Fog Computing-Based Object Detection Project

Project Overview:

Our project is designed to address the need for real-time object detection in images within a distributed fog computing system. This system enables efficient and parallel processing of incoming images from multiple clients while ensuring fast response times.

Project Objectives:

- Real-time object detection
- Utilizing fog computing
- Scalability and parallel processing

Our project is designed to address the need for real-time object detection in images within a distributed fog computing system. This system enables efficient and parallel processing of incoming images from multiple clients while ensuring fast response times.

Technologies Used:

- Python
- Flask-SocketIO
- Ultralytics YOLOv8 model for objects detection
- Socket programming
- Threading and parallelism
- Networking and communication

How It Works ?

Step 1 : Image Upload

- Clients, using Flutter applications, upload images to our central server. These images can be of various types and sizes, and clients can upload them concurrently.

Step 2 : Server

- The server acts as the central hub of our system. It listens for incoming connections from clients, and upon receiving an image, it handles the following tasks:
    1. Receives and stores the image.
    2. Distributes the image to multiple fog nodes for object detection.
    3. Sends the prediction result back to the client.

Step 3: Fog Nodes

- Our fog nodes are distributed computing units each listening on a socket, and each running the powerful YOLOv8 model for real-time object detection. These nodes are continuously available and responsive to new tasks.
- When a server sends an image to a fog node, it processes the image, detecting objects within it, and then returns the results to the server.

Step 4: Parallel Processing

- One of the key strengths of our system is parallel processing. As multiple clients upload images, the server can send these images to multiple fog nodes simultaneously. This means that different images can be processed at the same time on different fog nodes.

Step 5: Results

- As fog nodes complete object detection tasks, they send the results back to the server. The server accumulates the results as they arrive.

Step 6: Feedback to Clients

- Finally, the server sends the object detection results back to the clients who uploaded the images. Clients receive real-time feedback, allowing them to take further actions based on the detected objects.

Implementation Overview:

1. ### *Server Code Overview:*
   The initial section of the server code imports the necessary Python modules and libraries:

```
import os
import socket
import concurrent.futures
import threading
import pickle
import datetime
```

   - os (Operating System Interface): Provides functions for interacting with the operating system, including file and directory operations.
   - socket (Socket Programming): Enables network communication by creating and managing network sockets for sending and receiving data.
   - concurrent.futures: Part of the Python standard library and used for working with asynchronous tasks, such as thread and process pools.
   - threading (Threading): Provides a way to create and manage threads, allowing for concurrent execution of code within a single process.

- pickle (Object Serialization): A module for serializing and deserializing Python objects, used for data storage and communication between processes.
- datetime (Date and Time Operations): Offers classes and functions to work with dates, times, and time intervals, including formatting and parsing date and time values.

This part of the code sets up various global variables and configuration parameters:

```
9   UPLOADS_FOLDER = 'uploads/'  # Specify the folder
10  # Create a dictionary to store locks for each imag
11  image_locks = {}
12  image_list = []
13  node_availability = [True]*2
14  imagesSent={}
15  predictions = {}
16  imageConnection={}
17  counter=0
18  counter_lock = threading.Lock()  # Create a lock
19  node_availability_locks = threading.Lock()
```

- ***UPLOADS_FOLDER*** defines the folder where received images are stored.
- A dictionary, image_locks, is used to store locks for each image, ensuring that images are processed and deleted in a thread-safe manner.
- ***image_list*** is a list to maintain received images waiting to be processed.
- ***node_availability*** is a list to track the availability of fog nodes.
- ***imagesSent*** is used to associate images with the fog node to which they are sent.
- ***predictions*** stores the prediction results for images.
- ***imageConnection*** associates each image with the connection from which it was received.
- ***count*** is a global variable used to avoid image storage with the same name by two threads.
- ***counter_lock*** is the lock used to let access only by one thread.
- ***node_availability_locks*** is the lock used to avoid two threads uses the same node at the same time.

This part of the code specify the different functions implemented :

```
57  def run_app():
58      with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:
59          server_socket.bind(('192.168.1.18', 100))
60          server_socket.listen()
61
62          print(f"Server listening on {'192.168.1.18'}:{100}")
63
64          while True:
65              connection, address = server_socket.accept()
66              threading.Thread(target=handle_connection, args=(connection, address)).start()
67
```

- This function is responsible for listening to client connections and launching a new thread to handle each incoming connection.
- It binds the server socket to a specified IP address and port and listens for incoming connections.

```python
def handle_connection(connection, address):
    print(f"Connected to {address}")
    image_data = b''
    receiving = False

    while True:
        data = connection.recv(1024)
        if not data:
            break
        if b'ImageStart' in data:
            receiving = True
            data = data.replace(b'ImageStart', b'')
        if b'ImageEnd' in data:
            receiving = False
            data = data.replace(b'ImageEnd', b'')
            image_data += data
            # Generate a unique filename for each image
            timestamp = datetime.datetime.now().strftime("%Y%m%d_%H%M%S")
            global counter
            image_path = os.path.join(UPLOADS_FOLDER, f'received_image_{timestamp}_{counter}.jpg')
            with counter_lock:
                counter += 1
            with open(image_path, 'wb') as image_file:
                image_file.write(image_data)
            image_list.append(image_path)
            imageConnection[image_path]=connection
            # Clear the image data after saving the image
            image_data = b''
        if receiving:
            image_data += data
```

- This function is executed in a separate thread for each client connection.
- It receives image data from clients and saves it to a file with the use of the ***counter_lock*** to make sure of a unique image name.
- maintains the ***image_list*** for images that are ready for processing and updates ***imageConnection*** to keep track of which connection is associated with each image.

```python
def process_images():
    num_nodes = 2
    node_addresses = [("192.168.1.18", 200), ("192.168.1.18", 300)]

    with concurrent.futures.ThreadPoolExecutor(max_workers=num_nodes) as executor:
        futures = []
        print(image_list)
        while True:
            for j in range(num_nodes):
                if not image_list:
                    break
                with node_availability_locks:
                    if node_availability[j]:
                        image_path = image_list.pop(0)
                        imagesSent[image_path] = j
            # Create a new lock for this image
                        image_locks[image_path] = threading.Lock()
                        node_availability[j] = False

                        future = executor.submit(send_image_to_node, node_addresses[j], image_path)
                        futures.append((future, image_path))

            for future, image_path in futures:
                if future.done():
                    response = future.result()
                    node_index = imagesSent[image_path]
                    print(f"Objects detected by Node {node_index+1}: {response}")
                    futures.remove((future, image_path))
                    node_availability[node_index] = True
                    predictions[image_path] = response
            # Acquire the lock before deleting the image
                    with image_locks[image_path]:
                        if os.path.exists(image_path):
                            os.remove(image_path)
                # Send acknowledgment
                        if image_path in imageConnection:
                            imageConnection[image_path].send(bytes('\n'.join(response), 'utf-8'))
                            del imageConnection[image_path]
```

- This function manages the distribution of images to available fog nodes using a thread pool.
- It sends images to nodes as they become available and we use ***node_availability_locks*** lock to avoid two threads using the same node, and then waits for results. results are stored in ***futures*** List: which is a container that holds future objects, each representing an ongoing asynchronous task for sending images to fog nodes and receiving predictions.
- It uses the ***image_locks*** dictionary to ensure that images are processed and deleted in a thread-safe manner.
- When results are received, As futures in the futures list complete, their results are received by checking their status, allowing the system to collect and process the predictions made by fog nodes. and then predictions are sent back to the client and the image is deleted from the server

```python
def send_image_to_node(node_address, image_data):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.connect(node_address)
        s.sendall(b'ImageStart')  # Start the image transmission
        s.sendall(image_data)  # Send the image data
        s.sendall(b'ImageEnd')  # End the image transmission
        response = s.recv(1024)
        return pickle.loads(response)
```

- This function is responsible for establishing a socket connection with a fog node and sending an image path for processing.
- It sends the image data to the fog node, waits for a response, and returns the prediction results.

```python
if __name__ == "__main__":
    threading.Thread(target=run_app).start()
    process_images()
```

- In the main block, the script starts the run_app function in a separate thread to listen for client connections.
- It then calls the process_images function to manage image distribution and processing.

.

2- Node Code Overview:

```python
import socket
import pickle
from ultralytics import YOLO
```

- Utilize the Ultralytics YOLOv8 model for object detection. YOLO (You Only Look Once) is a popular deep learning model for real-time object detection in images.

```python
def object_detection(image_data):
    model = YOLO("yolov8m.pt")
    results = model.predict(image_data)
    result = results[0]

    detected_objects = []
    for obj in result.boxes:
        class_id = obj.cls[0].item()
        detected_objects.append(result.names[class_id])
    return detected_objects
```

- This function is responsible for performing object detection using the YOLOv8 model.
- It takes image data as input and returns a list of detected objects found in the image.

```python
def node_function(node_address):
    # Create a socket to listen for image data from the server
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.bind(node_address)
        s.listen()
        print("Listening on address:", node_address[0], "PORT:", node_address[1])

        while True:
            conn, addr = s.accept()
            with conn:
                image_data = b''
                receiving = False
                while True:
                    data = conn.recv(1024)
                    if not data:
                        break
                    if b'ImageStart' in data:
                        receiving = True
                        data = data.replace(b'ImageStart', b'')
                    if b'ImageEnd' in data:
                        receiving = False
                        data = data.replace(b'ImageEnd', b'')
                        image_data += data

                        # Convert received image data to a PIL Image object
                        image = Image.open(BytesIO(image_data))

                        # Perform object detection
                        detected_objects = object_detection(image)

                        # Send detected objects back to the server
                        conn.sendall(pickle.dumps(detected_objects))
                        break
                    if receiving:
                        image_data += data
```

- This function encapsulates the main functionality of the fog node.

- It listens for incoming connections on a specific IP address and port, receives image data from the server, and performs object detection using the YOLOv8 model.

```python
if __name__ == "__main__":
    # Define the node's IP and port
    node_address = ("192.168.1.18", 200)

    # Start the node
    node_function(node_address)
```

- In the main block of the fog node code, we specify the node's IP address and port to listen for connections.
- Then we start the ***node_function***, which listens for incoming image data from the server, processes it, and sends the detected objects back to the server.

3- Flutter Code Overview:

```dart
Future getImage(bool isCamera) async {
  final pickedFile = await picker.pickImage(
      source: isCamera ? ImageSource.camera : ImageSource.gallery);

  setState(() {
    if (pickedFile != null) {
      image = File(pickedFile.path);
    } else {
      print('No image selected.');
    }
  });
}
```

- This function is responsible for capturing or selecting an image. It uses the ImagePicker plugin to access the device's camera or image gallery based on the ***isCamera*** parameter.
- The selected image is stored in the ***image*** variable, and the user interface is updated accordingly.

```dart
Future sendImageToServer() async {
  if (image == null) return;

  try {
    socket = await Socket.connect('192.168.1.18', 100);
    listenForMessages();
  } catch (e) {
    print(e.toString());
  }
  if (image == null) {
    print('No image selected');
    return;
  }

  final File imageFile = File(image!.path);
  final List<int> imageBytes = await imageFile.readAsBytes();

  socket?.add(utf8.encode('ImageStart'));
  socket?.add(imageBytes);
  socket?.add(utf8.encode('ImageEnd'));
}
```

- This function is used to send the selected image to the server. It establishes a socket connection to the server at the specified IP address and port.
- It converts the image file into bytes, adds delimiters ('ImageStart' and 'ImageEnd'), and sends the image data to the server through the socket connection.

```
58  void listenForMessages() {
59    socket?.listen((List<int> event) {
60      final message = String.fromCharCodes(event);
61      setState(() {
62        acknowledgmentMessage = message;
63      });
64    });
65  }
66
```

- This function sets up a listener on the socket connection. It listens for incoming messages from the server.
- When a message is received, it converts the byte data into a string and updates the acknowledgmentMessage variable, which can be displayed to the user to show the acknowledgment or response from the server.

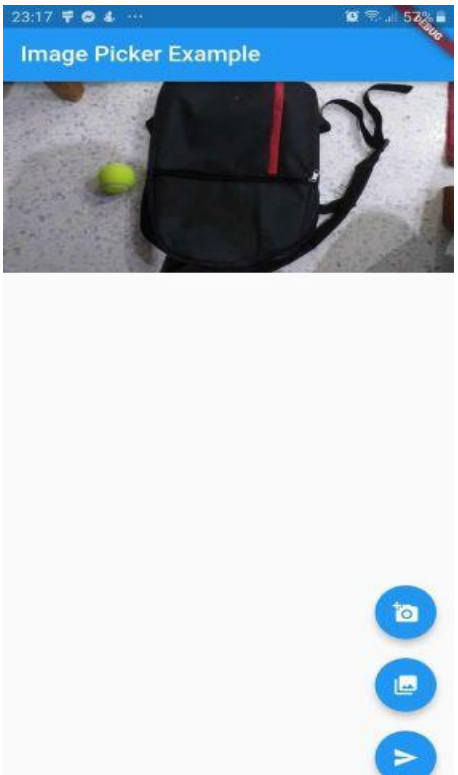
Benefits:

- ***Real-time object detection***: Clients receive results in near real-time.
- ***Scalability***: Our system can efficiently handle a large number of clients and images.
- ***Efficient parallel processing***: Multiple images can be processed simultaneously on multiple fog nodes, ensuring a responsive system.
- ***Optimized resource usage***: Images are removed from the server once they are successfully predicted and sent back to the client, ensuring efficient use of server storage resources.

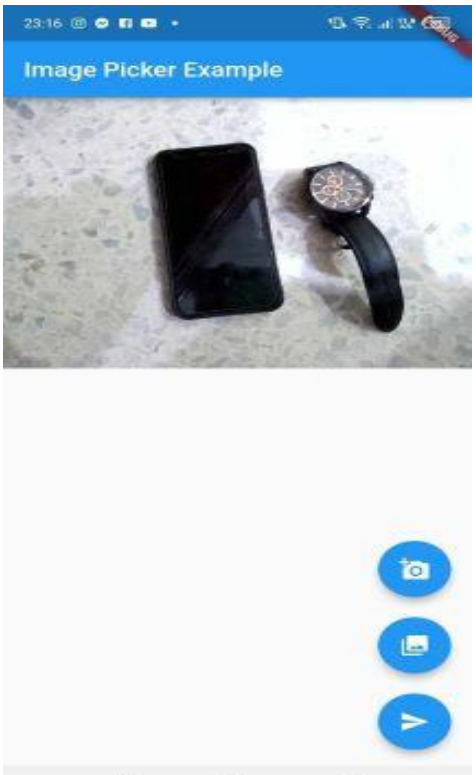
Simulation:

We will use 3 smartphones to act each as a client and 2 Fog Nodes and The Server:

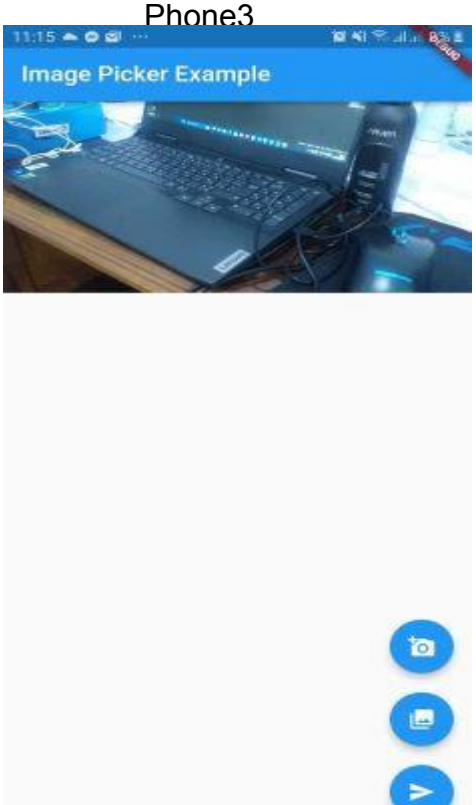
Phone1



Phone2



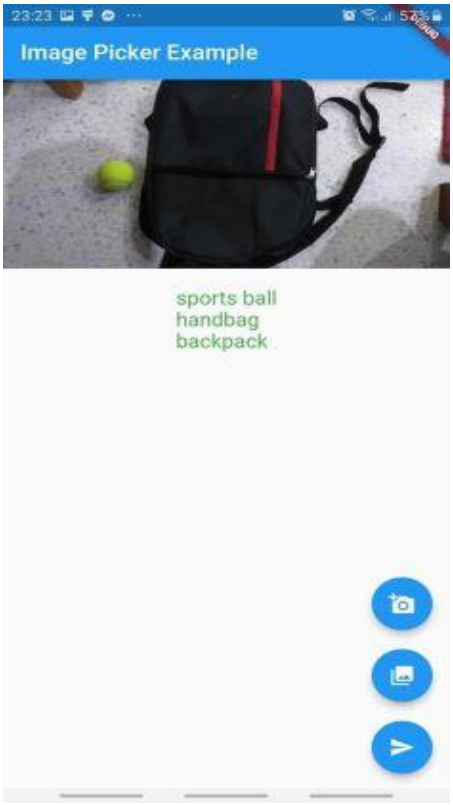
Phone3



- Like we see each client has captured an image and will now send it to the server by pressing the send button and it will wait for the prediction.
- Image Processing will happen simultaneously.

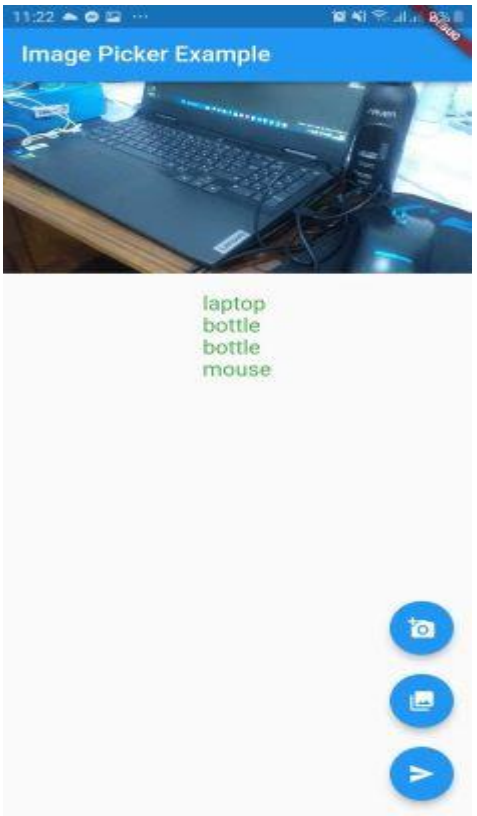
➔ After Processing is Complete👍

Prediction for client 1



Prediction for client 2

Prediction for client3



Conclusion: Like we can see for each client: Image objects are detected by the Fog Nodes and sent to the client.

Time Gain:

In the example of simulation when there are 3 clients and 2 Fog nodes:

-Using parallel Processing: Time to process the three images is 8 seconds.

-***Using One node(no parallel processing):***Time to process the three images is 15 seconds.

Video Explaining the work of Server and Fog Nodes:

Watch The video Sent separately.