



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) 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.

Table of Contents

Introduction	2
Task Breakdown Structure:	5
Beneficiaries of The Project.....	5
Detailed Analysis	6
1- Requirements:	6
2- Risks:	6
3- Feasibility:	6
Detailed Project Description	6
1- Objectives and Implementation:.....	6
2- Deliverables:	6
Roles of Each Member	7
System Architecture	7
Testing Scenarios.....	7
End User Guide	8
Sample as an End-User.....	8
Conclusion	11
References	12
Appendices	12

Table of Figures

Figure 1: System Architecture.....	7
Figure 2: Uploading Image or Batch.	8
Figure 3: Image Uploaded.....	8
Figure 4: Image Processed.	9
Figure 5: Image Downloaded.	9
Figure 6: Batch Uploaded.	9
Figure 7: Batch Uploaded.	10
Figure 8: Batch Downloaded.....	10
Figure 9 Image Processing Operations.....	12
Figure 10 Code.....	13
Figure 11 Code.....	14
Figure 12 Distributing Workload and Achieving Fault Tolerance	15
Figure 13 Redistributing Workload and Distributing Batch Workload	16
Figure 14 Redistributing Workload for Fault Tolerance.....	17
Figure 15 Code.....	17
Figure 16 Worker Nodes	18

Table of Figures

Table 1 Member Roles	7
----------------------------	---

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. Then Repeating 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 slaves 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
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

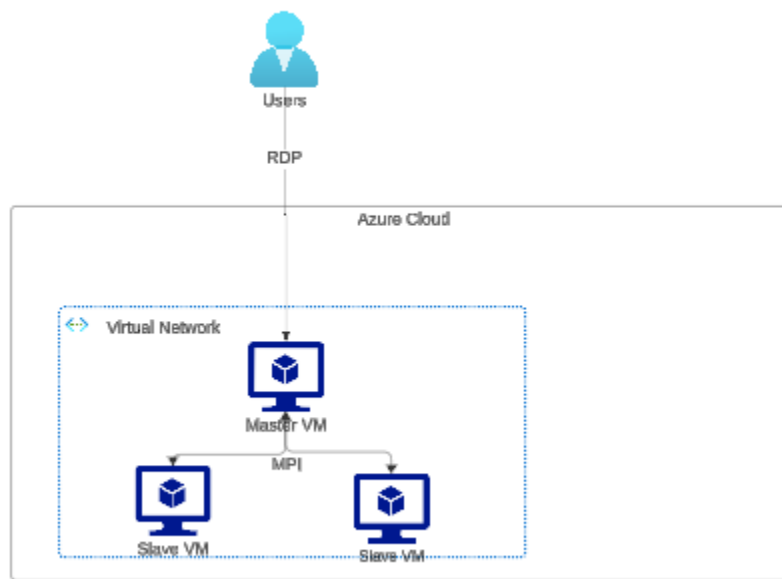


Figure 1: System Architecture.

- 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

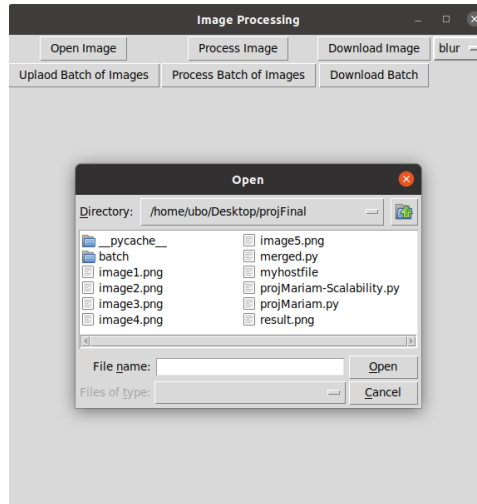


Figure 2: Uploading Image or Batch.

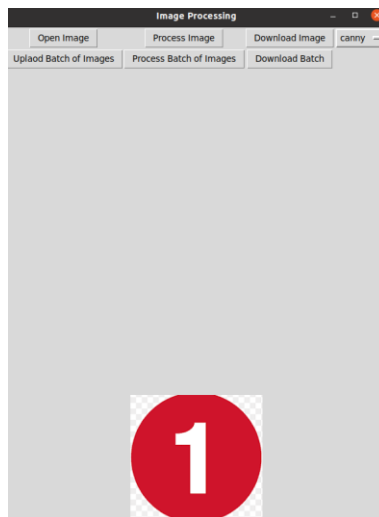


Figure 3: Image Uploaded.

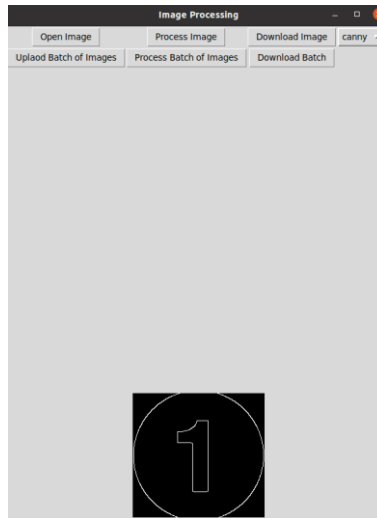


Figure 4: Image Processed.



result.png

Figure 5: Image Downloaded.

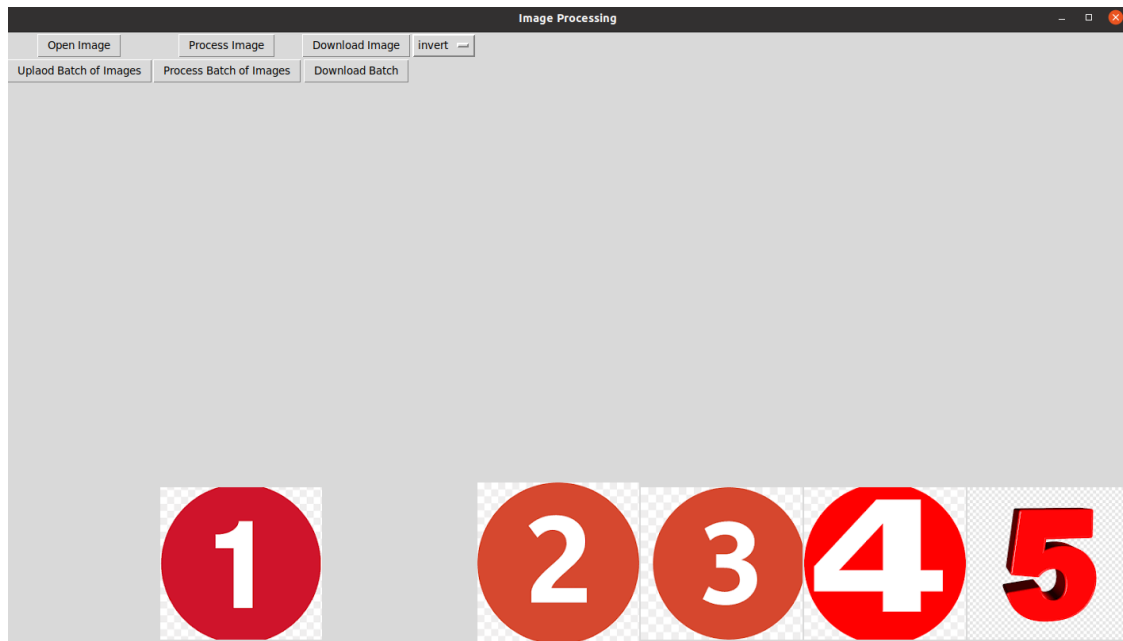


Figure 6: Batch Uploaded.

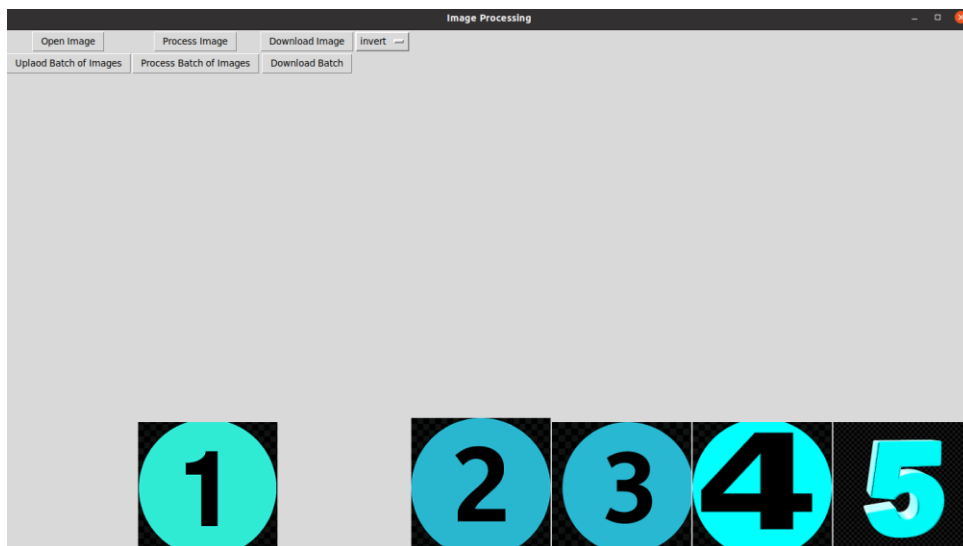


Figure 7: Batch Uploaded.

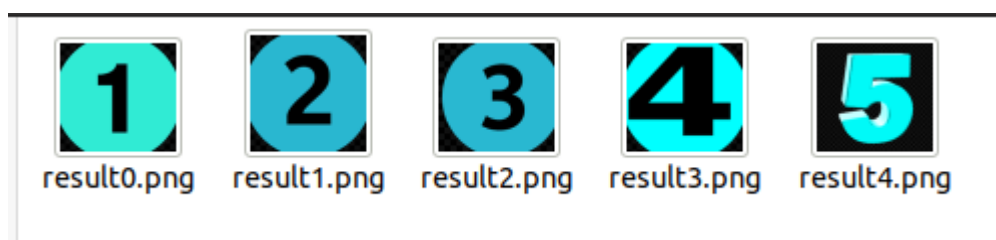


Figure 8: Batch Downloaded.

```

----- 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.

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>

Appendices

Code for Image Operations

```
1 import tkinter as tk
2 from tkinter import filedialog
3 from PIL import Image, ImageTk
4 import cv2
5 import numpy as np
6 from mpi4py import MPI
7 import os
8 import math
9 import zlib
10 import io
11 import time
12
13
14 IMAGE=np.zeros((1,1,1))
15 IMAGELIST=[]
16 all_labels = []
17
18
19 # Function to apply various image processing operations
20 def apply_image_processing(image, operation):
21     if operation == 'blur':
22         result = cv2.blur(image, (5, 5)) # Applying Gaussian blur
23     elif operation == 'gaussian':
24         result = cv2.GaussianBlur(image, (5, 5), 0) # Applying Gaussian blur
25     elif operation == 'median':
26         result = cv2.medianBlur(image, 5) # Applying median blur
27     elif operation == 'bilateral':
28         result = cv2.bilateralFilter(image, 9, 75, 75) # Applying bilateral filter
29     elif operation == 'canny':
30         result = cv2.Canny(image, 100, 200) # Applying Canny edge detection
31     elif operation == 'invert':
32         result = cv2.bitwise_not(image) # Invert colors
33     elif operation == 'brightness_increase':
34         result = cv2.convertScaleAbs(image, alpha=1.5, beta=0) # Increase brightness
35     elif operation == 'to_gray':
36         result = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) # Convert image to grayscale
37     elif operation == 'to_bw':
38         _, result = cv2.threshold(cv2.cvtColor(image, cv2.COLOR_BGR2GRAY), 127, 255, cv2.THRESH_BINARY) # Convert image to black and white
39     elif operation == 'contrast_stretching':
40         hist, bins = np.histogram(image.flatten(), 256, [0, 256])
41         cdf = hist.cumsum()
42         cdf_m = np.ma.masked_equal(cdf, 0)
43         cdf_m = (cdf_m - cdf_m.min()) * 255 / (cdf_m.max() - cdf_m.min())
44         cdf = np.ma.filled(cdf_m, 0).astype('uint8')
```

Figure 9 Image Processing Operations

```

45     result = cdf[image]
46 else:
47     result = image # No operation
48
49     return result
50
51 # Function to open file dialog and load image
52 def open_files():
53     files= filedialog.askopenfilenames()
54     #print(root.tk.splitlist(files))
55     global IMAGELIST
56     IMAGELIST=[]
57     for file in root.tk.splitlist(files):
58         img = cv2.imread(file)
59         IMAGELIST.append(img)
60         display_images_from_files(files)
61
62 def open_file():
63     file_path = filedialog.askopenfilename()
64     if file_path:
65         global IMAGE
66         IMAGE = cv2.imread(file_path)
67         display_image(IMAGE)
68
69 # Function to save processed image
70 def save_file():
71     global IMAGE
72     cv2.imwrite("result.png", IMAGE)
73     print("Image saved successfully!")
74
75 def save_batch():
76     global IMAGELIST
77     j=0
78     if not os.path.exists("batch"):
79         os.makedirs("batch")
80     for i in IMAGELIST:
81         path="batch/result"+str(j)+".png"
82         j+=1
83         cv2.imwrite(path, i)
84     print("Image Batch saved successfully!")
85
86 def display_image(image):
87     image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)

```

Figure 10 Code

```

88 image = Image.fromarray(image)
89 imageTk = ImageTk.PhotoImage(image)
90
91 for label in all_labels:
92     label.destroy()
93
94 i=0
95 label = tk.Label(image=imageTk)
96 label.photo = imageTk # assign to class variable to resolve problem with bug in `PhotoImage`
97
98 label.grid(row=2, column=i)
99 all_labels.append(label)
100
101 def display_images_from_files(files):
102     for label in all_labels:
103         label.destroy()
104     i=0
105     for file in files:
106         image = ImageTk.PhotoImage(Image.open(file))
107         label = tk.Label(image=image)
108         label.photo = image # assign to class variable to resolve problem with bug in `PhotoImage`
109
110         label.grid(row=2, column=i)
111         i+=1
112         all_labels.append(label)
113
114 def display_images():#display IMGLIST
115     global IMAGELIST
116
117     for label in all_labels:
118         label.destroy()
119     i=0
120     for image in IMAGELIST:
121         image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
122         image = Image.fromarray(image)
123         imageTk = ImageTk.PhotoImage(image)
124         label = tk.Label(image=imageTk)
125         label.photo = imageTk
126
127         label.grid(row=2, column=i)
128         i+=1
129         all_labels.append(label)
130
131 # Function to display image in GUI

```

Figure 11 Code

```

132 def prcs_image():
133     print(' ----- Distributing Processing of One Image ----- ')
134     global IMAGE
135     height, width, channels = IMAGE.shape
136     #print(height,width,channels)
137     workloads = [ (height // (size-1)) for i in range(size-1) ]
138     #print(workloads)
139     for i in range( height % size ):
140         workloads[i] += 1
141
142     #print(workloads)
143     start = 0
144     operation = selected_operation.get()
145
146     for i in range(1, size):
147         if i!=1:
148             start +=workloads[i-1]
149             #print(start+" "+rank)
150             end = start + workloads[i-1]
151             print("Master Assigned Rank/VM: ",i, "Rows Starting From: ",start, " To: ",end)
152             try:
153                 comm.send((IMAGE[start:end][:][:],start,end,operation),dest=i)
154             except MPI.Exception as e:
155                 print(f"Error while sending data to node {i}: {e}")
156                 redistribute_workload_image(operation, start, end) # Resend the task to another node
157
158     for source in range(1,size):
159         try:
160             result,start,end = comm.recv(source=source)
161             print("Master with rank ",rank,"Recieved From",source,"Rows Starting From: ",start, " To: ",end)
162             IMAGE[start:end][:][:]=result[:][:][:]
163         except MPI.Exception as e:
164             print(f"Error while receiving data from a worker node: {e}")
165             # Handle error and attempt to recover or redistribute workload
166             redistribute_workload_image(operation, start, end) # Resend the task to another node
167     print("Master Done Merging Results")
168     '''cv2.imshow("proc.png",IMAGE)
169     cv2.waitKey(0)
170     cv2.destroyAllWindows()'''
171     display_image(IMAGE)
172
173 #Function to redistribute workload for image processing
174 def redistribute_workload_image(operation, start, end):
175     global IMAGE

```

Figure 12 Distributing Workload and Achieving Fault Tolerance

```

176     try:
177         # Find another available worker node
178         for i in range(1, size):
179             if i != rank:
180                 comm.send((True,), dest=i) # Notify the node to prepare for receiving the task
181                 comm.send((IMAGE[start:end], start, end, operation), dest=i) # Send the task
182                 result, _, _ = comm.recv(source=i) # Receive the processed image
183                 IMAGE[start:end] = result
184             return
185     except MPI.Exception as e:
186         print(f"Error while redistributing workload: {e}")
187         # Handle error and attempt to recover or exit gracefully
188
189 def prcs_batch():
190     print(' ----- Distributing Batches ----- ')
191     global IMAGELIST
192     workloads = [ (len(IMAGELIST)) // (size-1) for i in range(size-1) ]
193     for i in range( len(IMAGELIST) % (size-1) ):
194         workloads[i] += 1
195     start = 0
196     end = 0
197     operation = selected_operation.get()
198
199     for i in range(1, size):
200         if i!=1:
201             start =end
202             #print(start+" "+rank)
203             end = start + workloads[i-1]
204             print("Master Assigned Rank/VM: ",i, "Batch Starting From: ",start, " To: ",end)
205             try:
206                 comm.send((IMAGELIST[start:end][:][:],start,end,operation),dest=i)
207             except MPI.Exception as e:
208                 print(f"Error while sending data to node {i}: {e}")
209                 redistribute_workload_batch(operation, start, end) # Resend the task to another node
210
211
212     for source in range(1,size):
213         try:
214             result,start,end = comm.recv(source=source)
215             print("Master with rank ",rank,"Recieved From",source,"Batch Starting From: ",start, " To: ",end)
216             IMAGELIST[start:end]=result
217         except MPI.Exception as e:
218             print(f"Error while receiving data from a worker node: {e}")
219             # Handle error and attempt to recover or redistribute workload

```

Figure 13 Redistributing workload and Distributing Batch workload


```

220     redistribute_workload_batch(operation, start, end) # Resend the task to another node
221     print('Master Done Merging Batches')
222     display_images()
223     '''cv2.imshow("prcsd.png",IMAGE)
224     cv2.waitKey(0)
225     cv2.destroyAllWindows()
226     display_image(IMAGE)'''
227
228 # Function to redistribute workload for batch image processing
229 def redistribute_workload_batch(operation, start, end):
230     global IMAGELIST
231     try:
232         # Find another available worker node
233         for i in range(1, size):
234             if i != rank:
235                 comm.send((True,), dest=i) # Notify the node to prepare for receiving the task
236                 comm.send((IMAGELIST[start:end], start, end, operation), dest=i) # Send the task
237                 result, _, _ = comm.recv(source=i) # Receive the processed images
238                 IMAGELIST[start:end] = result
239             return
240     except MPI.Exception as e:
241         print(f"Error while redistributing workload: {e}")
242         # Handle error and attempt to recover or exit gracefully
243
244 # MPI CODE
245 comm = MPI.COMM_WORLD
246 rank = comm.Get_rank()
247 size = comm.Get_size()
248
249
250 if rank == 0: # Master node
251 # Create main GUI window
252     root = tk.Tk()
253     root.title("Image Processing")
254
255 # Create frame for buttons
256     button_frame = tk.Frame(root)
257     button_frame.grid(row=0,column=0)
258
259 # Create button to open file dialog
260     open_button = tk.Button(button_frame, text="Open Image", command=open_file)
261     open_button.grid(row=0, column=1)
262
263 # Create button to download processed image

```

Figure 14 Redistributing Workload for Fault Tolerance

```

264     upload_imgs_button = tk.Button(button_frame, text="Upload Batch of Images", command=open_files)
265     upload_imgs_button.grid(row=1, column=1)
266
267
268     prcs_button = tk.Button(button_frame, text="Process Image", command=prcs_image)
269     prcs_button.grid(row=0, column=2)
270
271     prcs_batch_button = tk.Button(button_frame, text="Process Batch of Images", command=prcs_batch)
272     prcs_batch_button.grid(row=1, column=2)
273
274
275 # Create button to download processed image
276     download_button = tk.Button(button_frame, text="Download Image", command=save_file)
277     download_button.grid(row=0, column=3)
278
279     download_button = tk.Button(button_frame, text="Download Batch", command=save_batch)#!!!!add cmd
280     download_button.grid(row=1, column=3)
281
282
283
284 # Create dropdown menu for selecting operations
285     operations = ['blur', 'gaussian', 'median', 'bilateral', 'canny', 'invert', 'brightness_increase', 'to_gray', 'to_bw', 'contrast_stretching']
286     selected_operation = tk.StringVar(root)
287     selected_operation.set(operations[0])
288     operation_menu = tk.OptionMenu(button_frame, selected_operation, *operations)
289     operation_menu.grid(row=0, column=4)
290
291     # Create canvas to display images
292     canvas = tk.Canvas(root, width=500, height=500)
293     canvas.grid(row=1, column=0)
294
295     canvas_image = canvas.create_image(0, 0, anchor=tk.NW, image=None) # Initialize canvas image
296
297     root.mainloop()
298
299 else: # Worker nodes
300
301     #img = img[height//2][::]
302     while True:
303         img, start, end, operation = comm.recv(source=0)
304
305         if (type(img)==type([])): # Batch Images
306             print("Rank/VM: ",rank, "Batch Rows Starting From: ",start, " To: ",end)
307             result=[]

```

Figure 15 Code

```

308     j=0
309     for i in img:
310         print("Rank/VM: ",rank, "Finished: ",j, " /",end-start)
311         j+=1
312         prcsd= apply_image_processing(i,operation)
313         if operation in ['to_bw','to_gray','canny']:
314             prcsd = cv2.cvtColor(prcsd, cv2.COLOR_GRAY2BGR)
315         result.append(prcsd)
316
317         #cv2.imshow(str(rank)+"-"+str(i)+".png",i)
318         #cv2.waitKey(0)
319     #cv2.destroyAllWindows()
320     print("Rank/VM: ",rank, "Finished: ",j, " /",end-start)
321     comm.send((result,start,end), dest=0)
322     print("Rank/VM: ",rank, "Finished and Send Batch Starting From: ",start, " To: ",end)
323
324 else: #1 Imge
325     print("Rank/VM: ",rank, "Received Rows Starting From: ",start, " To: ",end)
326     result= apply_image_processing(img,operation)
327     if operation in ['to_bw','to_gray','canny']:
328         result = cv2.cvtColor(result, cv2.COLOR_GRAY2BGR)
329
330     comm.send((result,start,end), dest=0)
331     print("Rank/VM: ",rank, "Finished and Send Rows Starting From: ",start, " To: ",end)
332     '''cv2.imshow("prcsd.png",result)
333     cv2.waitKey(0)
334     cv2.destroyAllWindows()'''
335

```

Figure 16 Worker Nodes