allows programmatic message handling over the internet using various Amazon resources. The information (data) was passed between the web and app tiers using two queues: the request queue and the response queue. The web and app tiers are loosely connected by these queues.

2.1.1.3 Amazon S3 (Amazon Simple Storage Service)

Amazon S3 is a cloud-based object storage solution that offers scalable, permanent storage. The inputs and outputs were separated into two categories. The picture files are saved in the input bucket, while the anticipated outcomes are saved as key-value pairs in the output bucket.

2.1.2 Architecture workflow

2.1.2.1 RaspberryPI

Input: The Raspberry Pi was connected and interfaced with the Pi camera as seen in the image above. The Raspberry Pi's main role is to collect live frames using the Pi camera and deliver them to AWS storage for further processing. The photos are continually collected by RaspberryPi and uploaded to AWS S3 through the supplied internet connection.

Output: The AWS output received from SQS (at the later part of the project workflow) is sent to the RaspberryPi, which displays the findings to the user. The results were given in CLI format and included the requested information, such as name, majors, and degree level.

2.1.2.2 Amazon S3

Amazon S3 gets picture frames containing user frames from raspberryPi, as illustrated in the image above. The lambda function automatically gets invoked based on the inputs received in S3.

2.1.2.3 Amazon Lambda

To identify the face, Amazon Lambda will use the frames in S3 to run a machine learning facial recognition module. It will generate a query for dynamoDB and run it to collect necessary data after obtaining the classification result from the face recognition module.

2.1.2.4 Amazon DynamoDB

The query will be sent to Amazon DynamoDB, which is filled with student data such as name, major, and degree level. It will process the query and deliver the required data to the lambda in accordance with the query.

2.1.2.5 Amazon SQS

The dynamoDB results will be provided to Amazon SQS, which will be utilized to transmit the output to the raspberryPi.

2.2 Concurrency and Latency

Concurrency is achieved in our project by employing the following methods:

The images captured from Pi are sent using multithreading in order to get the response in place. The lambda service provided by the cloud manages the autoscaling on its own to handle the multiple requests. In turn, it invokes multiple functions on demand to achieve concurrency. The produced output is passed to the thread and received back in pi.

The time functions are used to calculate the overall system's latency. The frame capture time is recorded and saved. The time of receiving relevant output is also recorded, and the difference between the two times is used to compute total latency.

The initial latency reached was over 8 seconds, however, it quickly dropped to under 5 seconds and stayed there throughout the operation.

3. Testing and evaluation

These steps were followed to test the application:

1. On the local system, the machine learning model was tested right after it was trained with a custom data set.
2. Simultaneously, the programs for taking video from the Pi and transferring it to S3 were tested.
3. Furthermore, the cloudwatch platform was used to test the Lambda function.
4. The lambda function and the photos loaded from s3 were then tested. This stage also includes a test of the machine learning model's accuracy.
5. The entire system was recently tested when everything was in place, with photos recorded from Pi being transmitted to S3, Lambda being used to compute the result, and the results being delivered back to Pi.
6. The whole processing time, from sending inputs to getting output, was measured to determine the system's latency.

4. Code

The detailed functionality of every program is followed.

- **Files**
  - handler.py
    - This file handles the fetching of input frames from the S3 bucket and gives it to the eval_face_recognition.py file for face recognition. It also fetches the person's information from DynamoDB and sends it to the SQS queue.
  - Docker file
    - In this file, all the required files and libraries are included for creating the container image for performing face recognition using the Lambda function.
  - Student_Info.json
    - This file contains all the team members' details like Student Id, Name, Major, and Year in JSON format.
  - multithread.py
    - This file handles the capturing of videos from the Pi Camera, extracting the frames from videos, sending those frames and videos to S3, and receiving the results from the lambda function through the SQS queue.
  - eval_face_recognition.py
    - This file is accessed by the handler.py for performing the face recognition on the frames received through handler.py by loading the model weights and labels.
  - model_vggface2_best.pth
    - This file contains the machine learning model which we have trained using our face images captured from Pi Camera.
  - labels.json
    - This file contains the labels of the training and testing images included in our face recognition task.
- **Execution Flow**
  - The Dockerfile will be used to construct the container image, which will then be published to the AWS ECR. For the purpose of facial recognition, the lambda function "lambdaproc v1" will use this image.
  - Once the multithread.py file has been executed on the Pi, the video will be taken using the Pi Camera, and the frames will be extracted and uploaded to the S3 bucket. Videos will be added to the S3 bucket as well.
  - The lambda function will be invoked for each input frame after the frames are uploaded to the S3 bucket. The facial recognition function will then be executed on the input frame to identify the individual in the frame.
  - After the individual has been recognized, the details for that person will be retrieved from DynamoDB and transmitted to the SQS queue.
  - The multithread.py will fetch the results from the SQS queue in parallel (which also captures the videos from Pi Camera). On the Pi CLI, these findings will be shown.

- **Setup and execution:**
    - RaspberryPI
        - Install appropriate OS to a memory card and attach the memory card to the raspberryPi, create a user and enable remote access rights.
        - Install the Pi camera to the raspberryPi and enable the pi camera options.
        - Install python and required dependencies to the raspberryPi.
        - Run multithread.py to perform the above-mentioned operation.
    - Docker
        - Download the dockerfile and the above-mentioned files along with the other required files to construct the docker image.
        - Using dockerfile, create a container image and push it to the AWS ECR.
    - AWS Resources
        - Create S3 buckets, SQS result fetching queue and dynamoDB with student information.
    - Lambda
        - Create and set up Lambda function on AWS using the container image uploaded to AWS ECR.
        - Provide appropriate access rights for AWS resources to interact with each other.

5. Individual contribution:

5.1 Harsh Sanjaykumar Nagoriya(1222471185)

- Design:
    - It was critical to come up with the best feasible design that minimized development time and produced accurate results.
    - My role was to determine the RaspberryPI's functioning mechanism for multiple frames and video capture. My main responsibilities included dealing with the Raspberry Pi and machine learning models.
    - I also implemented all other scripts that came up throughout the project's development, such as dataset creation, jpg-png conversion, 160x160 conversion, image preprocessing, avi, mpeg encoding-decoding, and so on.

- Implementation:
    - **AWS access rights:** Each module must keep sufficient access permissions to execute additional activities. Every module we use has key access to AWS resources, which I configured and provisioned.
    - **Pi handling:** A mini-computer, the Raspberry Pi, requires numerous pre-configurations, including OS installation, user management and access privileges, camera configuration, SSH (remote computer handling), library installation, and so on. I was in charge of all of these Pi-related activities.
    - **Automation scripts and dataset collection:** At least 35 photos of each person are required for the face-recognition machine learning model. To cope with the tiresome chore of capturing several photographs for three individuals, I wrote a manual script that records the video, captures frames from it, converts the frames into three channels, and converts the image to a 160x160p.png format.
    - **Frames capturing:** The project's ultimate goal is to collect real-time frames on the Raspberry Pi and transfer them to the cloud for processing. I wrote a script that captures live frames from the Pi camera, pre-processes them, and converts them to the appropriate format. This includes picture conversion to png and video conversion to avi.
    - **Pi to S3:** To make it function in the cloud, I built a script to interface with S3 and upload the Pi recorded images to it.

- Testing
    - **Unit Testing:** Before integrating with other functioning scripts, each module I wrote had to be tested. I completed unit testing on each module/algorithm I wrote, taking into account various scenarios and conditions.
    - **Integration Testing:** After constructing several algorithm modules, we combined them all together to see if they worked properly. I devised a number of situations in my modules that may create problems when executing merged code, and I put them all to the test.

5.2 Dhruval Vinod Patel (1221832001)

Design: In the design phase, various decisions were made to properly implement the distributed application for this project which utilizes various AWS services. I designed the flow for how to trigger the lambda function whenever a frame is uploaded to S3 bucket and then how to capture the results of lambda function to display in the correct format by making use of SQS. Moreover, I designed how to implement the multithreading concept in order to simultaneously implement both 1) sending the captured videos and frames extracted from those videos to S3 and 2) to poll the SQS queue for fetching the face recognition results returned from the lambda function invocation. In order to calculate the latency for each face recognition request, I designed a TimeMap (which is a dictionary with key's as frame names and values are the upload time of that respective frame to S3 bucket) .

Implementation: I implemented all my tasks for this project using Python. I performed the following tasks:

1) Created the Docker file to build the container image by including all the required files and libraries for smooth running of the face recognition code.
2) Created the lambda handler function for handling the trigger events, running the face recognition code for the input frame, and then returning the results.
3) Created the IAM roles for Lambda function to provide it access permission to fetch the frames uploaded in S3 bucket.
4) Developed the JSON file having the team members academic information and used it to create the DynamoDB table for storing this information.
5) Fetched the details of person (recognized by lambda function) from DynamoDB and sended it to SQS queue for displaying it on the Pi CLI.
6) Created the threads to simultaneously run the video capture function and SQS message fetching function.
7) Deleted the message from the SQS result holding queue after the recognized persons results are displayed on the Pi.
8) Setup the Docker Desktop for creating the container image locally and then uploaded it to the AWS ECR using AWS CLI.

Testing: For the tasks which I implemented in this project, I conducted their testing in following 3 phases:

1) **Unit Testing:** I tested each and every function created by me for incorporating the multithreading functionality on the Pi. For the multithreading process, I tested the following functions: 1) vidCap (for capturing the videos and frames) and 2) mssgResp (for getting the face recognition results alongwith person's information from SQS).
2) **Integration Testing:** After integrating the Lambda handler functionality with face recognition module and DynamoDB, I tested the flow of our code from receiving the frames **from S3 bucket** by Lambda handler to sending it to the face recognition module for face recognition and then fetching that person's information from DynamoDB.
3) **End-to-End Testing:** We tested the entire flow of our project by checking the following things: input frame is correctly stored in S3 buckets, the lambda function is able to fetch frame from bucket, lambda function invocation, prediction results from face recognition module, fetching the information from DynamoDB and sending that information to SQS for displaying it on Pi.