UCD School of Electrical & Electronic Engineering

COMP40660

Advances in Wireless Networking

Assignment 2

Edge Computing (Group project)

Student 1: Sheng Wang	Student Number: 20403764
Student 2: Tianru Ji	Student Number: 24208429
Student 3: Bingxin Li	Student Number: 20205615

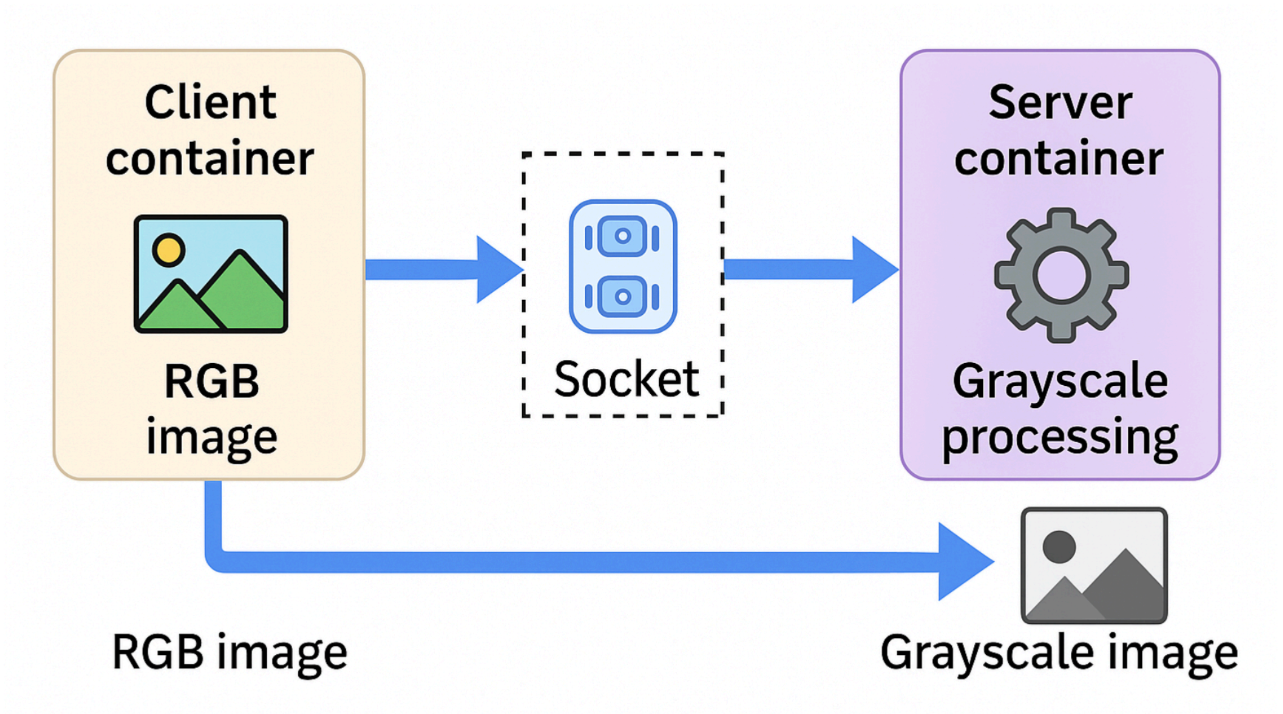
Date: April 27, 2025

Assignment2-Design document

1. Introduction and System Overview

In modern computing systems, lightweight devices such as edge devices often face limitations in computational power and bandwidth. To address these constraints, offloading is used to move complex computational tasks to a nearby server for processing.
This project implements a lightweight offloading system using Docker networking. Two Ubuntu containers are deployed: the client container generates a 32 × 32 RGB image and transmits it to the server container over an IPC channel. The server container performs a grayscale conversion on the received image and sends the processed grayscale image back to the client.
By simulating this offloading behavior between containers, this project reflects typical future application scenarios where edge devices delegate computationally intensive tasks to more powerful servers to enhance the system efficiency and performance.

2. System Architecture



The system consists of two Ubuntu containers connected via Docker's default bridge network.

Client container

- Sends the raw image data to the server container through a socket conneciton.
  Server container

- Listens for incoming connections at a specific port (e.g. 9898).
- Receives the RGB image and processes it to a grayscale image.
- Sends the resulting 32 $\times$ 32 grayscale image back to the client.
  This architecture separates data preparation and heavy computation, reflecting a typical offloading scenario in future edge computing systems.

3. Implementation

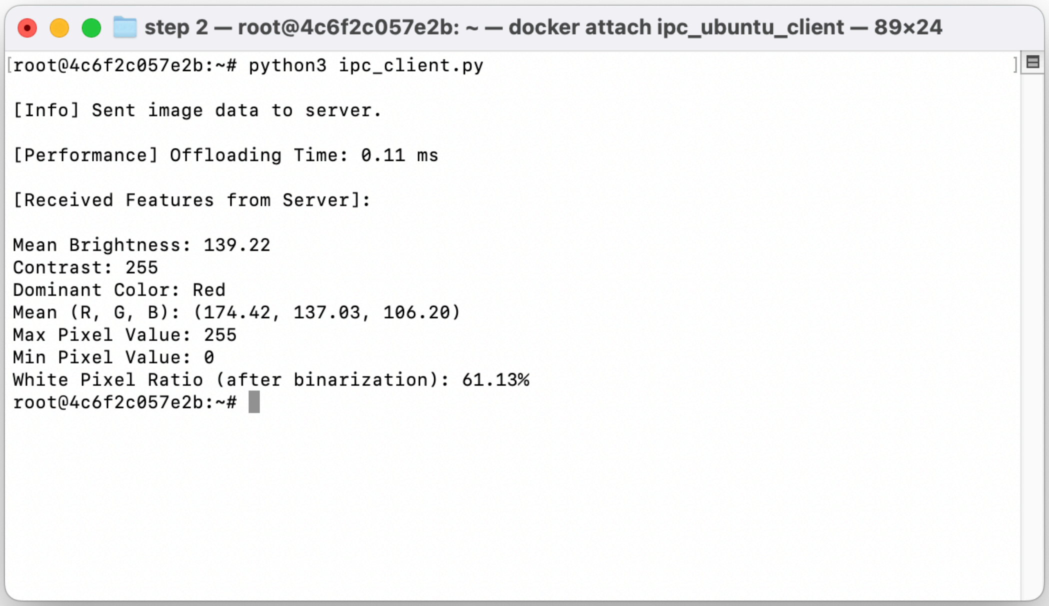
3.1 Container Setup

Two Ubuntu containers are created to simulate a lightweight client and a more powerful server environment.
The containers are connected via Docker's default bridge network, allowing internal communication through private IP addresses.

3.2 Communication Scripts

There are two Python scripts: **ipc_client.py** and **ipc_server.py**.
ipc_client.py is responsible for:

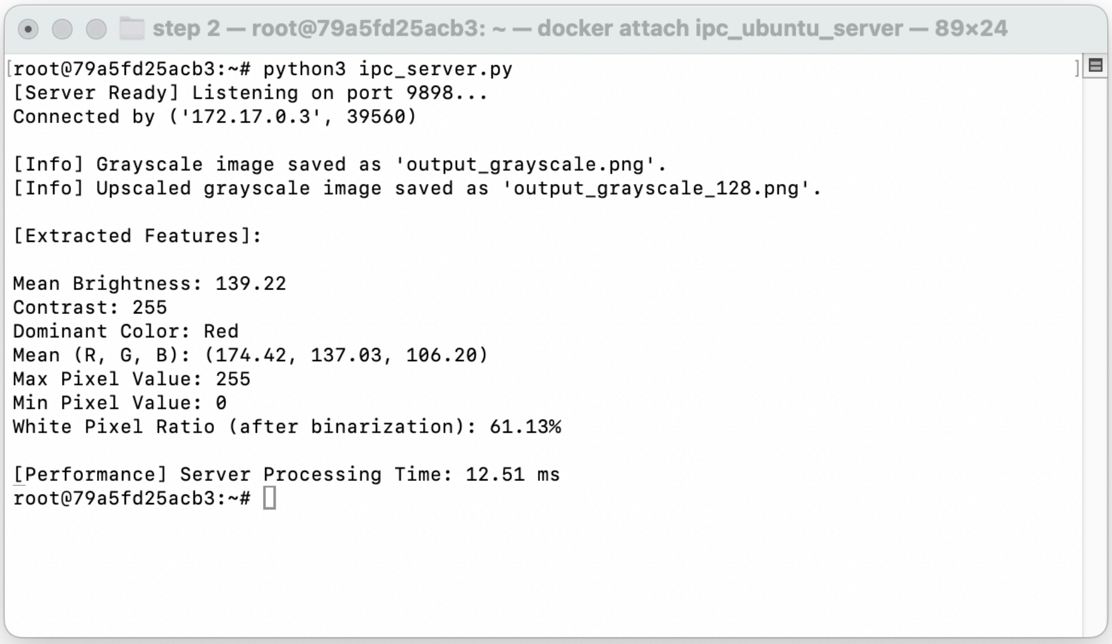
- **Image Preparation**: Loads a local image file (eevee.png) and converts the image into a list format suitable for JSON serialization.
- **Data Transmission**: Establishes a TCP socket connection to the server container using the server's private IP and port. Then sends the data.
- **Performance Measurement**: Measures the **offloading time** (the time taken to send the image).
- **Receiving Response**: Waits for and receives the processed results from the server.
  The result is shown in the following figure:


step 2 — root@4c6f2c057e2b: ~ — docker attach ipc_ubuntu_client — 89×24

```
[root@4c6f2c057e2b:~# python3 ipc_client.py

[Info] Sent image data to server.

[Performance] Offloading Time: 0.11 ms

[Received Features from Server]:

Mean Brightness: 139.22
Contrast: 255
Dominant Color: Red
Mean (R, G, B): (174.42, 137.03, 106.20)
Max Pixel Value: 255
Min Pixel Value: 0
White Pixel Ratio (after binarization): 61.13%
root@4c6f2c057e2b:~#
```

ipc_server.py is responsible for:

- **Data Reception**: Receives the full serialized image data from the client.
- **Image Processing**: Decodes the received JSON data into a 32 × 32 RGB image array. Then converts the RGB image to a grayscale image and saves it as **output_grayscale.png**. Also saves an upscaled 128 × 128 grayscale version as **output_grayscale_128.png**.
  Feature Extraction:
  Calculates **mean brightness** and **contrast**.
  Identifies the **dominant color** (Red, Green, or Blue).
  Computes **mean values for R, G, B channels**.
  Determines the **white pixel ratio** after binarization.
  - White Pixel Ratio Calculation: After converting the grayscale image into a binarized image (where each pixel is either 0 for black or 255 for white based on a threshold of 127), the server calculates the percentage of white pixels relative to the total number of pixels. It helps characterize the brightness distribution of the processed image.
- **Performance Measurement**: Measures the server-side image processing time.
- **Sending Results**: Sends the processed image back to the client container as a response.


step 2 — root@79a5fd25acb3: ~ — docker attach ipc_ubuntu_server — 89×24

```
root@79a5fd25acb3:~# python3 ipc_server.py
[Server Ready] Listening on port 9898...
Connected by ('172.17.0.3', 39560)

[Info] Grayscale image saved as 'output_grayscale.png'.
[Info] Upscaled grayscale image saved as 'output_grayscale_128.png'.

[Extracted Features]:

Mean Brightness: 139.22
Contrast: 255
Dominant Color: Red
Mean (R, G, B): (174.42, 137.03, 106.20)
Max Pixel Value: 255
Min Pixel Value: 0
White Pixel Ratio (after binarization): 61.13%

[Performance] Server Processing Time: 12.51 ms
root@79a5fd25acb3:~# 
```

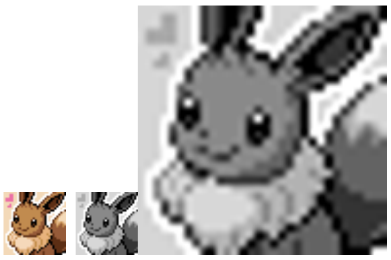
3.3 Offloading Process

- Upon receiving the RGB image, the server processes it by applying a weighted grayscale conversion formula.
- The processed grayscale image is then sent back to the client container through the same socket connection.

4. Results and Conclusion

The server container successfully received the original 32×32 RGB image sent from the client. After processing, the server converted the image into a 32×32 grayscale image, and a 128×128 grayscale version output. The processed grayscale image is then sent back to the client container, completing the full offloading cycle.

The original image and the output:



Through this, we demonstrate the lightweight offloading system, based on Docker container networking and socket communication.

The link to the exported docker hub image are:
`https://hub.docker.com/r/izlx/awn_assign2_task2_ipc_server`
`https://hub.docker.com/r/izlx/awn_assign2_task2_ipc_client`

Here are the python scripts that used in this offloading scenario:

```python
#!/usr/bin/env python3
# ipc_server.py

import numpy as np
import json
import socket
import struct
import time
from PIL import Image

HOST = '0.0.0.0'
PORT = 9898

def recv_all(sock, length):
    data = b''
    while len(data) < length:
        more = sock.recv(length - len(data))
        if not more:
            raise EOFError(f"Expected {length} bytes but received {len(
    data)} bytes before socket closed.")
        data += more
    return data

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    s.bind((HOST, PORT))
    s.listen()
    print("[Server Ready] Listening on port 9898...")

    conn, addr = s.accept()

    with conn:
        print('Connected by', addr)

        raw_msglen = recv_all(conn, 4)
        msglen = struct.unpack('>I', raw_msglen)[0]

        full_data = recv_all(conn, msglen)

        image = json.loads(full_data.decode('utf-8'))
        image_array = np.array(image)  # (32, 32, 3)

        start_proc_time = time.time()

        R = image_array[:, :, 0]
        G = image_array[:, :, 1]
        B = image_array[:, :, 2]

        mean_brightness = np.mean(image_array)
        contrast = np.max(image_array) - np.min(image_array)

        mean_R = np.mean(R)
        mean_G = np.mean(G)
        mean_B = np.mean(B)

        dominant_color = 'Red' if mean_R > mean_G and mean_R > mean_B else \
                         'Green' if mean_G > mean_R and mean_G > mean_B else 'Blue'

        grayscale = np.mean(image_array, axis=2)
        binarized = np.where(grayscale > 127, 255, 0)
        white_pixels = np.sum(binarized == 255)
        total_pixels = binarized.size
        white_pixel_ratio = (white_pixels / total_pixels) * 100

        grayscale_image = Image.fromarray(grayscale.astype(np.uint8))
        grayscale_image.save('output_grayscale.png')
        print("\n[Info] Grayscale image saved as 'output_grayscale.png'.")

        grayscale_big = grayscale_image.resize((128, 128), Image.BICUBIC)
        grayscale_big.save('output_grayscale_128.png')
        print("[Info] Upscaled grayscale image saved as 'output_grayscale_128.png'.")

        response = (
            f"Mean Brightness: {mean_brightness:.2f}\n"
            f"Contrast: {contrast}\n"
            f"Dominant Color: {dominant_color}\n"
            f"Mean (R, G, B): ({mean_R:.2f}, {mean_G:.2f}, {mean_B:.2f})\n"
            f"Max Pixel Value: {np.max(image_array)}\n"
            f"Min Pixel Value: {np.min(image_array)}\n"
            f"White Pixel Ratio (after binarization): {white_pixel_ratio:.2f}%"
        )

        print("\n[Extracted Features]:\n")
        print(response)

        end_proc_time = time.time()

        processing_time = end_proc_time - start_proc_time
        print(f"\n[Performance] Server Processing Time: {processing_time*1000:.2f} ms")

        conn.sendall(response.encode('utf-8'))
```

Code 1: ipc_server.py

```python
#!/usr/bin/env python3
# ipc_client.py

import numpy as np
import socket
import json
import time
import struct
from PIL import Image
```

```python

HOST = '172.17.0.2'
PORT = 9898

time.sleep(1)

image_path = 'eevee.png'  # path to the image
try:
    img = Image.open(image_path)
except FileNotFoundError:
    print(f"[Error] Image file '{image_path}' not found!")
    exit(1)

if img.mode != 'RGB':
    print(f"[Info] Converting image to RGB...")
    img = img.convert('RGB')

if img.size != (32, 32):
    print(f"[Info] Resizing image to 32x32...")
    img = img.resize((32, 32))

image_array = np.array(img).tolist()

json_data = json.dumps(image_array).encode('utf-8')

with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
    s.connect((HOST, PORT))

    start_time = time.time()

    # send 4 bytes to tell the size of the data
    s.sendall(struct.pack('>I', len(json_data)))

    # send all data
    s.sendall(json_data)

    print("\n[Info] Sent image data to server.")

    end_time = time.time()

    offloading_time = end_time - start_time
    print(f"\n[Performance] Offloading Time: {offloading_time*1000:.2f}
    ms")

    response = s.recv(4096)
    print("\n[Received Features from Server]:\n")
    print(response.decode('utf-8'))
```

Code 2: ipc_client.py