University of East London

AIN SHAMS UNIVERSITY

FACULTY OF ENGINEERING

CREDIT HOURS ENG. PROGRAM

Computer engineering and software systems

AIN SHAMS UNIVERSITY

FACULTY OF ENGINEERING

Distributed Computing: CSE354

Distributed Image Processing System using Cloud Computing

Phase Four

Team 18

Submitted to:

Prof. Ayman M. Bahaa-Eldin

Eng. Mostafa Ashraf

Submitted by:

Mariam Essam Raafat Fakhry Bishay 20P6483

Mariam Mohamed GamalEldin Ashmawy 20P4318

Rawan Ahmed Muhammed Muhammed Abdulhay 20P5251

Sherwet Mohamed Khalil Barakat 20P8105

Introduction

The project is aimed to provide distributed image processing applications on cloud. In the project, Nodes (Master, Slave1, Slave2, and Slave3) communicate with each other through MPI (Message-Passing Interface). The project also has the goal of ensuring scalability and fault tolerance but to some degree, while also ensuring proper testing is applied. Advanced Image Processing Operations will be implemented using OPENCV and NUMPY, GUI is provided by exploiting the approach of Remote Desktop, where the user connects to Master vm using Remote Desktop Protocol. The main aim of this distribution is to ensure high performance on image manipulation.

Youtube Link: https://youtu.be/k-Sa5HB5nIQ
Github Link: https://github.com/XXMariamX/Distributed-Project

Table of Contents

Table of Figures

Table of Figures

Task Breakdown Structure:

- Project starts by implementing Image Processing Operations (**OPENCV**); including Filtering, Edge Detection and Color Manipulation Techniques. Also implementing GUI (**TKintr**) that includes the following:
    1- Uploading 1 image or a batch of images.
    2- A dropdown for different image processing operations to be applied.
    3- Downloading the processed image(s) for the batch.
    4- Image Viewing Window
- Setting up Cloud Environment (Microsoft Azure) by creating multiple vms on the cloud, also, adding in-bound rules for the vm to allow networking between the nodes.
- MPI was integrated into the project. We have faced **some problems** in implementing such a step, which we dealt with using reinstalling build essentials, and eventually we had to install MPI from scratch using the src code.
    - Host file was created that includes nodes' private ip addresses.
    - Setting Up **NFS** (Network File System) for sharing common files between the nodes.
    - Then MPI Cluster was configured by setting SSH between nodes to allow communication between nodes without using passwords, in the following steps:
    1- Creating SSH key pair.
    2- Passing the public key to the other node you want to communicate with using **ssh-copy-id** command. ThenRepeating the steps for other nodes.
- Considering a degree of Fault Tolerance. Our mechanism was based on using a watcher to check every 3 seconds if the program crashes and run it again. However, on each run, at the beginning, new host file is edited to include only nodes which were successfully SSHed, Thus when one slave is stopped the program will crash and rerun with an edited host file.

Beneficiaries of The Project

For project beneficiaries:

- Firstly, it ensures high processing through distribution.
- Secondly and most importantly, it was for educational purposes, the earnings were to learn how to develop a distributed system. Learn how to set up cloud environment and virtual machines. Also how to take into consideration what degree of fault tolerance and scalability to implement based on the requirements. Additionally, how to test cloud applications, not to mention handling the configurations, and setting an architecture.

Detailed Analysis

1- Requirements:

• **Distributed Processing**: The system should be able to distribute image processing tasks across multiple virtual machines in the cloud.

• **Image Processing Algorithms:** Implement various image processing algorithms such as filtering, edge detection, and color manipulation.

• **Scalability**: The system should be scalable, allowing for the addition of more virtual machines as the workload increases.

• **Fault Tolerance:** The system should be resilient to failures, with the ability to reassign tasks from failed nodes to operational ones.

2- Risks:

- System Components Integration might need troubleshooting.
- Performance Issues.
- Free tiers on cloud might run out, and there is no financial support.

3- Feasibility:

- Can use other cloud service platforms in case free tier in one cloud ran out.
- Good troubleshooting and debugging can be carried out.

Detailed Project Description

1- Objectives and Implementation:

The project allows users to connect to master node through RDP (remote desktop protocol. Then the user is allowed to upload image for processing, and the application handles the distribution of workload among slaves. Also, the project handles stopping one of the salves and provides some degree of scalability. In addition to testing and providing documentation and user manual for the users.

2- Deliverables:

Our implementation of the project manages successfully to answer the following questions:
1. What is the main objective of our project?
2. Which technologies have we decided to use and why?
3. What are the key components of our system architecture?
1. How does our worker thread process tasks?
2. What basic image processing operations have we implemented?
3. How have we set up our cloud environment?
1. What advanced image processing operations have we implemented?
2. How does our system handle distributed processing?
3. How have we implemented scalability and fault tolerance?
1. What were the results of our system testing?
2. What information is included in our system documentation?
3. How did we deploy our system to the cloud?

Roles of Each Member

Table 1 Member Roles

Mariam Essam Raafat	Setting up Cloud Environment, MPI Code & Configuration, Documentation, GUI
Mariam Mohamed GamalEldin	Setting up Cloud Environment, Fault Tolerance, MPI Code & Configuration
Rawan Ahmed Muhammed	Image Processing, GUI Implementation, MPI Code, Documentation
Sherwet Mohamed Khalil	Scalability, MPI Code & Configuration

System Architecture

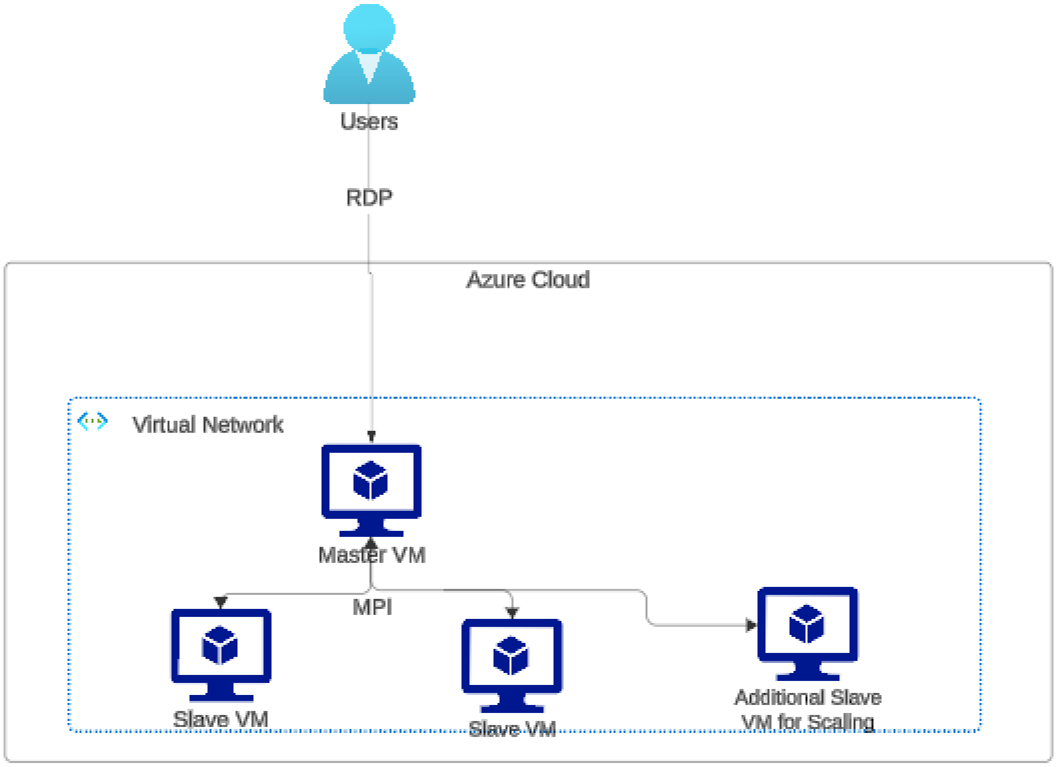


Figure 1: System Architecture.

- Master VM -> Provide GUI to users and distribute workload.
- Slave VMs -> receives workload, process images and send back to Master.
- RDP -> Remote desktop protocol allows user to connect to vm desktop remotely.

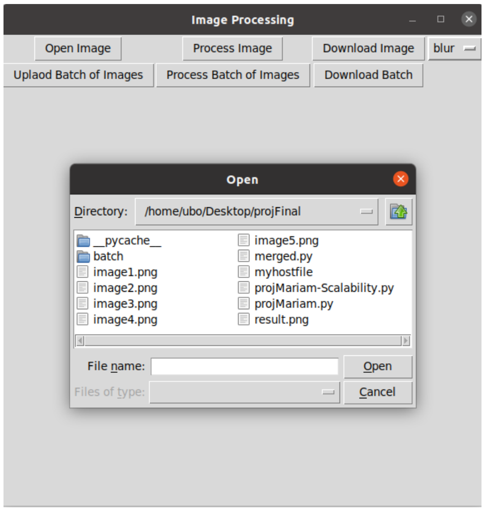
Testing Scenarios

This part is already included in the video, our approach however was based on manual testing, where we tried to stop vm (Slave 2) and found that our program closed and restarted due to our watcher, which provides a degree of availability and fault tolerance.

End User Guide

1- Users must install Remote Desktop Connection app.
2- Connect to 20.84.91.62 (Master's Public IP Address).
3- Using our pre-established username and password (azureuser, azureuser) respectively.
4- Install WinSCP. Connect to master IP. Then share images from your local host to Master, and ensure placig images in /home/azureuser/projFinal
5- Open terminal and run $ cd/home/azureuser/projFinal
6- Then $ ./run
7-

Sample as an End-User



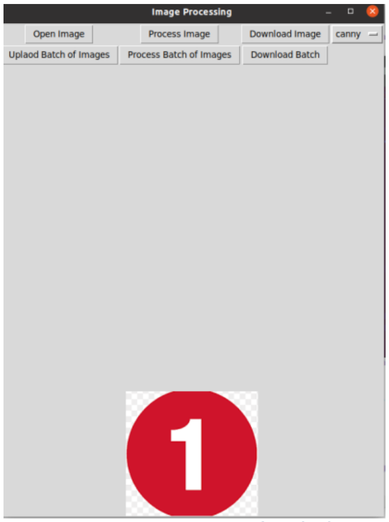
blur

Directory: /home/ubo/Desktop/projFinal

```
__pycache__        image5.png
batch              merged.py
image1.png         myhostfile
image2.png         projMariam-Scalability.py
image3.png         projMariam.py
image4.png         result.png
```

File name:

Figure 2: Uploading Image or Batch.



canny

Figure 3: Image Uploaded.

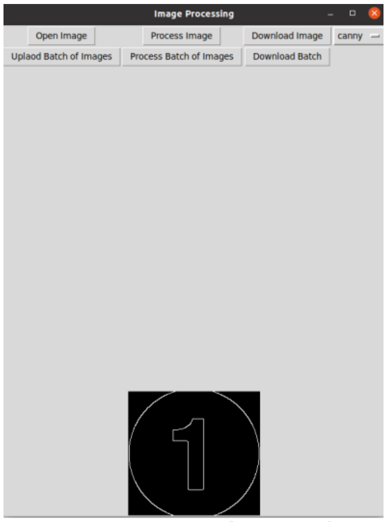


Figure 4: Image Processed.

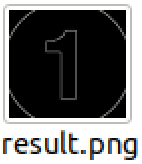


Figure 5: Image Downloaded.

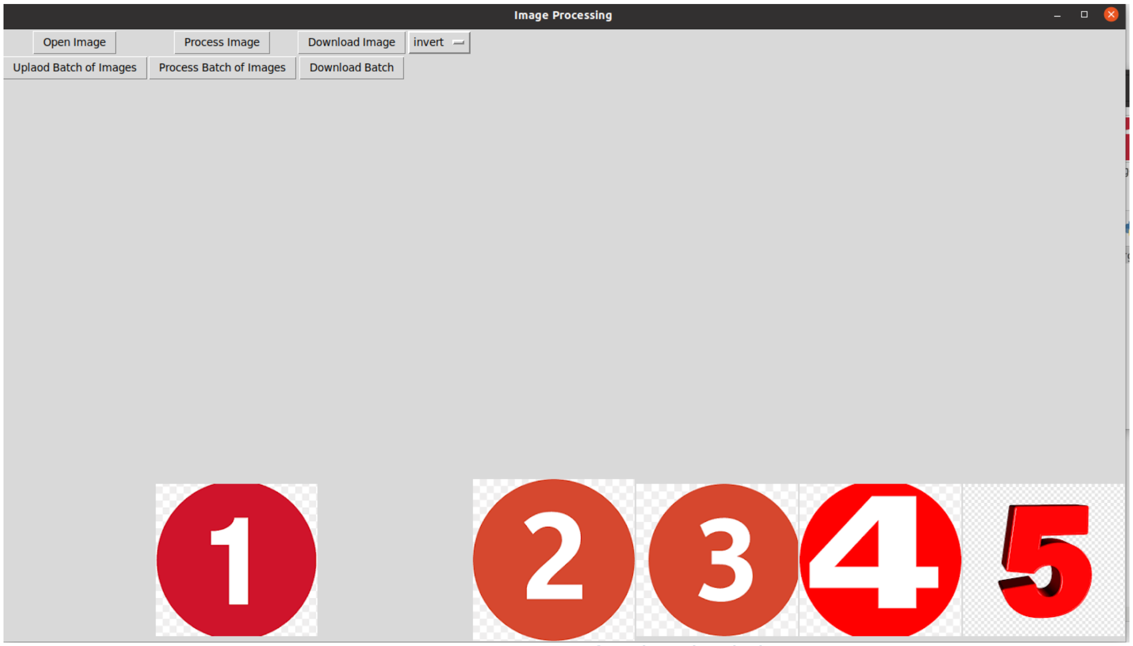


Figure 6: Batch Uploaded.

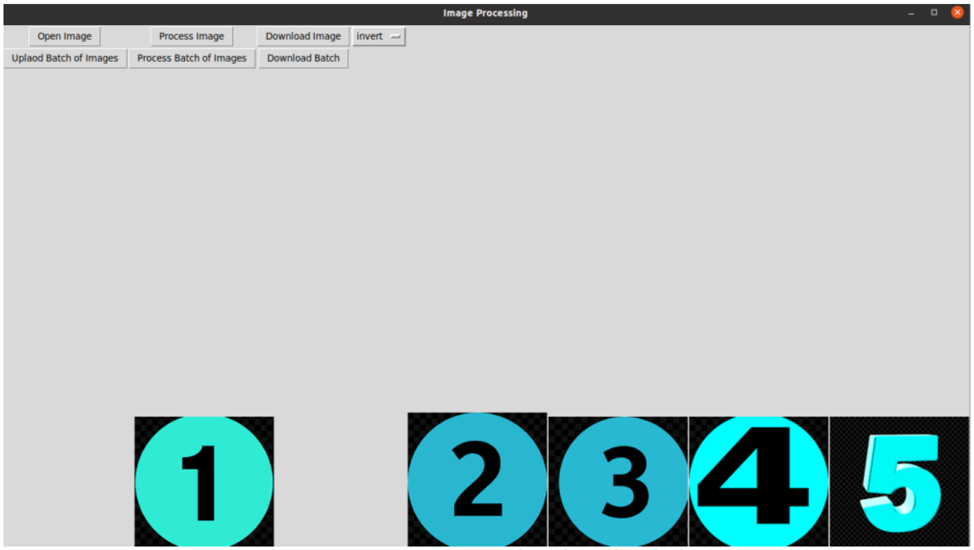


Figure 7: Batch Uploaded.

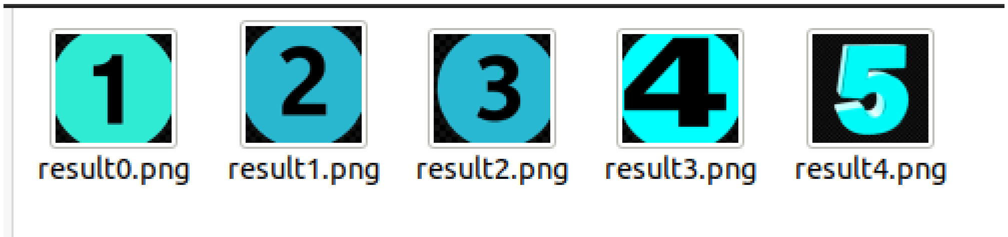
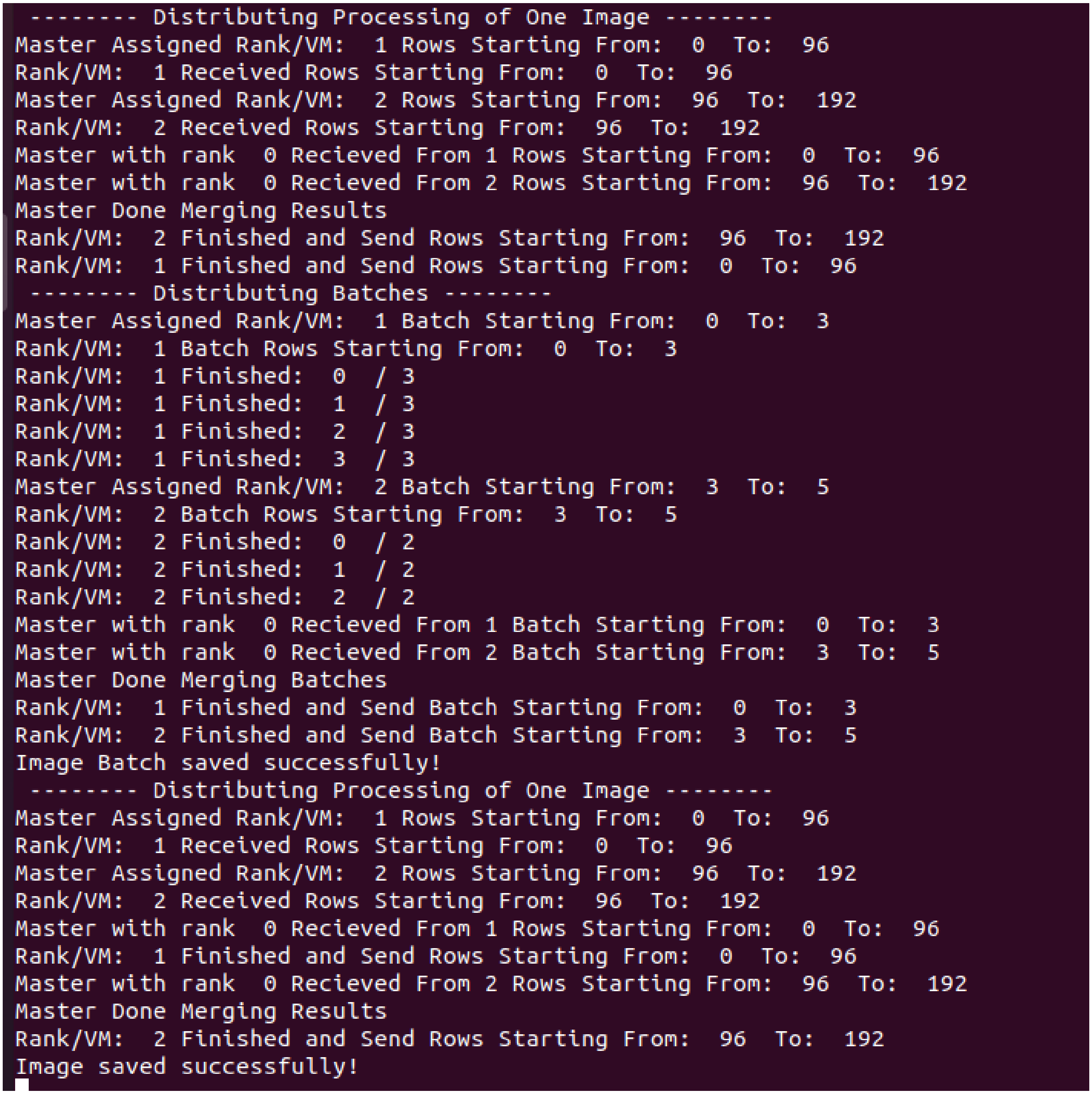


Figure 8: Batch Downloaded.



```
 -------- Distributing Processing of One Image --------
Master Assigned Rank/VM:  1 Rows Starting From:  0  To:  96
Rank/VM:  1 Received Rows Starting From:  0  To:  96
Master Assigned Rank/VM:  2 Rows Starting From:  96  To:  192
Rank/VM:  2 Received Rows Starting From:  96  To:  192
Master with rank  0 Recieved From 1 Rows Starting From:  0  To:  96
Master with rank  0 Recieved From 2 Rows Starting From:  96  To:  192
Master Done Merging Results
Rank/VM:  2 Finished and Send Rows Starting From:  96  To:  192
Rank/VM:  1 Finished and Send Rows Starting From:  0  To:  96
 -------- Distributing Batches --------
Master Assigned Rank/VM:  1 Batch Starting From:  0  To:  3
Rank/VM:  1 Batch Rows Starting From:  0  To:  3
Rank/VM:  1 Finished:  0  / 3
Rank/VM:  1 Finished:  1  / 3
Rank/VM:  1 Finished:  2  / 3
Rank/VM:  1 Finished:  3  / 3
Master Assigned Rank/VM:  2 Batch Starting From:  3  To:  5
Rank/VM:  2 Batch Rows Starting From:  3  To:  5
Rank/VM:  2 Finished:  0  / 2
Rank/VM:  2 Finished:  1  / 2
Rank/VM:  2 Finished:  2  / 2
Master with rank  0 Recieved From 1 Batch Starting From:  0  To:  3
Master with rank  0 Recieved From 2 Batch Starting From:  3  To:  5
Master Done Merging Batches
Rank/VM:  1 Finished and Send Batch Starting From:  0  To:  3
Rank/VM:  2 Finished and Send Batch Starting From:  3  To:  5
Image Batch saved successfully!
 -------- Distributing Processing of One Image --------
Master Assigned Rank/VM:  1 Rows Starting From:  0  To:  96
Rank/VM:  1 Received Rows Starting From:  0  To:  96
Master Assigned Rank/VM:  2 Rows Starting From:  96  To:  192
Rank/VM:  2 Received Rows Starting From:  96  To:  192
Master with rank  0 Recieved From 1 Rows Starting From:  0  To:  96
Rank/VM:  1 Finished and Send Rows Starting From:  0  To:  96
Master with rank  0 Recieved From 2 Rows Starting From:  96  To:  192
Master Done Merging Results
Rank/VM:  2 Finished and Send Rows Starting From:  96  To:  192
Image saved successfully!
```

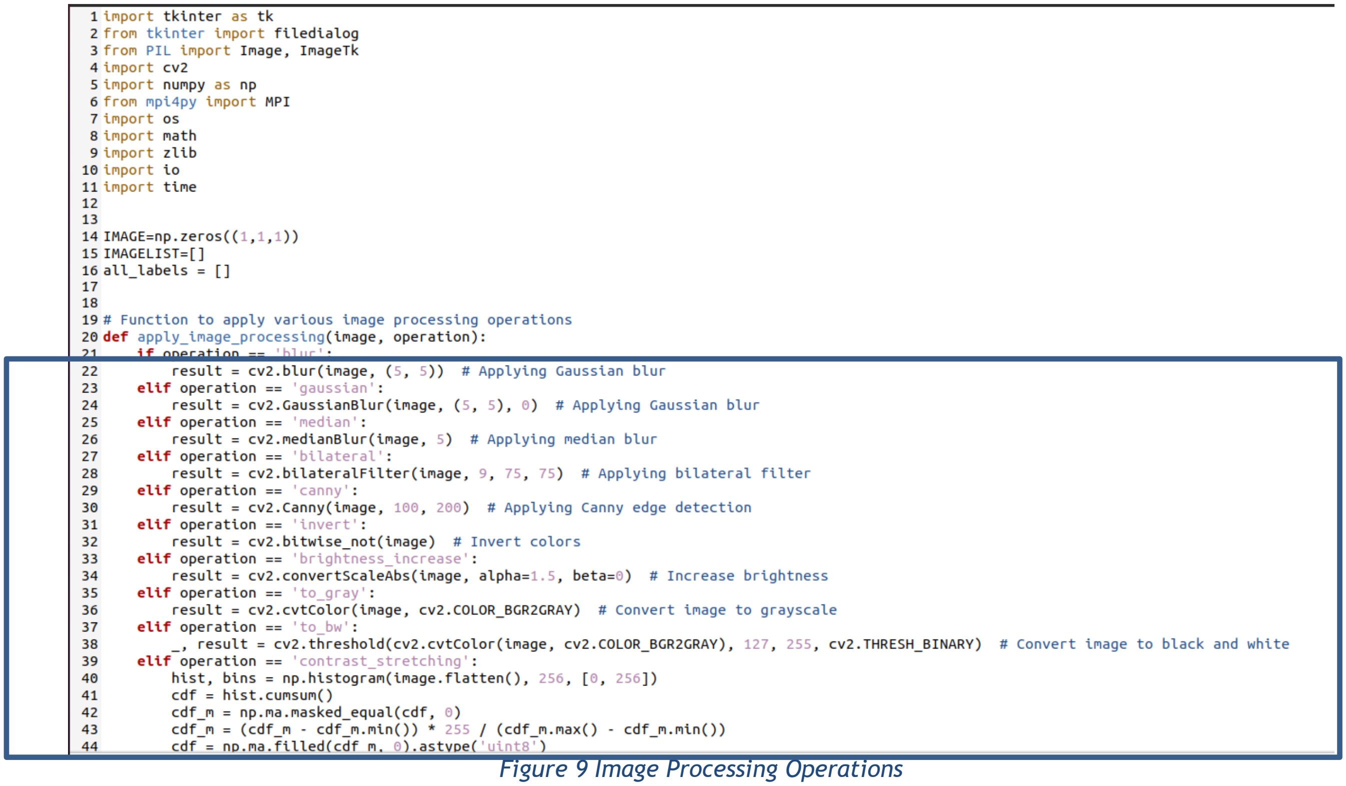
Figure 9 Logging

Conclusion

Our application provides:

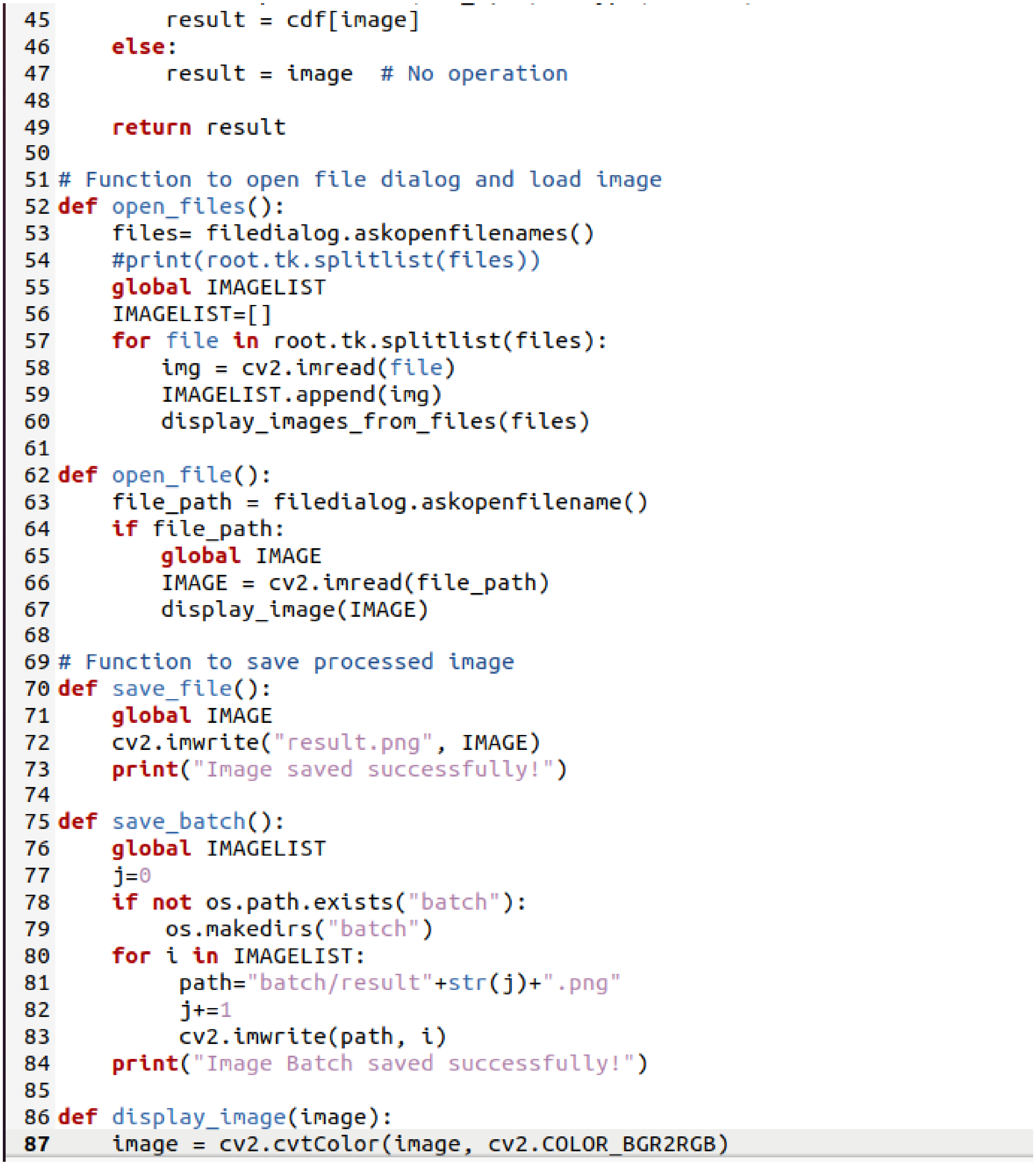
- Fault Tolerance.
    - Scalability.
    - Distribution.
    - High Processing.
    - User-Friendly GUI.

Code



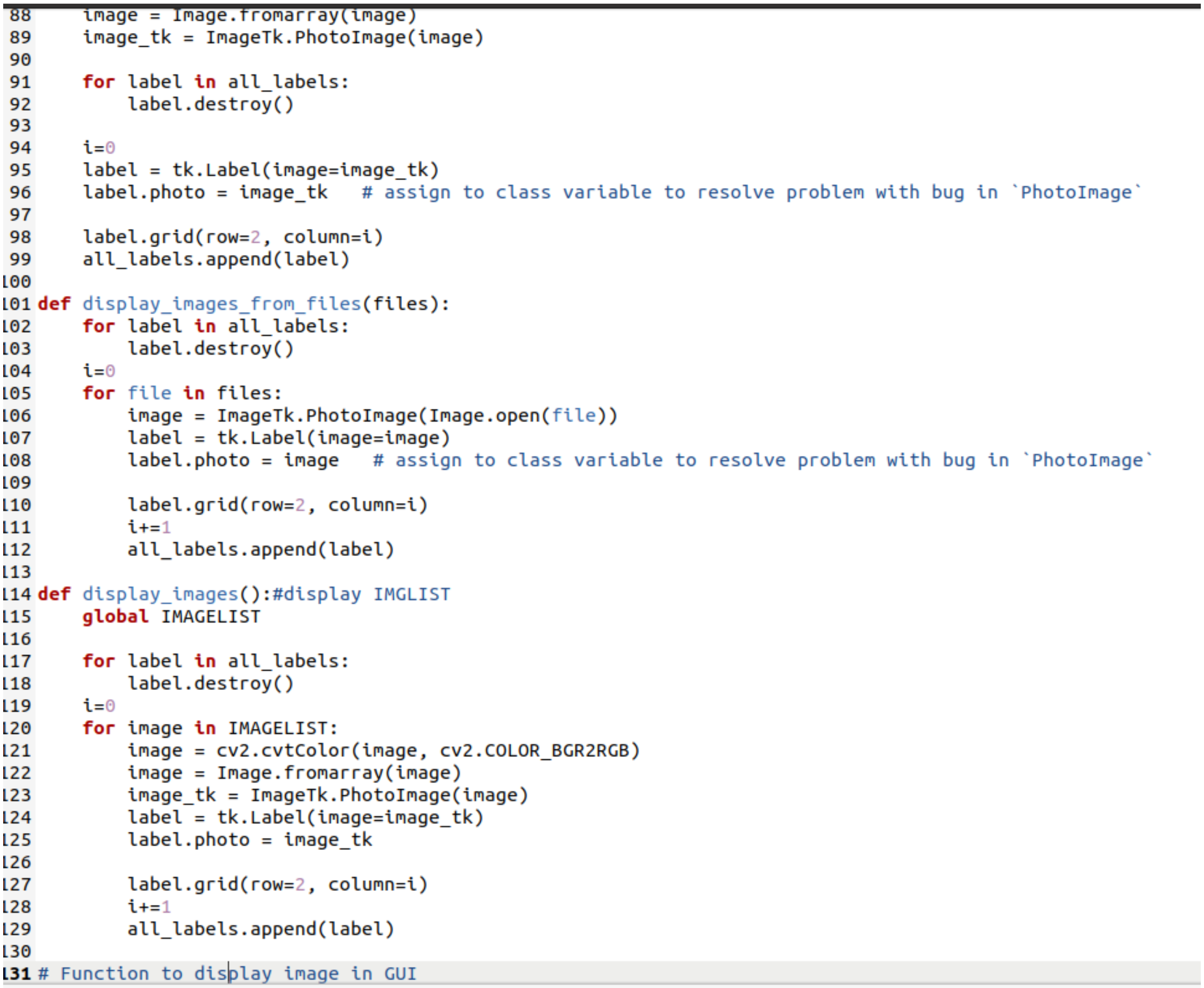
```
import tkinter as tk
from tkinter import filedialog
from PIL import Image, ImageTk
import cv2
import numpy as np
from mpi4py import MPI
import os
import math
import zlib
import io
import time


IMAGE=np.zeros((1,1,1))
IMAGELIST=[]
all_labels = []


# Function to apply various image processing operations
def apply_image_processing(image, operation):
    if operation == 'blur':
        result = cv2.blur(image, (5, 5))  # Applying Gaussian blur
    elif operation == 'gaussian':
        result = cv2.GaussianBlur(image, (5, 5), 0)  # Applying Gaussian blur
    elif operation == 'median':
        result = cv2.medianBlur(image, 5)  # Applying median blur
    elif operation == 'bilateral':
        result = cv2.bilateralFilter(image, 9, 75, 75)  # Applying bilateral filter
    elif operation == 'canny':
        result = cv2.Canny(image, 100, 200)  # Applying Canny edge detection
    elif operation == 'invert':
        result = cv2.bitwise_not(image)  # Invert colors
    elif operation == 'brightness_increase':
        result = cv2.convertScaleAbs(image, alpha=1.5, beta=0)  # Increase brightness
    elif operation == 'to_gray':
        result = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)  # Convert image to grayscale
    elif operation == 'to_bw':
        _, result = cv2.threshold(cv2.cvtColor(image, cv2.COLOR_BGR2GRAY), 127, 255, cv2.THRESH_BINARY)  # Convert image to black and white
    elif operation == 'contrast_stretching':
        hist, bins = np.histogram(image.flatten(), 256, [0, 256])
        cdf = hist.cumsum()
        cdf_m = np.ma.masked_equal(cdf, 0)
        cdf_m = (cdf_m - cdf_m.min()) * 255 / (cdf_m.max() - cdf_m.min())
        cdf = np.ma.filled(cdf_m, 0).astype('uint8')
```

Figure 9 Image Processing Operations



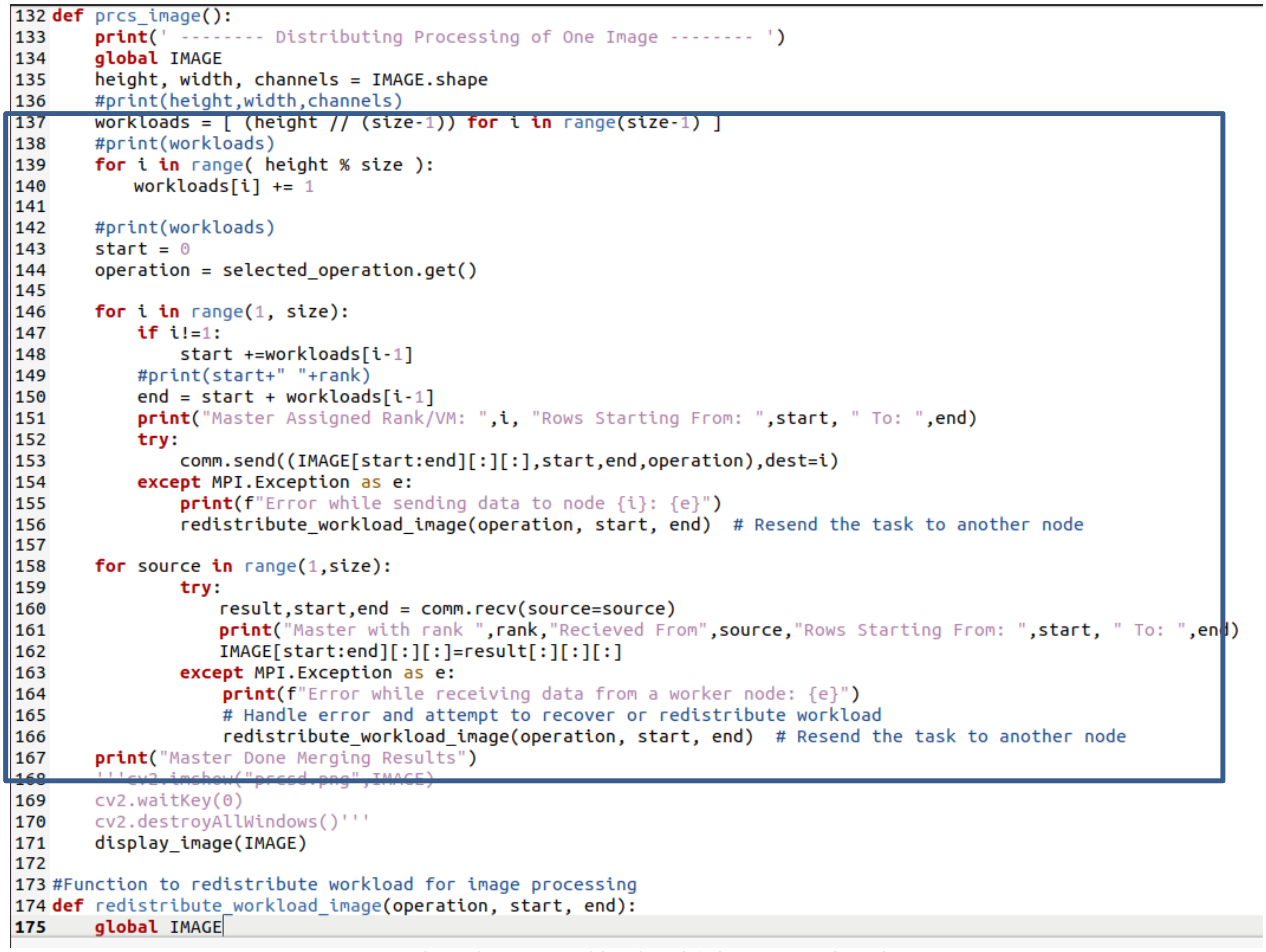
```python
        result = cdf[image]
    else:
        result = image   # No operation

    return result

# Function to open file dialog and load image
def open_files():
    files= filedialog.askopenfilenames()
    #print(root.tk.splitlist(files))
    global IMAGELIST
    IMAGELIST=[]
    for file in root.tk.splitlist(files):
        img = cv2.imread(file)
        IMAGELIST.append(img)
        display_images_from_files(files)

def open_file():
    file_path = filedialog.askopenfilename()
    if file_path:
        global IMAGE
        IMAGE = cv2.imread(file_path)
        display_image(IMAGE)

# Function to save processed image
def save_file():
    global IMAGE
    cv2.imwrite("result.png", IMAGE)
    print("Image saved successfully!")

def save_batch():
    global IMAGELIST
    j=0
    if not os.path.exists("batch"):
        os.makedirs("batch")
    for i in IMAGELIST:
         path="batch/result"+str(j)+".png"
         j+=1
         cv2.imwrite(path, i)
    print("Image Batch saved successfully!")

def display_image(image):
    image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
```

Figure 10 Code



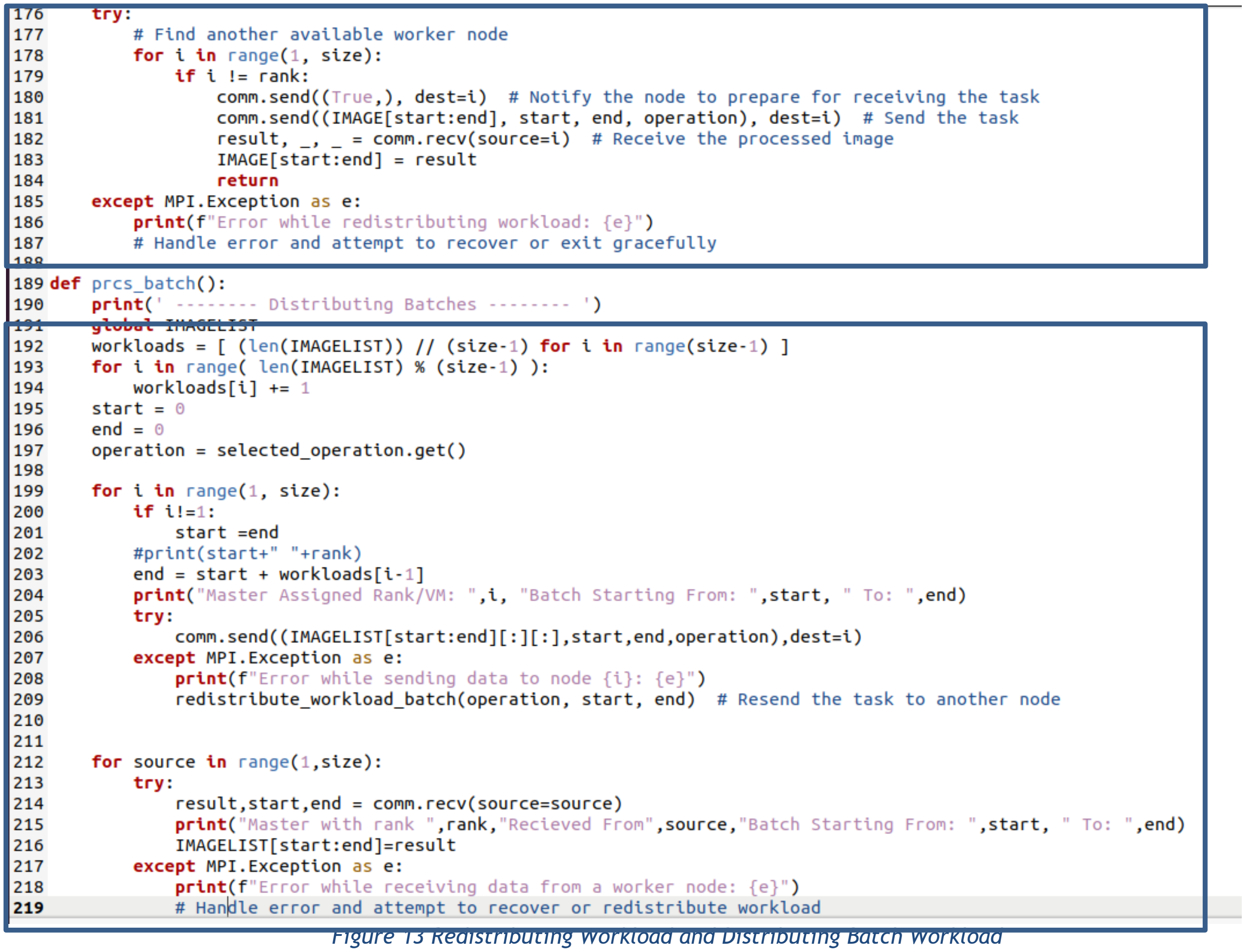
```python
    image = Image.fromarray(image)
    image_tk = ImageTk.PhotoImage(image)

    for label in all_labels:
        label.destroy()

    i=0
    label = tk.Label(image=image_tk)
    label.photo = image_tk   # assign to class variable to resolve problem with bug in `PhotoImage`

    label.grid(row=2, column=i)
    all_labels.append(label)

def display_images_from_files(files):
    for label in all_labels:
        label.destroy()
    i=0
    for file in files:
        image = ImageTk.PhotoImage(Image.open(file))
        label = tk.Label(image=image)
        label.photo = image   # assign to class variable to resolve problem with bug in `PhotoImage`

        label.grid(row=2, column=i)
        i+=1
        all_labels.append(label)

def display_images():#display IMGLIST
    global IMAGELIST

    for label in all_labels:
        label.destroy()
    i=0
    for image in IMAGELIST:
        image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
        image = Image.fromarray(image)
        image_tk = ImageTk.PhotoImage(image)
        label = tk.Label(image=image_tk)
        label.photo = image_tk

        label.grid(row=2, column=i)
        i+=1
        all_labels.append(label)

# Function to display image in GUI
```

Figure 11 Code



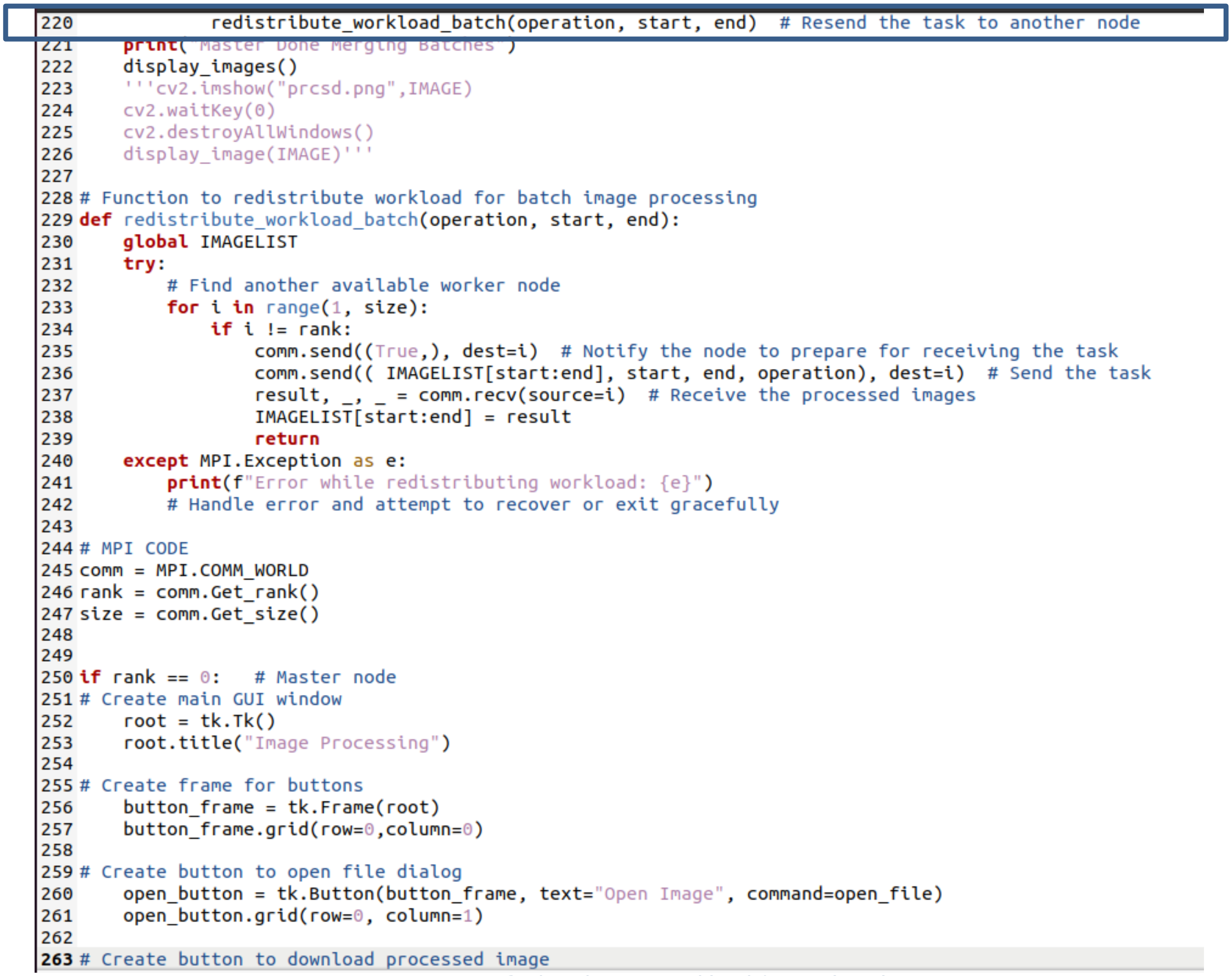
```
def prcs_image():
    print(' -------- Distributing Processing of One Image -------- ')
    global IMAGE
    height, width, channels = IMAGE.shape
    #print(height,width,channels)
    workloads = [ (height // (size-1)) for i in range(size-1) ]
    #print(workloads)
    for i in range( height % size ):
        workloads[i] += 1

    #print(workloads)
    start = 0
    operation = selected_operation.get()

    for i in range(1, size):
        if i!=1:
            start +=workloads[i-1]
        #print(start+" "+rank)
        end = start + workloads[i-1]
        print("Master Assigned Rank/VM: ",i, "Rows Starting From: ",start, " To: ",end)
        try:
            comm.send((IMAGE[start:end][:][:],start,end,operation),dest=i)
        except MPI.Exception as e:
            print(f"Error while sending data to node {i}: {e}")
            redistribute_workload_image(operation, start, end)  # Resend the task to another node

    for source in range(1,size):
            try:
                result,start,end = comm.recv(source=source)
                print("Master with rank ",rank,"Recieved From",source,"Rows Starting From: ",start, " To: ",end)
                IMAGE[start:end][:][:]=result[:][:][:]
            except MPI.Exception as e:
                print(f"Error while receiving data from a worker node: {e}")
                # Handle error and attempt to recover or redistribute workload
                redistribute_workload_image(operation, start, end)  # Resend the task to another node
    print("Master Done Merging Results")
    '''cv2.imshow("prcsd.png",IMAGE)
    cv2.waitKey(0)
    cv2.destroyAllWindows()'''
    display_image(IMAGE)

#Function to redistribute workload for image processing
def redistribute_workload_image(operation, start, end):
    global IMAGE
```

Figure 12 Distributing Workload and Achieving Fault Tolerance



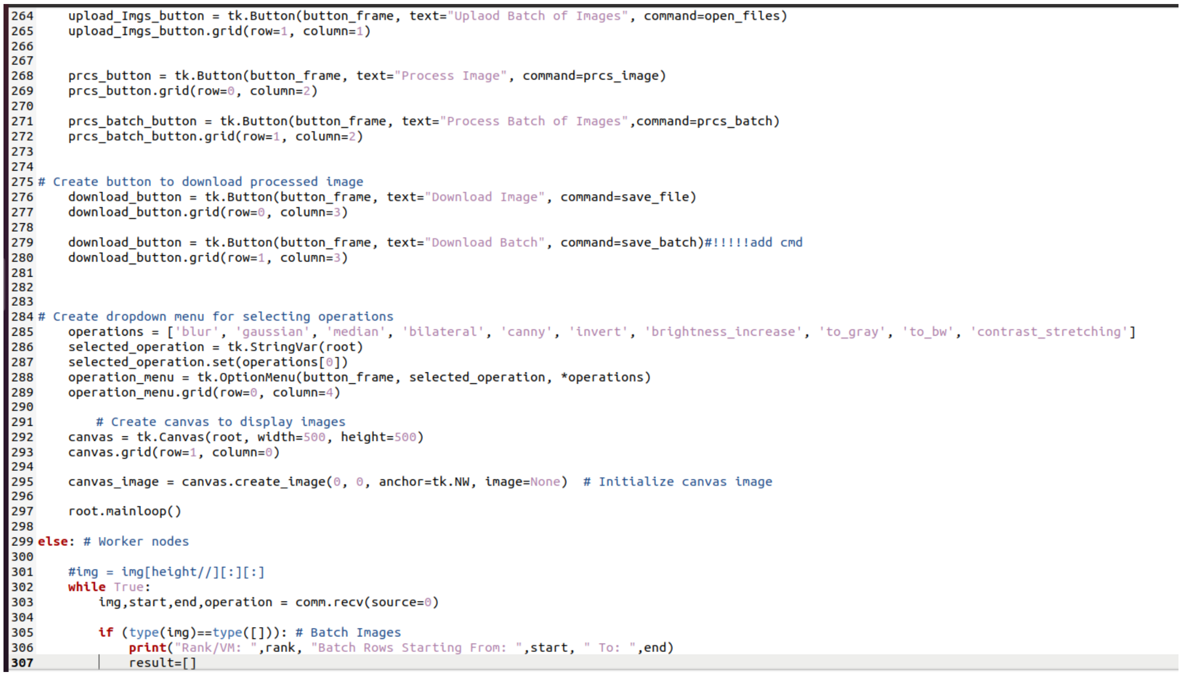
```
    try:
        # Find another available worker node
        for i in range(1, size):
            if i != rank:
                comm.send((True,), dest=i)  # Notify the node to prepare for receiving the task
                comm.send((IMAGE[start:end], start, end, operation), dest=i)  # Send the task
                result, _, _ = comm.recv(source=i)  # Receive the processed image
                IMAGE[start:end] = result
                return
    except MPI.Exception as e:
        print(f"Error while redistributing workload: {e}")
        # Handle error and attempt to recover or exit gracefully

def prcs_batch():
    print(' -------- Distributing Batches -------- ')
    global IMAGELIST
    workloads = [ (len(IMAGELIST)) // (size-1) for i in range(size-1) ]
    for i in range( len(IMAGELIST) % (size-1) ):
        workloads[i] += 1
    start = 0
    end = 0
    operation = selected_operation.get()

    for i in range(1, size):
        if i!=1:
            start =end
        #print(start+" "+rank)
        end = start + workloads[i-1]
        print("Master Assigned Rank/VM: ",i, "Batch Starting From: ",start, " To: ",end)
        try:
            comm.send((IMAGELIST[start:end][:][:],start,end,operation),dest=i)
        except MPI.Exception as e:
            print(f"Error while sending data to node {i}: {e}")
            redistribute_workload_batch(operation, start, end)  # Resend the task to another node


    for source in range(1,size):
        try:
            result,start,end = comm.recv(source=source)
            print("Master with rank ",rank,"Recieved From",source,"Batch Starting From: ",start, " To: ",end)
            IMAGELIST[start:end]=result
        except MPI.Exception as e:
            print(f"Error while receiving data from a worker node: {e}")
            # Handle error and attempt to recover or redistribute workload
```

Figure 13 Redistributing Workload and Distributing Batch Workload



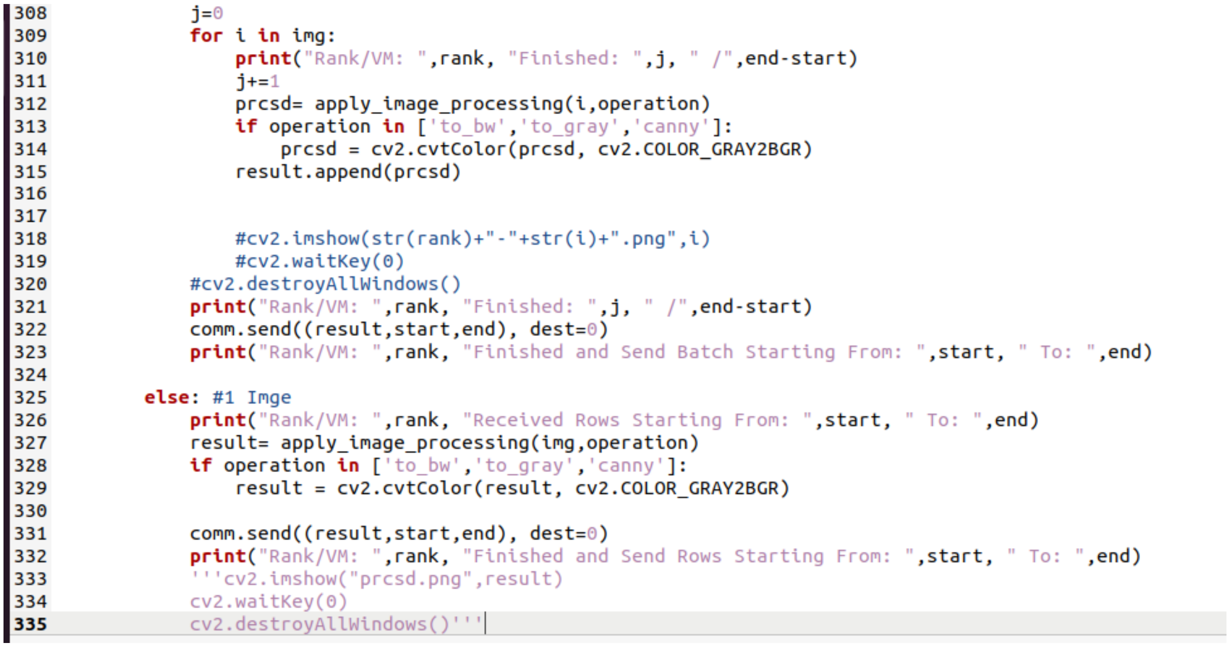
```
        redistribute_workload_batch(operation, start, end)  # Resend the task to another node
    print("Master Done Merging Batches")
    display_images()
    '''cv2.imshow("prcsd.png",IMAGE)
    cv2.waitKey(0)
    cv2.destroyAllWindows()
    display_image(IMAGE)'''

# Function to redistribute workload for batch image processing
def redistribute_workload_batch(operation, start, end):
    global IMAGELIST
    try:
        # Find another available worker node
        for i in range(1, size):
            if i != rank:
                comm.send((True,), dest=i)  # Notify the node to prepare for receiving the task
                comm.send(( IMAGELIST[start:end], start, end, operation), dest=i)  # Send the task
                result, _, _ = comm.recv(source=i)  # Receive the processed images
                IMAGELIST[start:end] = result
                return
    except MPI.Exception as e:
        print(f"Error while redistributing workload: {e}")
        # Handle error and attempt to recover or exit gracefully

# MPI CODE
comm = MPI.COMM_WORLD
rank = comm.Get_rank()
size = comm.Get_size()


if rank == 0:   # Master node
# Create main GUI window
    root = tk.Tk()
    root.title("Image Processing")

# Create frame for buttons
    button_frame = tk.Frame(root)
    button_frame.grid(row=0,column=0)

# Create button to open file dialog
    open_button = tk.Button(button_frame, text="Open Image", command=open_file)
    open_button.grid(row=0, column=1)

# Create button to download processed image
```

Figure 14 Redistributing Workload for Fault Tolerance



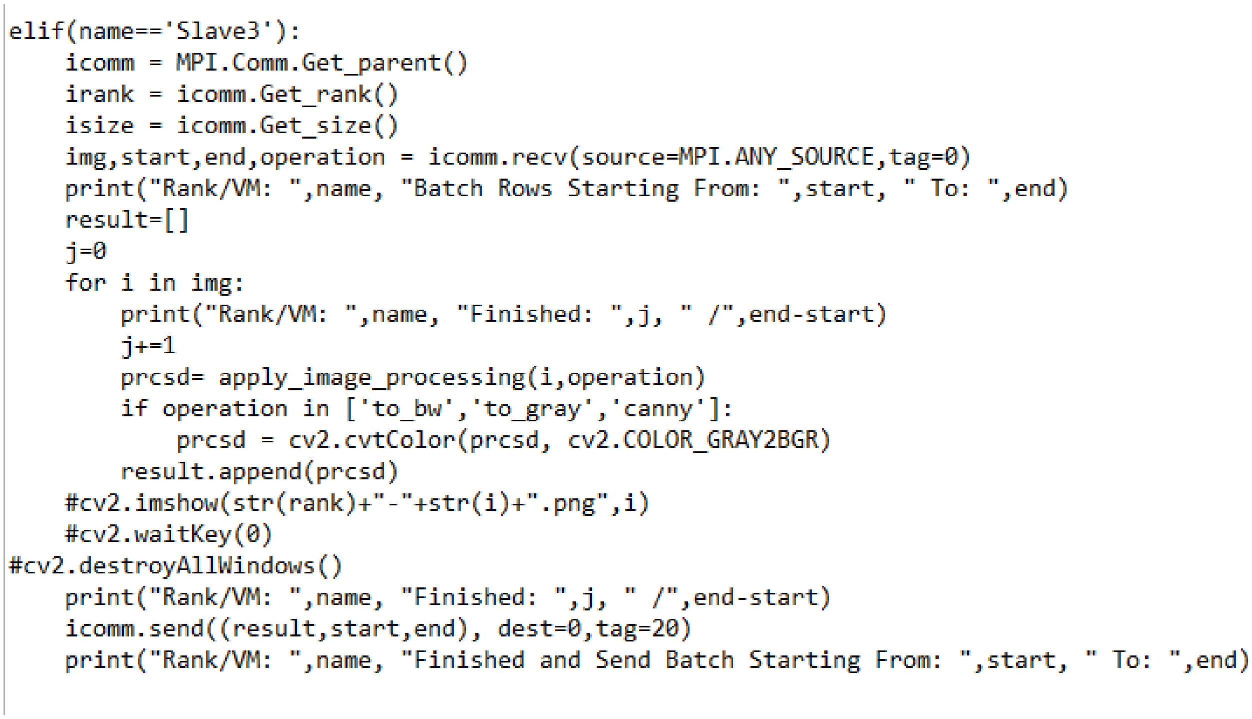
```
    upload_Imgs_button = tk.Button(button_frame, text="Uplaod Batch of Images", command=open_files)
    upload_Imgs_button.grid(row=1, column=1)


    prcs_button = tk.Button(button_frame, text="Process Image", command=prcs_image)
    prcs_button.grid(row=0, column=2)

    prcs_batch_button = tk.Button(button_frame, text="Process Batch of Images",command=prcs_batch)
    prcs_batch_button.grid(row=1, column=2)


# Create button to download processed image
    download_button = tk.Button(button_frame, text="Download Image", command=save_file)
    download_button.grid(row=0, column=3)

    download_button = tk.Button(button_frame, text="Download Batch", command=save_batch)#!!!!!add cmd
    download_button.grid(row=1, column=3)



# Create dropdown menu for selecting operations
    operations = ['blur', 'gaussian', 'median', 'bilateral', 'canny', 'invert', 'brightness_increase', 'to_gray', 'to_bw', 'contrast_stretching']
    selected_operation = tk.StringVar(root)
    selected_operation.set(operations[0])
    operation_menu = tk.OptionMenu(button_frame, selected_operation, *operations)
    operation_menu.grid(row=0, column=4)

        # Create canvas to display images
    canvas = tk.Canvas(root, width=500, height=500)
    canvas.grid(row=1, column=0)

    canvas_image = canvas.create_image(0, 0, anchor=tk.NW, image=None)  # Initialize canvas image

    root.mainloop()

else: # Worker nodes

    #img = img[height//][:][:]
    while True:
        img,start,end,operation = comm.recv(source=0)

        if (type(img)==type([])): # Batch Images
            print("Rank/VM: ",rank, "Batch Rows Starting From: ",start, " To: ",end)
            result=[]
```

Figure 15 Code



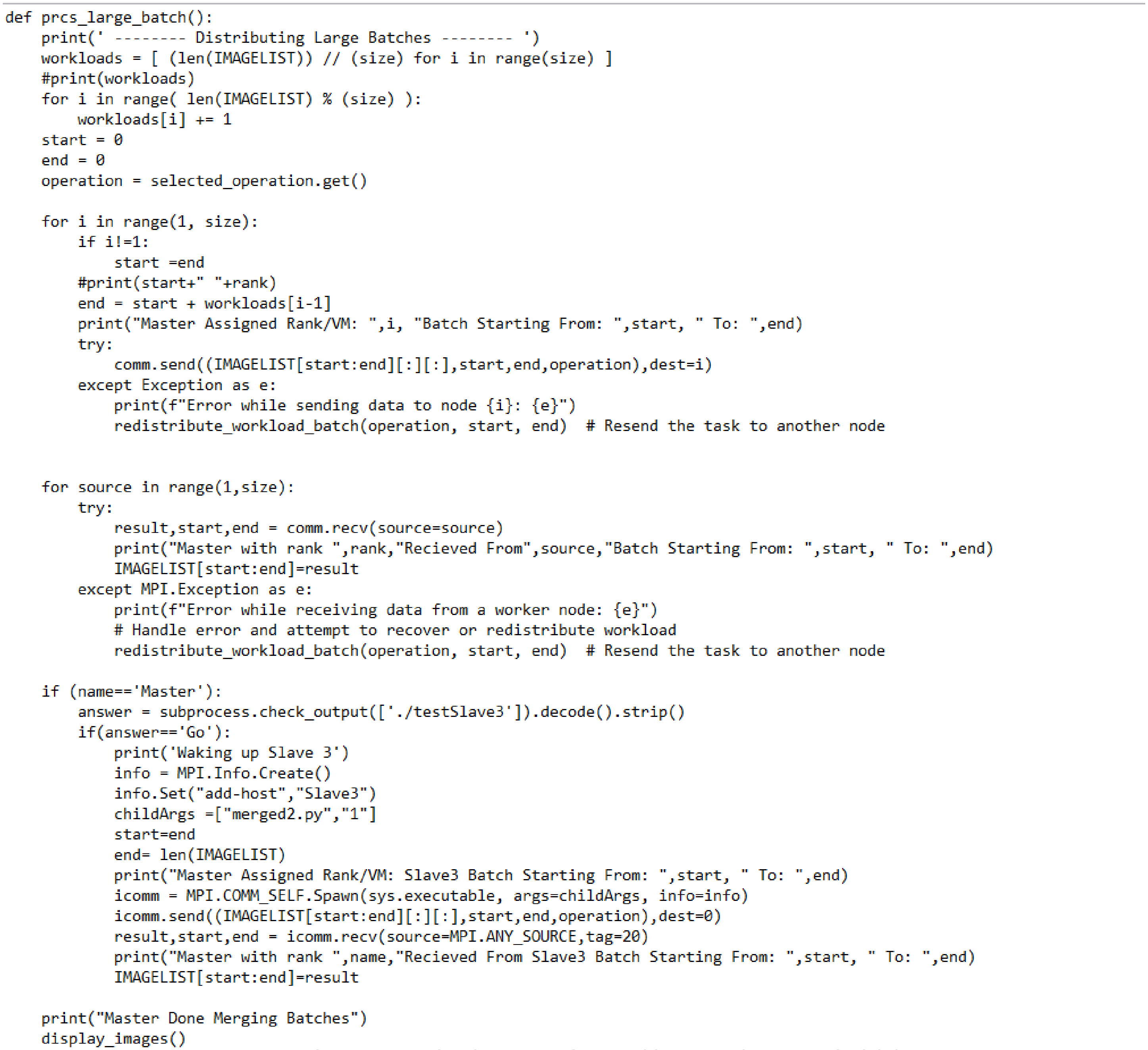
```
        j=0
        for i in img:
            print("Rank/VM: ",rank, "Finished: ",j, " /",end-start)
            j+=1
            prcsd= apply_image_processing(i,operation)
            if operation in ['to_bw','to_gray','canny']:
                prcsd = cv2.cvtColor(prcsd, cv2.COLOR_GRAY2BGR)
            result.append(prcsd)


            #cv2.imshow(str(rank)+"-"+str(i)+".png",i)
            #cv2.waitKey(0)
        #cv2.destroyAllWindows()
        print("Rank/VM: ",rank, "Finished: ",j, " /",end-start)
        comm.send((result,start,end), dest=0)
        print("Rank/VM: ",rank, "Finished and Send Batch Starting From: ",start, " To: ",end)

    else: #1 Imge
        print("Rank/VM: ",rank, "Received Rows Starting From: ",start, " To: ",end)
        result= apply_image_processing(img,operation)
        if operation in ['to_bw','to_gray','canny']:
            result = cv2.cvtColor(result, cv2.COLOR_GRAY2BGR)

        comm.send((result,start,end), dest=0)
        print("Rank/VM: ",rank, "Finished and Send Rows Starting From: ",start, " To: ",end)
        '''cv2.imshow("prcsd.png",result)
        cv2.waitKey(0)
        cv2.destroyAllWindows()'''
```

Figure 16 Worker Nodes



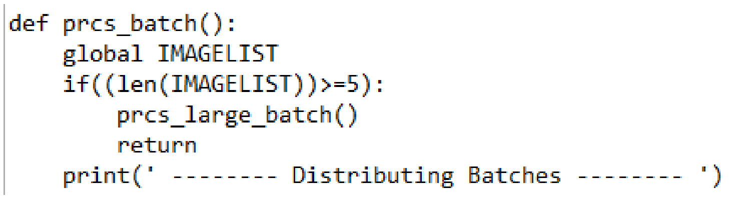
```
elif(name=='Slave3'):
    icomm = MPI.Comm.Get_parent()
    irank = icomm.Get_rank()
    isize = icomm.Get_size()
    img,start,end,operation = icomm.recv(source=MPI.ANY_SOURCE,tag=0)
    print("Rank/VM: ",name, "Batch Rows Starting From: ",start, " To: ",end)
    result=[]
    j=0
    for i in img:
        print("Rank/VM: ",name, "Finished: ",j, " /",end-start)
        j+=1
        prcsd= apply_image_processing(i,operation)
        if operation in ['to_bw','to_gray','canny']:
            prcsd = cv2.cvtColor(prcsd, cv2.COLOR_GRAY2BGR)
        result.append(prcsd)
    #cv2.imshow(str(rank)+"-"+str(i)+".png",i)
    #cv2.waitKey(0)
#cv2.destroyAllWindows()
    print("Rank/VM: ",name, "Finished: ",j, " /",end-start)
    icomm.send((result,start,end), dest=0,tag=20)
    print("Rank/VM: ",name, "Finished and Send Batch Starting From: ",start, " To: ",end)
```

Figure 17 Additional Slave For Scalability On Batch Of 5 or More Images



```
def prcs_large_batch():
    print(' -------- Distributing Large Batches -------- ')
    workloads = [ (len(IMAGELIST)) // (size) for i in range(size) ]
    #print(workloads)
    for i in range( len(IMAGELIST) % (size) ):
        workloads[i] += 1
    start = 0
    end = 0
    operation = selected_operation.get()

    for i in range(1, size):
        if i!=1:
            start =end
        #print(start+" "+rank)
        end = start + workloads[i-1]
        print("Master Assigned Rank/VM: ",i, "Batch Starting From: ",start, " To: ",end)
        try:
            comm.send((IMAGELIST[start:end][:][:],start,end,operation),dest=i)
        except Exception as e:
            print(f"Error while sending data to node {i}: {e}")
            redistribute_workload_batch(operation, start, end)  # Resend the task to another node


    for source in range(1,size):
        try:
            result,start,end = comm.recv(source=source)
            print("Master with rank ",rank,"Recieved From",source,"Batch Starting From: ",start, " To: ",end)
            IMAGELIST[start:end]=result
        except MPI.Exception as e:
            print(f"Error while receiving data from a worker node: {e}")
            # Handle error and attempt to recover or redistribute workload
            redistribute_workload_batch(operation, start, end)  # Resend the task to another node

    if (name=='Master'):
        answer = subprocess.check_output(['./testSlave3']).decode().strip()
        if(answer=='Go'):
            print('Waking up Slave 3')
            info = MPI.Info.Create()
            info.Set("add-host","Slave3")
            childArgs =["merged2.py","1"]
            start=end
            end= len(IMAGELIST)
            print("Master Assigned Rank/VM: Slave3 Batch Starting From: ",start, " To: ",end)
            icomm = MPI.COMM_SELF.Spawn(sys.executable, args=childArgs, info=info)
            icomm.send((IMAGELIST[start:end][:][:],start,end,operation),dest=0)
            result,start,end = icomm.recv(source=MPI.ANY_SOURCE,tag=20)
            print("Master with rank ",name,"Recieved From Slave3 Batch Starting From: ",start, " To: ",end)
            IMAGELIST[start:end]=result

    print("Master Done Merging Batches")
    display_images()
```

Figure 18 Process Large Batch Function Responsible For Implementing Scalability



```
def prcs_batch():
    global IMAGELIST
    if((len(IMAGELIST))>=5):
        prcs_large_batch()
        return
    print(' -------- Distributing Batches -------- ')
```

Figure 19 Editing Process Batch Function To Detect Large Batches

Scalability Mechanism

- Batch of size 5 or more triggers scaling
- Master spawns new communication channel to communicate with slave 3 (The additional node)
- Workload is distributed among the three slaves
- On normal batches slave 3 will not process images

Fault-Tolerance Mechanism

- A watcher check every three seconds if python process is active
- If not it will run test script where nodes are sshed and hosts file will be edited to include only healthy nodes
- Thus on fault when system crashed it will reopen again with a healthy hosts file

References

[1] https://mpi4py.readthedocs.io/en/stable/

[2] https://learn.microsoft.com/en-us/azure/virtual-machines/linux/use-remote-desktop?tabs=azure-cli