Sai Krishna Kaushik Garlapati

ID - 029336162

***Implement a Distributed System for Image Processing***

# Assignment:

In this assignment, you will implement a distributed system for processing images in parallel. The system will consist of a number of worker nodes, each of which will be responsible for processing a portion of the input image.

To begin, you will need to choose a programming language and a communication protocol for the worker nodes to use to communicate with each other. You may choose any language and protocol you wish, but it should be one that is well-suited to distributed computing.

Once you have chosen a language and a protocol, you will need to implement the following components:

1. A master node that is responsible for dividing the input image into smaller chunks and distributing them to the worker nodes. The master node should also be responsible for collecting the processed chunks from the worker nodes and reassembling them into a single output image.
2. A worker node that is responsible for receiving a chunk of the input image from the master node, processing it, and returning the processed chunk to the master node. The processing step can be anything you wish - for example, you could implement a simple image filtering algorithm or something more complex like facial recognition.
3. A simple client program that can be used to submit an input image to the master node for processing. The client program should be able to display the output image once it has been processed by the worker nodes.

# Required Tools:-

Docker

Docker Compose

Python

# Example Code

**Master Node -**

class MasterNode:

def \_\_init\_\_(self, num\_worker\_nodes):

self.num\_worker\_nodes = num\_worker\_nodes

self.comm = CommunicationProtocol() # Choose a communication protocol here

def process\_image(self, input\_image\_path):

# Load the input image

input\_image = Image.open(input\_image\_path)

width, height = input\_image.size

# Divide the input image into smaller chunks

chunk\_width = width // self.num\_worker\_nodes

chunk\_height = height

input\_chunks = []

for i in range(self.num\_worker\_nodes):

chunk = input\_image.crop((i \* chunk\_width, 0, (i + 1) \* chunk\_width, chunk\_height))

input\_chunks.append(chunk)

# Distribute the chunks to the worker nodes

for i, chunk in enumerate(input\_chunks):

self.comm.send(i, chunk)

# Collect the processed chunks from the worker nodes

output\_chunks = []

for i in range(self.num\_worker\_nodes):

chunk = self.comm.receive(i)

output\_chunks.append(chunk)

# Reassemble the output chunks into a single image

output\_image = Image.new('RGB', (width, height))

for i, chunk in enumerate(output\_chunks):

output\_image.paste(chunk, (i \* chunk\_width, 0))

return output\_image

**Worker node ➖**

class WorkerNode:

def \_\_init\_\_(self):

self.comm = CommunicationProtocol() # Choose a communication protocol here

def process\_chunk(self):

# Receive a chunk of the input image from the master node

chunk = self.comm.receive()

# Convert the image to grayscale and apply the Sobel edge detection filter

img\_array = np.array(chunk)

img\_array = np.mean(img\_array, axis=2) # Convert to grayscale

img\_array = sobel(img\_array)

# Convert the image back to a PIL Image object and return it

processed\_chunk = Image.fromarray(img\_array)

return processed\_chunk

Client code ➖

**// client code**

import sys

from PIL import Image

if \_\_name\_\_ == '\_\_main\_\_':

if len(sys.argv) != 3:

print('Usage: python client.py <input\_image> <num\_worker\_nodes>')

sys.exit(1)

input\_image\_path = sys.argv[1]

num\_worker\_nodes = int(sys.argv[2])

master\_node = MasterNode(num\_worker\_nodes)

output\_image = master\_node.process\_image(input\_image\_path)

output\_image.show()

# Deliverables

You should measure the time it takes to process the entire image, and compare it to the time it would take to process the image on a single node.

Once you have implemented and tested your system, you should write a short report discussing your design choices, the challenges you faced, and any improvements you would make if you were to do the project again.