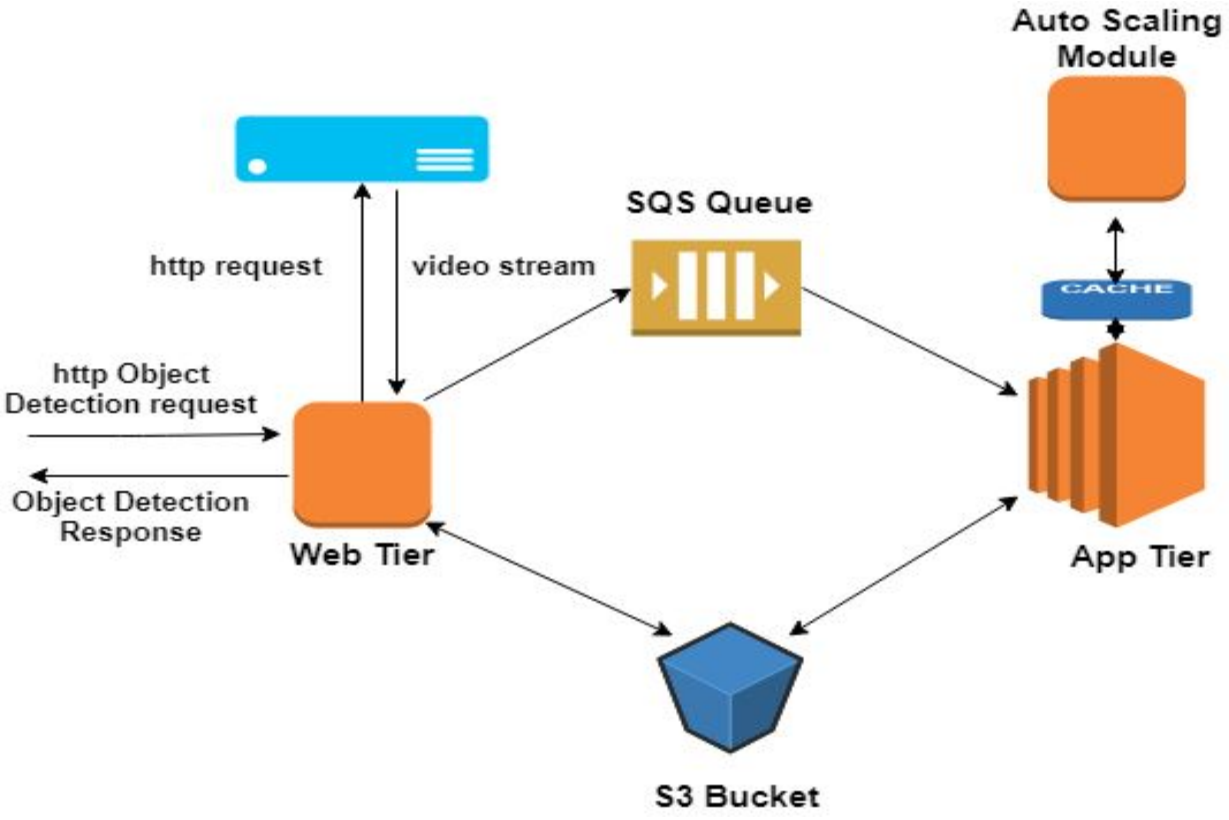
CSE 546 Cloud Computing
Project 1 - Iaas - Amazon Web Services

A video surveillance service to users using cloud resources to perform deep learning on videos collected and detect objects in real time.

Group Members:

1. Aditi Panigrahi (apanigr2@asu.edu)
2. Akash Kadam(aakadam2@asu.edu)
3. Piyush Waykole (pwaykole@asu.edu)

1. Architecture



Architecture Diagram

The system is designed to be a surveillance system. Our design exposes the Web Tier as a service to the user. The user can get object detection result from the video surveillance system by making an http GET call to the 'WebTier'.

1.1 WebTier:

Functions : The functions of the web tier include fetching the video object for a http call by a user dynamically, storing this video file in a S3 bucket and using the video file name(which is supposed to be unique but can be duplicated for requests with very little time difference) as the unique id for the user request into the SQS queue. Then return the result to user upon completion of object detection

Design : The 'WebTier' is designed to handle multiple requests concurrently using multithreading. The scenario for duplicate video file names which is observed for. We have used Springboot framework for creating the java application. The WebTier contains a ObjectDetectionController which serves the http user requests and a ObjectDetectionControllerThread which handles the functionalities asynchronously

Implementation : The 'WebTier' is the application which contains the endpoint available to the user. The requests are handled using multiple threads and only the s3 call to put the video file is synchronized to handle duplicate file names fetched for videos. Once a user request is received the request is processed asynchronously. The user

1.2 App Tier:

Functions: The main objective of the App Tier is to process the deep learning model and return the result to the S3 bucket. App Tier polls messages from SQS queue, downloads the video from S3 bucket, sends the downloaded video to the deep learning model and stores the result of that video into S3 bucket in <Key,Pair> format.

Design: The AppTier is created in Java spring boot framework and AWS SDK in java to perform AWS related operations. App tier can handle multiple messages from SQS queue concurrently.

The App Tier is mainly divided into two parts,

1. Service Thread - This part downloads the video then calls the deep learning model on the video and returns the result to the controller.

2. Service Controller - This part polls the messages from queue and creates a new service thread for each new message. This part controls the execution of the Service Threads and also puts the result of each video into S3 Bucket.

Implementation:

1. Service Thread is treated as a single unit of execution. This is implemented in program as ObjectDetectionService.java. This class extends Callable interface which effectively is used to create new threads but with a return type and used extensively with Future<> framework of java to monitor the lifecycle of the thread. This Thread internally spawns the subprocess and runs the deep learning model on the provided video and returns the result.
2. Service Controller is implemented as ObjectDetectionController.java. This controller class spawns the service threads and monitors the threads lifecycle and once execution of the thread is finished, it writes the result into S3 Bucket into <Key, Pair> form. This controller also checks whether AutoScaler instance has sent the terminated signal or not. If terminated signal is sent

by AutoScaler then no new messages are polled from SQS queue and waits for all the current executing threads to finish the execution and then shuts down the instance.

1.3 AutoScaler Tier:

Functions: The Only responsibility of the Autoscaler Tier is to respond to the number of messages in the queue and scale in/scale out the AppTier instances. AutoScaler Tier does not process any deep learning model rather it is only created to monitor messages in SQS Queue and take a decision which instances should be terminated and how many instances should be terminated by analysing various factors.

Design: AutoScaler Tier is also written in Java spring boot application framework and keeps on monitoring instances using AWS SDK for java. It contains only the logic required to scale In and scale Out the AppTier instances, i.e., only one single

Implementation:

1. AutoScaler after fixed interval times, checks message counts in SQS Queue and number of instances running. It takes into account, current messages executing, number of messages waiting in the Queue, number of instances running and how many messages each instance can accommodate and arrive at a decision that how many instances of the AppTier must be created(Scale out) or how many instances should be stopped (scale in).
2. The algorithm and design decision is explained below in the next section 'Autoscaling'

2. Autoscaling

Our application supports autoscaling by considering Web Tier and App Tier conditions and parameters. The simple logical algorithm of how auto scaling works is given below.

- Autoscaling is based on Master-slave strategy, where Master only has the responsibility of scaling out and scaling in AppTier based on the count of requests in Queue and count of running thread(consumed and still processing messages from a queue) in all running ec2 instances.
- Master will (create) scale-out *P* AppTier instances when there are *N* number of messages in Queue and each instance has a capacity of M then the number of scale-out instances *P* will be ((N - (number of instances)*M) / M). where the total number of instances cannot be more than 20.
- Master will (shutdown) scale-in *P* App Tier instances when there are *N* number of messages in Queue and each instance has a capacity of M then the number of instances need to scale-in will be (mod (N - (number of instances)*M) / M). Autoscaling master will keep at least one AppTier instance running at any point of time.
- AutoScaling model also has the logic to make sure that no instance will be terminated while still consuming the messages from a queue and having running processes by implementing a shared caching mechanism using S3 buckets.

3. Code

Web Tier :

On startup on the WebTier, it creates a request queue 'VideoObjectRequestQueue-V', where V is the version number. The WebTier is the producer for this queue and the AppTier is the consumer. The queue acts as a buffer when the number of requests increases and the AppTier is required to scale out in order to serve all the requests. The WebTier also create two buckets, one to store the video file for consumption by AppTier when it consumes the specific request from the queue and one to fetch the request from once the AppTier finishes its processing and stores the result in this bucket for the specific requests.

The WebTier contains the code for handling synchronous http calls from the user to the video surveillance system. Inside WebTier project, upon receipt of a request the WebTier Controller 'ObjectDetectionController.java' spawns a 'ObjectDetectionControllerThread' thread to handle it asynchronously.

a)The first functionality is to call the Raspberry Pi cluster to fetch the video for object detection. We have written a 'VideoService' to implement this functionality.The 'VideoService' is called for every user request. Inside 'VideoService', video request call to Raspberry Pi cluster is done in a synchronized fashion(using a lock) because if there are duplicate video files fetched in concurrent requests it can cause concurrency issues as it takes time to fetch/download the video.

b)The second functionality is to put this video file fetched into a S3 bucket. Duplicate ids are not supported by S3,so any duplicate file names will be updates. The 'UploadObject' program in the project is called for this functionality. com.cc.S3Service contains all the CRUD operations required on S3 buckets and can be used by calling the respective method.

c)Next, the 'ObjectDetectionControllerThread.java' that is handling this user request waits for minimum n seconds(where n is the average time observed for the AppTier to complete object detection). This involves checking the S3 bucket specified by the AppTier to fetch object detection results. We have set the wait time between the fetch request for every specific requests based on our testing. Wait time is introduced because we did not want to introduce the overhead of continuous get requests to S3.

d) Once the AppTier finishes its execution, after consuming the request from queue, the S3 bucket storing the results is updated and is available.

e)In the next S3 get request, the WebTier gets the result from the bucket and returns the (filename, result) key-value pair to the user synchronously. The videoFile is deleted but the result is still present with the file name is the S3 bucket. The asynchronous thread stops after the request is complete.

The properties for bucket names, queue names and properties can be changed in the application properties based on the cloud environment where the WebTier is run.

App Tier

App Tier code consists of mainly two java classes. The rest of the classes are supporting classes to these two.

1. ObjectDetectionModelService.java -

   getVideoPath() method downloads the video from S3 bucket and stores it on local drive and returns the path of the downloaded video on the disk.

Call() method creates a subprocess using ProcessBuilder() class and invokes the deep learning model and waits for the model to complete the execution, once done it returns the result.

deletVideoFile() method deletes the downloaded video on local disk.

2. ObjectDetectorController.java -

   getMsgFromQ() method polls one message at a time from Queue. This message is essentially the name of the video on which deep learning model has to be invoked.

   continuePoll() method checks if Autoscaler has allowed this instance to continue polling from queue or not. In case instance shut down is initiated by autoscaler, this method returns false. Autoscaler writes this decision of whether an instance is allowed to poll or not into an S3 bucket (S3Poll) and this method periodically keeps checking the value of the object, if the value is true then continue polling else stop the instance.

   getObject() method creates the threads of the ObjectDetectionModelService class and keeps on monitoring the life cycle. If execution of the thread is finished then writes the result into S3 bucket as <Videoname, Result> key-pairs. Once result is written into S3 bucket, this also deletes the video from S3 bucket.

Auto-Scaling Module

On startup, the auto scaling module creates the two S3 buckets required for caching mechanism explained above, to maintain the count of running threads on all running instances and to signal AppTier instance to stop consuming from the request queue.

The auto scaling module contains two different schedulers namely, to scaleOut & scaleIn and stopInstances.

a) The first scheduler gets the number of queues in the queue and has the logic to scale out or scale in the AppTier. The 'scale out' and 'scale in' schedulers are run based on the same logic(based on number of requests in queue and current running threads on the already running 'AppTier' instances).
b) Thes scheduler either runs scale out or scale in and never runs them concurrently. The 'Autoscaling.java' class contains the logic for deciding whether to scale In or scale out and also the scheduler for stopping/terminating the instances.The EC2Service.java contains the functionalities to start or stop EC2 AppTier instances when called from 'Autoscaling.java'. The 'S3Service.java' and 'SQSService.java' contains all the logic to do CRUD operations on S3 and SQS respectively.
c) The second scheduler is to stop/terminate the Instances once it is decided to scale in the specific running instances of the AppTier. This seconds scheduler is necessary because, at the time of scaling in the instances of AppTier could be still processing some requests so we need them to complete before stopping/terminating those instances. This scheduler uses the caching mechanism explained the AutoScaling section above inorder to check whether the running threads on the instances needed to be stopped have finished their processing and exited. It checks only those instances to stop which are not consuming any more new requests.

On shutdown, the Auto scaling module is designed to gracefully terminate all the running instances of AppTier.

Running the programs on AWS

You can run this system using 2 ways:

i)Using the AWS AMI's provided in README file:

a. Launch WebTier AMI
b. Launch Auto-Scaling AMI

Make sure that the WebTier has AllTraffic enabled inbound. Now your system is ready to use and will automatically scale in and out based on the number of http calls it receives from the user.

ii)Using the source code included in this submission:

a.Change the bucket names and queue names as desired in the application.properties file. Use the jars or create the jar using the source code provided in this submission.

b.Create an AWS AMI using the 'AppTier' jar and update that in application properties of Auto scaling module. Make sure the applications starts once the AMI is launched

c.Create AWS AMIs for Auto Scaling project and WebTier project and launch them.

Make sure that the WebTier has AllTraffic enabled inbound. Now your system is ready to use and will automatically scale in and out based on the number of http calls it receives from the user.

4. Project status

The project had the following specifications:

1. Implementation of Automatic Scale-Up and Scale-Down(as a separate 'AutoScaler' module in our design) : Done
2. All the inputs (video names) and outputs (detection results) should be stored in S3 for persistence. They should be stored in a bucket in the form of (input, output) pairs, e.g., ("video-1550538579.h264", "person") : Done
3. The app should handle all the requests as fast as possible, and it should not miss any request. The object detection requests should all be correct : Done
4. Duplicate requests are handled : Done