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

1- 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.

Youtube Link https://youtu.be/xe20gTOvc8E

Github Link https://github.com/XXMariamX/Distributed-Project

Table of Contents

Table of Figures

Table of Figures

2- 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 the common files to be shared 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.
    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.

3- 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.

4- 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.

5- 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?

6- 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

7- System Architecture:

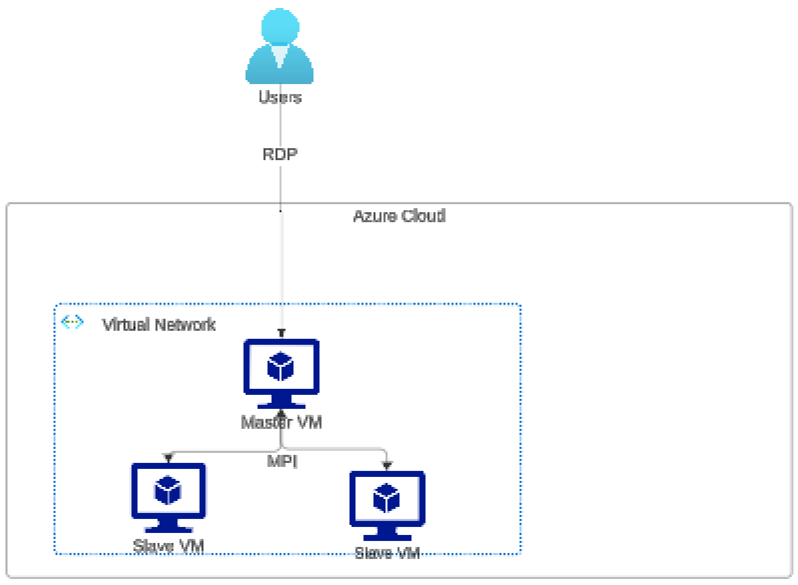

Figure 1: System Architecture.

- 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.

8- 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.

9- 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

Sample as an End-User

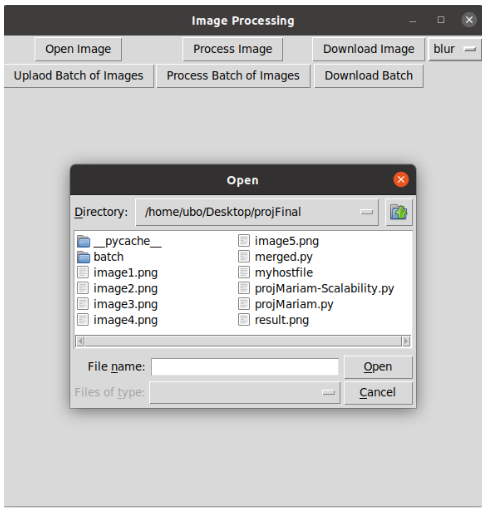

Figure 2: Uploading Image or Batch.

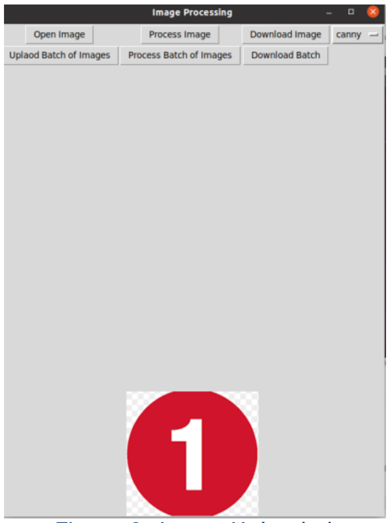

Figure 3: Image Uploaded.

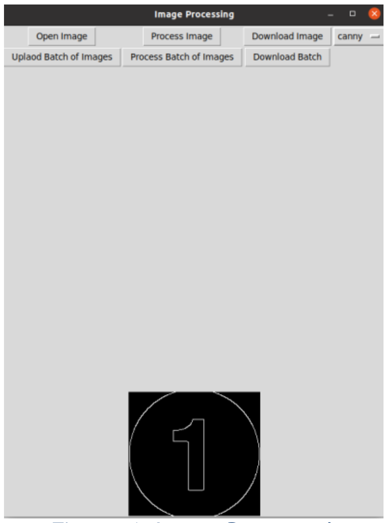

Figure 4: Image Processed.

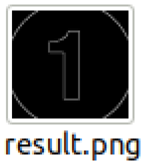

Figure 5: Image Downloaded.

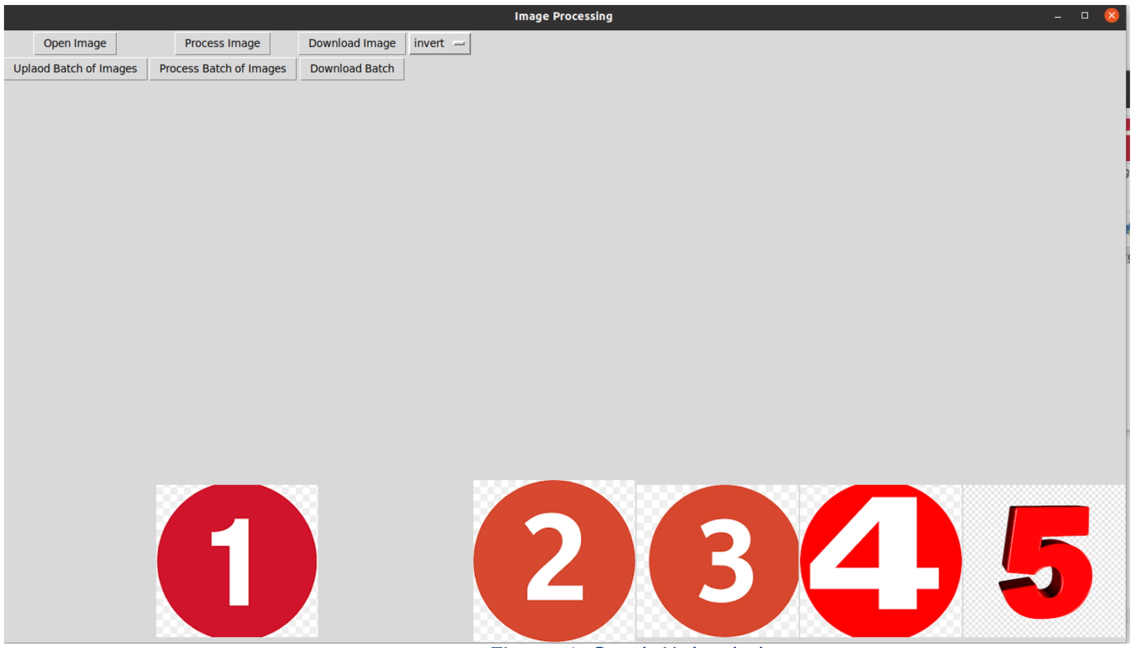

Figure 6: Batch Uploaded.

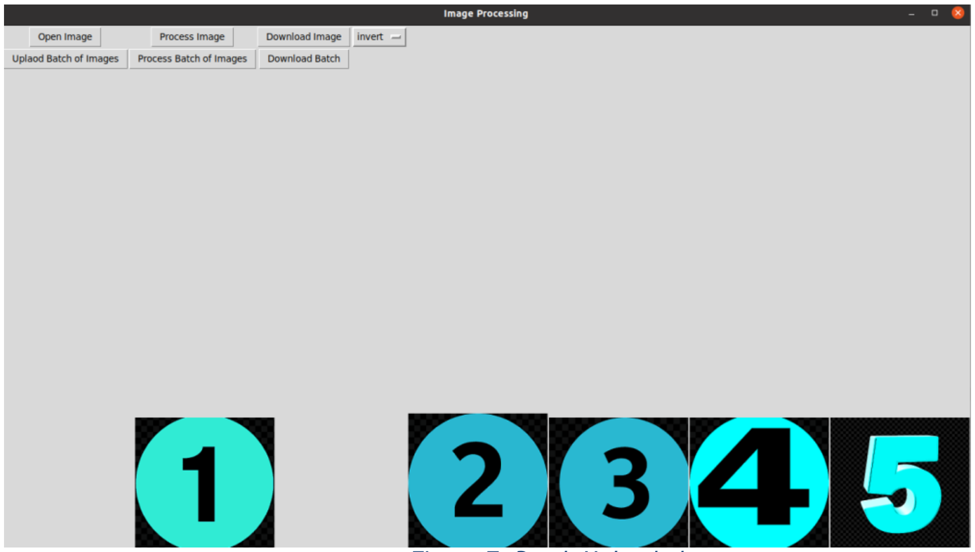

Figure 7: Batch Uploaded.

Figure 8: Batch Downloaded.

10- Conclusion:

Our application provides:

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

11- 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