Data-intensive Computing 2024S (194.048)

Assignment 3 - Group 39

Panadero P. Raquel (12231231), Paul v. Hirschhausen (01453360), Yahya Jabary (11912007)

2024-06-28

Contents

1. Introduction

This report examines the performance and scalability differences between local and remote execution of data-intensive tasks using AWS cloud infrastructure. To this end, we developed an object detection application based on the YOLO (You Only Look Once) algorithm for identifying objects in images. The application was deployed and tested in two environments: locally, through a web service, and remotely, utilizing AWS Cloud services.

We begin with a detailed overview of the general problem, followed by an explanation of the applied methodology and approach, focusing on the distinctions between local and cloud deployment and execution. Finally, we present the resulting performance measures and draw conclusions based on the findings.

2. Problem Overview

Efficiently processing and computing object detection for large volumes of image data is a resource-intensive endeavor, particularly when employing complex algorithms such as YOLO. Local execution typically encounters limitations in processing power and scalability, which can lead to slower inference times and potential bottlenecks in performance. These constraints make it challenging to maintain efficiency and speed when dealing with large datasets.

In contrast, cloud infrastructure provides scalable and flexible resources designed to manage data-intensive tasks more effectively. The cloud's computational capabilities enable fast processing and efficient handling of extensive image data, ensuring robust performance under heavy workloads. By leveraging AWS cloud services, it is possible to achieve significant improvements in both the speed and reliability of object detection tasks, offering an efficient and scalable solution in addition to local execution.

3. Methodology and Approach

In this chapter, we outline the methodology and approach used for developing an object detection application that utilizes the YOLO algorithm to identify objects in images. The methodology is divided into two main parts: local execution and AWS cloud-based execution. The steps for each part are detailed below.

3.1. Local Execution

This section outlines the methodology employed in developing the Flask server for local execution of an object detection application using the YOLO algorithm. The server was designed to handle image uploads, perform real-time object detection, and provide system information through a series of defined REST API endpoints.

The Flask server was developed using Python, leveraging various libraries for image processing, system monitoring, and web server functionality. Key libraries included OpenCV (`cv2`) for image manipulation and YOLO integration, `numpy` for numerical operations, `psutil` for system information retrieval, `GPUtil` for GPU monitoring (if available), and `requests` for HTTP communication with the server.

The core functionality of object detection was encapsulated in the `ObjectDetection` class. This class was responsible for initializing the YOLO model the corresponding config files. The `ObjectDetection` class provided a method detect_objects to process image data received from client requests: - Input Handling: Accepting base64 encoded image data. - Image Processing: Conversion of encoded image data to OpenCV format, preparation of image blob for YOLO, and execution of object detection using the YOLO network. - Output: Detection results including object labels, confidence scores, and optionally annotated images.

The client script iteratively processes images from the specified folder, encodes them accordingly, and uploades them to the Flask server for object detection. Additionally, we perform a calculation of total and average transfer times, inference times for each uploaded image, and retrieval of comprehensive system information using a defined API endpoint (`/api/system_info`).

An schematic overview can be found in the below infrastructure concept.

3.2. AWS Cloud-based Execution

In this section, we provide a detailed description of the offloading solution's architecture and the AWS setup process. An architectural diagram is included to illustrate the overall system design and workflow.

To establish our AWS Cloud-based object detection service, we leverage the Boto3 API extensively for all operations, eliminating any manual use of the AWS Management Console for component creation.

The process begins with the creation of a DynamoDB table and an S3 bucket using Boto3. For integrating Lambda with the S3 bucket, we first develop the Lambda function code locally, ensuring it includes all necessary dependencies. Once the function is ready, we package these dependencies into a ZIP file and upload it to the previously created S3 bucket using Boto3. We also upload the YOLO config files.

Next, we employ the Boto3 API to create a Lambda Layer that encapsulates these dependencies. This approach is chosen for two primary reasons: first, it separates the core function logic from its dependencies, promoting cleaner and more maintainable code; second, it allows these dependencies to be shared across multiple functions, facilitating potential future expansions of our service.

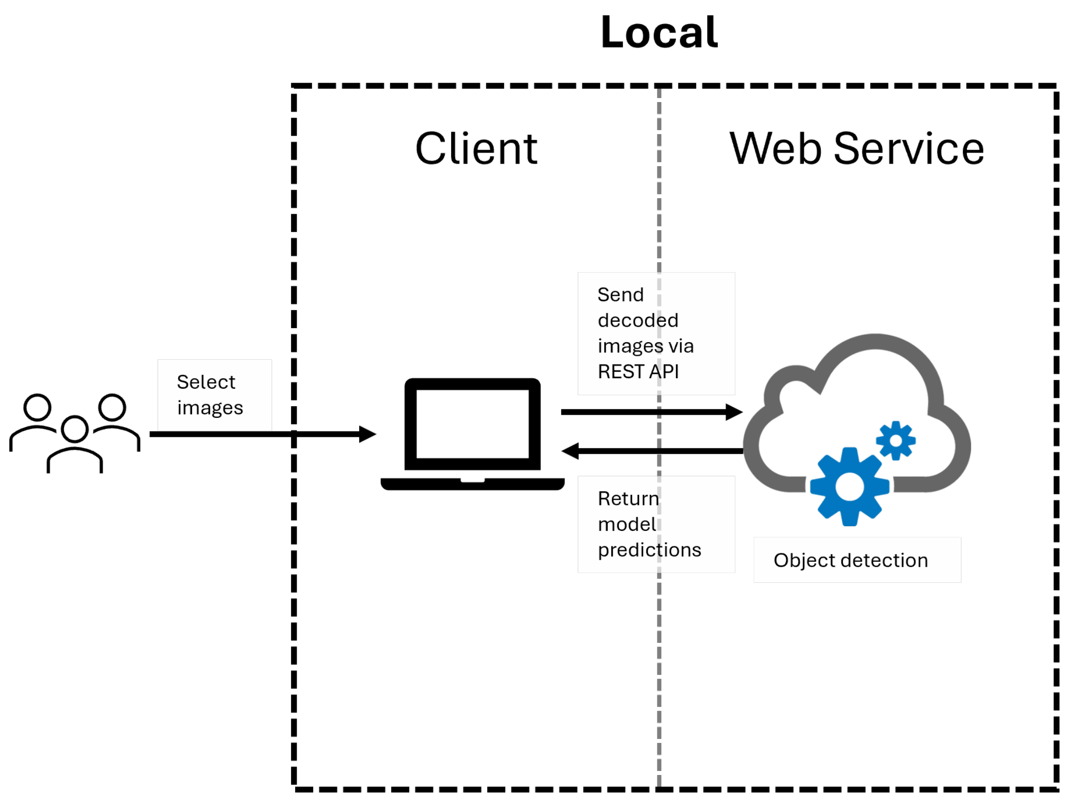

Figure 1: Local Client Server Architecture for Object Detection

Finally, using Boto3, we establish a Lambda function that is linked to both the S3 bucket for event triggers and the Lambda Layer containing the dependencies. This setup ensures that our Lambda function is equipped with the necessary resources and dependencies to perform object detection seamlessly whenever new images are uploaded to the S3 bucket.

The Lambda function first checks if the necessary YOLO configuration files and the uploaded image are present locally in the temporary (`tmp`) folder. If these files are not already available locally, the Lambda function downloads them from the S3 bucket that triggered the function. Subsequently, the uploaded image is processed for object detection. The detected objects along with relevant metadata are then stored as an item in the DynamoDB table associated with the application.

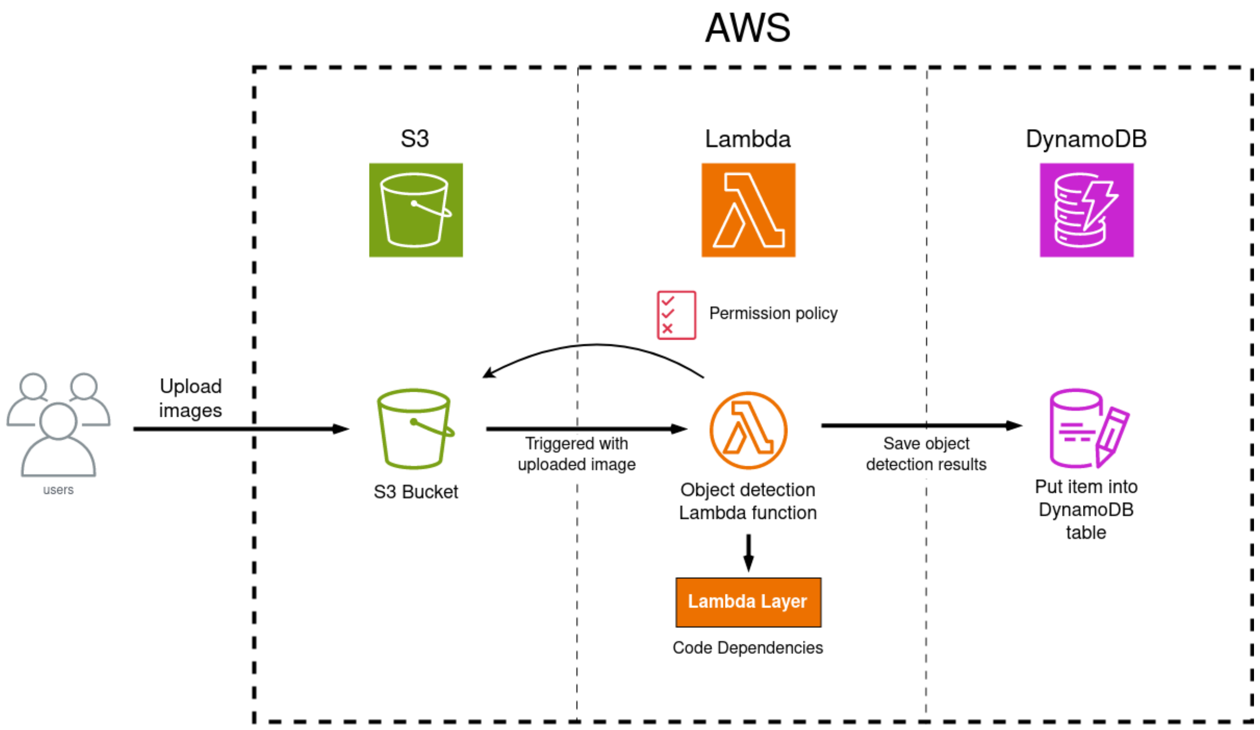
AWS
S3
Lambda
DynamoDB
Permission policy
users
Upload images
S3 Bucket
Triggered with uploaded image
Object detection Lambda function
Save object detection results
Put item into DynamoDB table
Lambda Layer
Code Dependencies

Figure 2: AWS Architecture for Object Detection

4. Results

In this final chapter, we will explore the different deployment approaches and their respective performance outcomes. Our experiment utilized 100 images, which were either sent via a REST call to the web service in the local deployment approach or uploaded to an S3 bucket for the cloud deployment.

From the performance metrics shown in the table below, it is evident that the local web service deployment is significantly faster, with a performance boost of 2-3 times compared to the cloud deployment.

Local and Cloud execution results:

Metric	Client Server	AWS
Total Images Processed	100	100
Total Transfer Time	25.3988 seconds	64.0640 seconds
Average Transfer Time	0.2540 seconds	0.6406 seconds
Average Inference Time	0.0939 seconds	0.7637 seconds

Metric	Client Server	AWS
CPU Usage	12.4%	not applicable
Current CPU Frequency	1003.7976 MHz	not applicable
Max CPU Frequency	3600.0 MHz	not applicable
Min CPU Frequency	400.0 MHz	not applicable
Physical Cores	4	not applicable
Total Cores	8	not applicable

The superior performance of the local deployment can be attributed to several factors, possibly including the limited number of images (100) used in the experiment. With fewer images, the overhead associated with data transfer and network latency in the cloud deployment becomes more pronounced, highlighting the local server's efficiency.

When considering cloud versus local execution, both approaches have their advantages and disadvantages. Local execution typically offers lower latency and faster processing times, especially when dealing with smaller datasets, due to the elimination of network-related delays. It also provides greater control over hardware and environment configurations, which can be optimized for specific tasks. On the other hand, cloud execution offers superior scalability and flexibility, allowing for easy adjustments to compute resources based on demand. In this example, AWS would handle the scaling of compute resources, enabling the application to scale up or down as needed.

5. Conclusions

This report analyzed the performance and scalability differences between local and AWS cloud-based execution of data-intensive tasks using an object detection application based on the YOLO algorithm. Testing with 100 images revealed that local deployment significantly outperforms cloud deployment, with a 2-3 times performance boost due to lower latency and reduced data transfer overhead.

However, AWS cloud infrastructure offers superior scalability and flexibility, essential for handling larger datasets and variable workloads. While local execution is advantageous for smaller datasets with immediate processing needs, cloud deployment is ideal for extensive and dynamic tasks requiring scalable resources.