AIN SHAMS UNIVERSITY

FACULTY OF ENGINEERING

CSE354: Distributed Computing

Computer Engineering and Software Systems

Distributed Image Processing System using Cloud Computing

Submitted to

Prof. Ayman Bahaa Eldin

Prepared by Team 33

Yousif Mohamed Mousa	**20P5383**
Mohaned Hesham Nasr	**20P2204**
Abdallah Mohamed Elkhalafawy	**20P4739**
Mohamed Hany Berisha	**20P4587**

Table of Contents

1. Introduction
2. Detailed Project Description
3. Beneficiaries of the Project
4. Detailed Analysis
5. Task Breakdown Structure
6. Roles
7. System Architecture and Design
8. Testing Scenarios and Results
9. End-User Guide
10. Conclusion
11. Refrences
12. Diagrams

Introduction :

This project aims to develop a distributed image processing system utilizing cloud computing technologies. The system is designed to process large-scale images efficiently by leveraging the parallel processing capabilities of MPI (Message Passing Interface) and the scalable infrastructure provided by AWS services such as S3, SQS, and EC2. This project demonstrates the effective use of cloud resources for computational tasks, providing an example of how modern technology can enhance traditional image processing techniques.

Detailed Project Description

Objective

The primary objective of this project is to design and implement a distributed system capable of processing images using various operations like edge detection, color inversion, blurring, erosion, and dilation. The system distributes the workload across multiple virtual machines in a cloud environment, ensuring scalability and efficiency.

Components

- AWS S3: Storage for images and processed results.
- AWS SQS: Queue service for task distribution.
- AWS EC2: Virtual machines for running the distributed processing tasks.
- MPI: For parallel processing and communication between nodes.

Workflow

1. User Interaction: Users upload images and select processing operations via a GUI.
2. Task Distribution: The master node uploads the images to S3 and sends tasks to the SQS queue.
3. Image Processing: Worker nodes retrieve tasks from the queue, download images from S3, process them, and upload the results back to S3.
4. Result Notification: Worker nodes notify the master node about the completion of tasks, which then displays the processed images to the user.

Beneficiaries of the Project

The primary beneficiaries of this project include:

- Academic Institutions: For teaching and research in distributed systems and image processing.
- Businesses: Companies that require scalable image processing solutions.
- Developers: Open-source community and developers interested in distributed computing.

Detailed Analysis

Requirements:

Functional Requirements:

- Upload images to the system.
- Select image processing operations.
- Distribute tasks among multiple worker nodes.
- Collect and display processed images.

Non-Functional Requirements:

- Scalability: Handle increasing number of images and operations.
- Fault Tolerance: Ensure system reliability in case of node failures.
- Performance: Efficiently process images in parallel.

Tools and Libraries:

- Boto3: AWS SDK for Python.
- Tkinter: Python library for GUI.
- MPI4Py: Python bindings for MPI.
- OpenCV: Image processing library.
- ‘threading' and ‘mpi4py’ for concurrency and parallel processing.
- ‘paramiko' for SSH connections.
- ‘json' for parsing JSON data.

Task Breakdown Structure

Project Initialization:

- Define project scope.
- Set up version control (Git).

GUI Development:

- Design user interface.
- Implement file upload functionality.
- Implement operation selection functionality.

Master Node Implementation:

- Set up S3 and SQS clients.
- Implement task distribution logic.

Worker Node Implementation:

- Implement image processing functions.
- Implement task reception and processing.

Integration and Testing:

- Integrate GUI with backend.
- Test end-to-end workflow.

Deployment:

- Deploy on AWS.
- Monitor and optimize performance.

Roles:

Yousif: Lead master node implementation and testing.

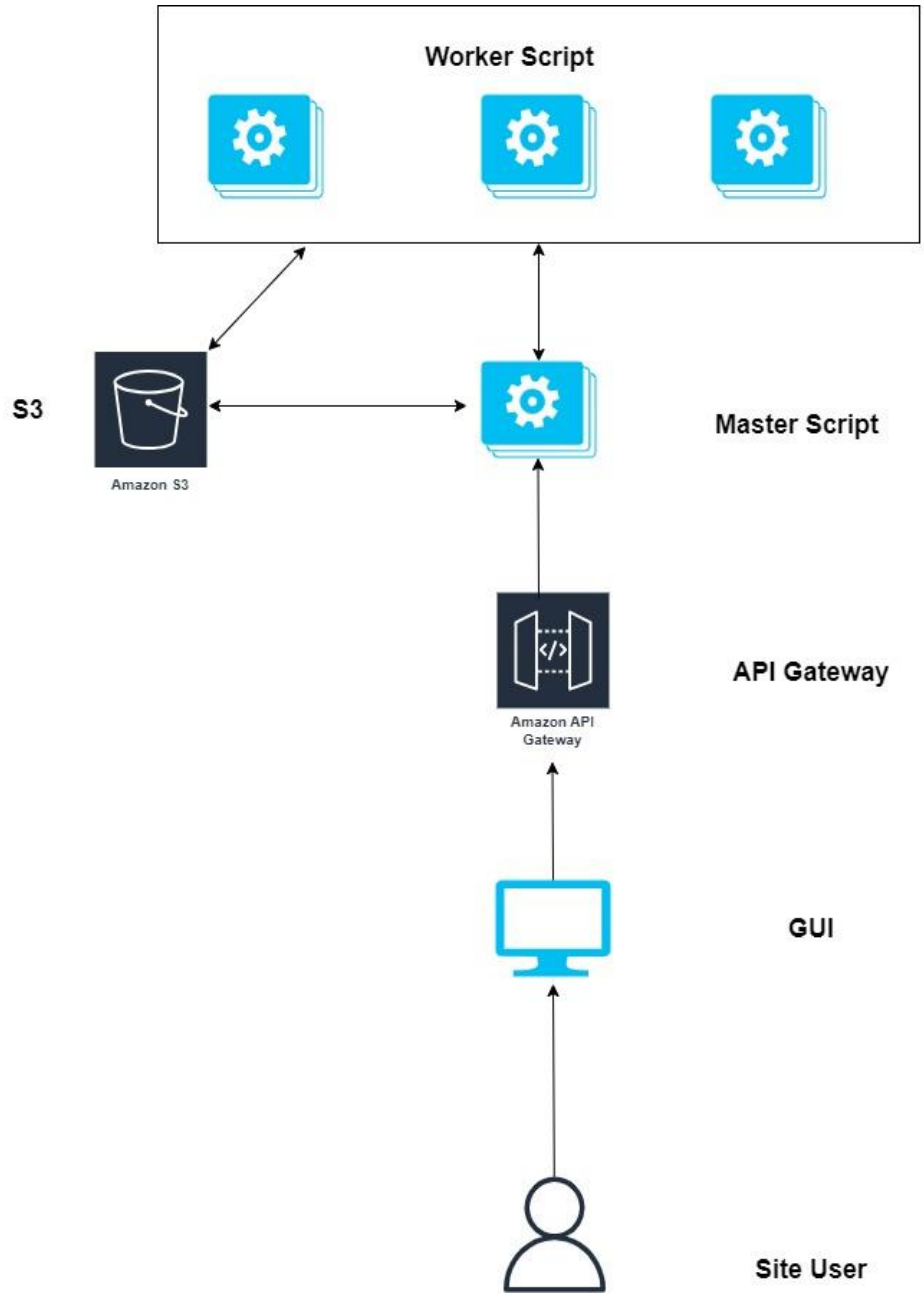
Mohaned: Lead worker node implementation and deployment.

Abdallah : Lead GUI development .

Mohamed : Lead GUI integration.

System Architecture and Design

Architecture Diagram :



Description:

The system follows a distributed architecture where the master node orchestrates the workflow, and multiple worker nodes process tasks in parallel. The GUI interacts with the master node to initiate processing.

Testing Scenarios and Results

Testing Scenarios:

Functional Testing:

- Upload and process a single image.
- Upload and process multiple images.

Performance Testing:

- Measure processing time for different operations.
- Test scalability with increasing number of worker nodes.

Results:

- Functional tests passed successfully with expected results.
- Performance tests indicated linear scalability with additional nodes.

End-User Guide

Installation:

- Install required Python packages.
- Configure AWS credentials.

Running the Application:

- Start the master node: python master.py
- Start the worker nodes: mpiexec -n <number_of_nodes> python worker.py

- Run the GUI application: python gui.py

Using the Application:

- Upload images using the GUI.
- Select desired operations.
- View processed images in the GUI.

Conclusion:

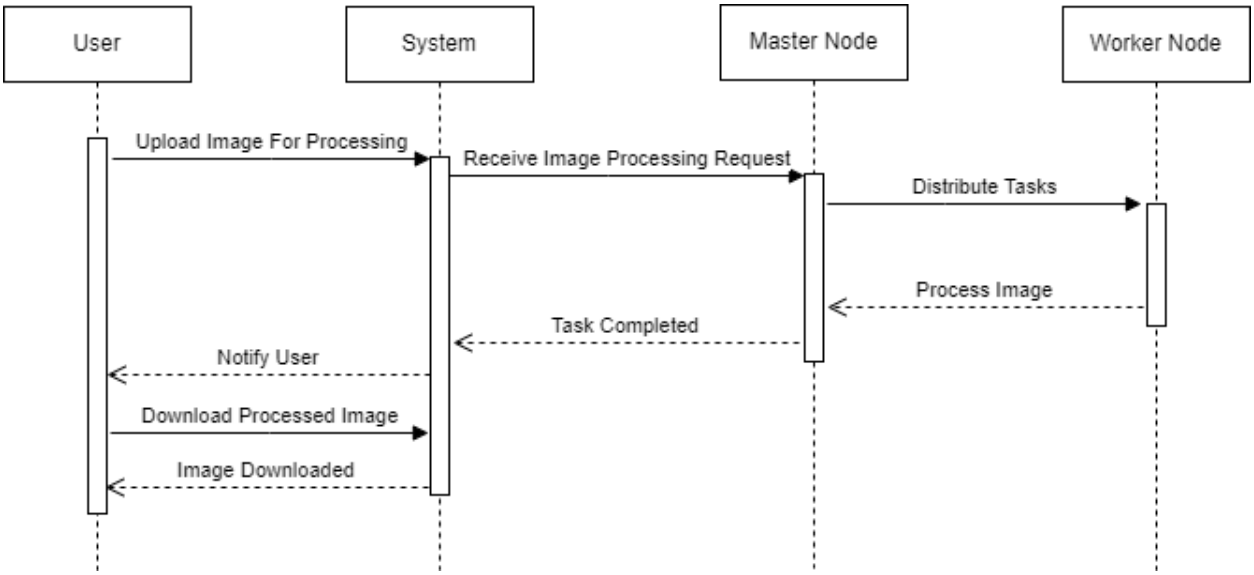
The distributed image processing system successfully demonstrates scalable and fault-tolerant image processing using AWS and MPI. The project provides a robust solution for distributed image processing tasks, beneficial for various users including academic institutions and businesses.

Refrence:

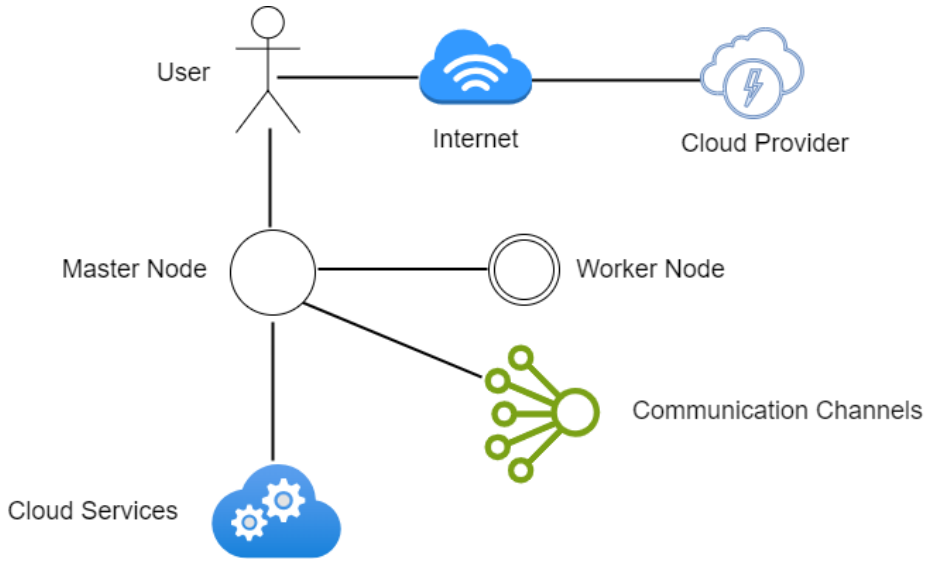
- https://medium.com/analytics-vidhya/how-to-access-aws-s3-using-boto3-python-sdk-e5fbd3d276bd
- https://mpitutorial.com/tutorials/running-an-mpi-cluster-within-a-lan/
- https://mpi4py.readthedocs.io/en/stable/
- https://docs.opencv.org/4.x/
- https://boto3.amazonaws.com/v1/documentation/api/latest/index.html
- https://docs.aws.amazon.com/

Diagrams:

Sequence Diagram:



Network Diagram:



Component Diagram: