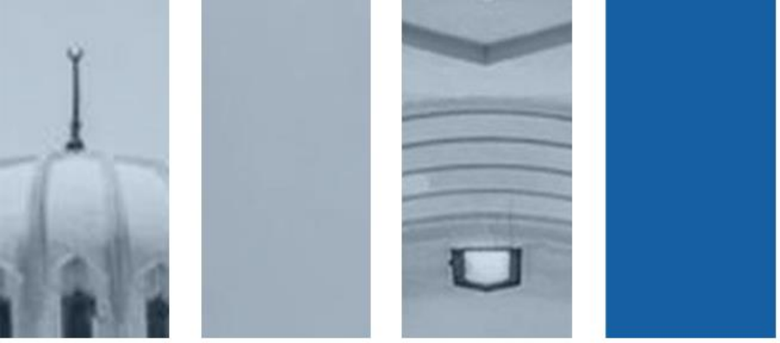

Distributed Image Processing System using Cloud

Program: Computer Engineering and Software Systems

Course Code: CSE354

Course Name: Distributed Computing

Submitted to

Prof. Ayman M. Bahaa-Eldin

Ain Shams University

Faculty of Engineering

Spring Semester – 2024

Team 19

Personal Information

20P3485	**Mohamed Amr EL Mallah**
20P8449	**Mohamed Ibrahim Elsayed Barakat**
20P7105	**Salma Nasrelden Aboelela Hendawy**
20P3844	**Youssef Emad Eldin Elshahat Barakat**

Distributed Image Processing System using Cloud Computing

Table of contents:

Table of figures:

Introduction:

In today's digital era, the demand for image processing applications continues to rise, driven by various fields such as healthcare, entertainment, surveillance, and more. However, the computational complexity of image processing tasks often poses challenges in terms of processing time and resource utilization. To address these challenges, the integration of cloud computing and distributed systems has emerged as a powerful solution, enabling efficient parallel processing of image data.

The "Distributed Image Processing System using Cloud Computing" project aims to leverage the scalability and computational resources offered by cloud environments to implement a robust and efficient image processing system. By distributing processing tasks across multiple virtual machines in the cloud, the system can handle large volumes of image data effectively while ensuring scalability and fault tolerance.

This project focuses on developing a distributed system using Python programming language and cloud-based virtual machines. The system will utilize either OpenCL or MPI (Message Passing Interface) for parallel processing of image data, enabling the implementation of various image processing algorithms such as filtering, edge detection, and color manipulation.

Project Scope:

The project aims to develop a distributed image processing system utilizing cloud computing technologies. It involves designing and implementing a system capable of distributing image processing tasks across multiple virtual machines in the cloud. The system will support various image processing algorithms such as filtering, edge detection, and color manipulation. It should be scalable to accommodate an increasing workload by adding more virtual machines and should maintain fault tolerance to handle node failures gracefully.

Objectives:

- Design and implement a distributed image processing system using Python.
- Utilize cloud-based virtual machines for distributed computing.

- Use image processing algorithms including filtering, edge detection, and color manipulation.
- Ensure scalability to accommodate increased workload by adding virtual machines dynamically.
- Ensure fault tolerance by reassigning tasks from failed nodes to operational ones.

Requirements:

- Distributed Processing: The system should distribute image processing tasks across multiple virtual machines in the cloud.
- Image Processing Algorithms: Use filtering, edge detection, and colour manipulation algorithms.
- Scalability: The system should dynamically scale by adding virtual machines as the workload increases.
- Fault Tolerance: The system should handle node failures gracefully by reassigning tasks from failed nodes to operational ones.
- User Interface: Develop a user-friendly interface for users to upload images, select processing operations, monitor task progress, and download processed images.
- Cloud Computing Platform: Select a suitable cloud computing platform (e.g., AWS, Azure, Google Cloud) for hosting virtual machines.
- Parallel Processing Framework: Choose either OpenCL or MPI for parallel processing of image data.
- Monitoring System: Implement a monitoring system to track the progress of image processing tasks.
- Documentation: Provide comprehensive documentation including system architecture, setup instructions, and user guide.

Beneficiaries of the project:

1. **Researchers and Academia:** Researchers and academics involved in fields such as computer vision, image processing, and distributed systems can benefit from the project's advancements. The system provides a platform for exploring and experimenting with different image processing algorithms in a distributed computing environment, enabling them to conduct research and develop new techniques more efficiently.

2. **Healthcare Professionals:** In the healthcare industry, medical imaging plays a crucial role in diagnosis, treatment planning, and monitoring of patients. Healthcare professionals can benefit from the project by utilizing the distributed image processing system to enhance the speed and accuracy of medical image analysis. This can lead to faster diagnoses, improved treatment outcomes, and ultimately better patient care.
3. **Entertainment and Media Industry:** The entertainment and media industry often deals with large volumes of image and video data for tasks such as video editing, special effects, and content creation. The distributed image processing system can streamline these workflows by providing efficient parallel processing capabilities, enabling content creators to produce high-quality media content more effectively.
4. **Surveillance and Security Agencies:** Surveillance systems rely heavily on image processing technologies for tasks such as object detection, tracking, and facial recognition. By leveraging the distributed image processing system, surveillance and security agencies can enhance the capabilities of their surveillance systems, improving situational awareness and response times in critical situations.
5. **E-commerce and Retail:** E-commerce platforms and retail businesses can benefit from the project by integrating image processing capabilities for tasks such as product recognition, image-based search, and visual recommendation systems. The distributed system enables real-time processing of images, enhancing the user experience and driving sales through personalized product recommendations.
6. **Government and Public Sector:** Government agencies and public sector organizations can leverage the distributed image processing system for various applications, including satellite image analysis, urban planning, disaster management, and environmental monitoring. By processing large-scale image data efficiently, these organizations can make data-driven decisions and address societal challenges more effectively.

User Stories:

- As a user, I want to upload an image to the system for processing.
- As a user, I want to select the type of image processing operation to be performed.
- As a user, I want to download the processed image once the operation is complete.
- As a user, I want to monitor the progress of the image processing task.

System Architecture:

1. User Interface (UI):
    - Provides an interface for users to interact with the system.
    - Allows users to upload images, select processing operations, monitor task progress and view the processed images.
2. Master Node:
    - Virtual machine in the cloud responsible for the application backend before the image processing.
    - Handles user requests and generates messages to the worker nodes.
    - Divide the image into many segments using image segmentation methods.
    - Distributes image processing tasks to worker nodes.
    - Manages scalability and fault tolerance.
    - Sends back the processed image to the client UI.
3. Worker Nodes:
    - Virtual machines in the cloud responsible for actual image processing using rpc architecture.
    - Receive tasks from the backend (master node), perform processing using parallel computing, and return results.
4. Communication Layer:
    - Facilitates communication between different components of the system using TCP and web sockets to RPC communication and we will define it below.
    - Utilizes messaging protocols or frameworks for task distribution and result retrieval.
5. Monitoring:
    - Monitors system performance, resource utilization, and task progress.

Selected Technologies:

1. Cloud Platform:
    - **Microsoft Azure**: Cloud provider with virtual machines (Azure VMs), storage (Azure Blob Storage), and messaging services (Azure Service Bus).
2. Programming Language:
    - **Python**: Selected for its ease of development, rich ecosystem of libraries (e.g., CV for image processing), and suitability for parallel computing.
3. Parallel Computing Framework:
    - **MPI (Message Passing Interface)**: Enables distributed computing and communication between worker nodes.
4. Communication:
    - RPC with TCP and WebSockets

Considerations:

- **Scalability**: Ensure the architecture can scale horizontally by adding more worker nodes dynamically.
- **Fault Tolerance**: Implement mechanisms to handle node failures, such as task reassignment and redundancy.
- **Cost Optimization**: Optimize resource usage to minimize operational costs, especially in cloud environments where costs can scale with usage.

Why we used these technologies:

1. Cloud Platform - Microsoft Azure:
    - **Azure VMs**: These provide scalable and customizable virtual machines, allowing the deployment of various applications without worrying about hardware infrastructure.
    - **Azure Blob Storage**: Offers scalable object storage for documents, images, videos, and other unstructured data, enabling efficient data management and access.

    Reason for Selection: Microsoft Azure was chosen for its robust infrastructure, extensive services, and reliable performance. It offers a wide range of scalable solutions that fit the project's requirements, ensuring flexibility and efficiency in deployment and management.

2. Programming Language - Python:
   - **Ease of Development**: Python's simple and readable syntax makes it easy to write and maintain code, accelerating development cycles.
   - **Rich Ecosystem of Libraries**: Python boasts a vast collection of libraries for various purposes, such as computer vision (CV) for image processing, machine learning, data analysis, and more. This wealth of resources enhances productivity and facilitates the implementation of complex functionalities.
   - **Suitability for Parallel Computing**: Python supports parallel computing through frameworks like MPI, enabling efficient utilization of resources and faster processing of tasks.

   Reason for Selection: Python was chosen for its combination of simplicity, versatility, and powerful libraries. Its suitability for parallel computing aligns well with the project's requirements for efficient data processing and analysis.

3. Parallel Computing Framework - MPI (Message Passing Interface):
   - **Distributed Computing**: MPI enables efficient communication and coordination between multiple processes running on distributed computing systems, allowing for parallel execution of tasks.
   - **Scalability**: It supports scaling across multiple nodes, enabling the system to handle large datasets and compute-intensive workloads effectively.
   - **Performance**: MPI is known for its high performance and low overhead, making it well-suited for demanding parallel computing tasks.

   Reason for Selection: MPI was chosen for its proven track record in parallel computing, especially in high-performance computing (HPC) environments. It provides the necessary tools and mechanisms for efficient parallelization of algorithms and processing of large datasets, essential for the project's objectives.

Communication:

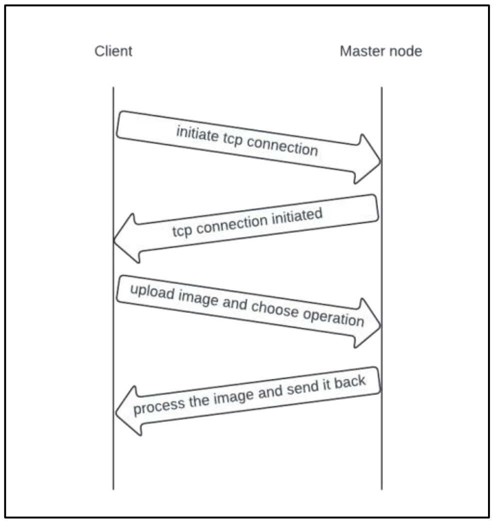
Client
Master node
initiate tcp connection
tcp connection initiated
upload image and choose operation
process the image and send it back

Figure 1 client-master node protocol

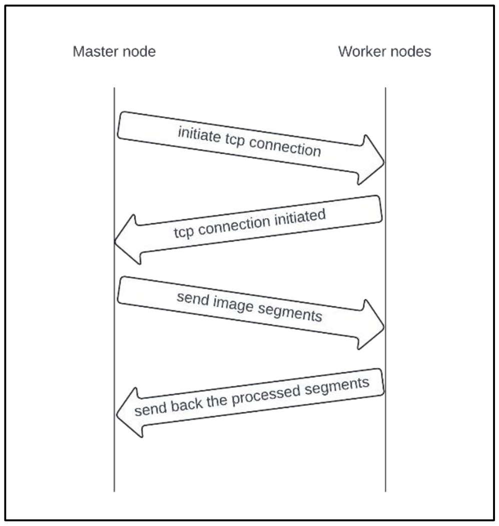

Figure 2 master node-worker node protocol

We will use TCP and WebSockets for RPC Communication

Cost analysis:

Developing a distributed image processing system using cloud computing Application involves various costs, including:

Development Costs:

- **Software Development:**
    - **Resource Time:**
        - Programming (Python) - Analyzing, designing, coding, and testing the application.
        - Network Engineering - Designing and implementing the network architecture.
    - **Tools and Frameworks:**
        - Python development environment (IDE)
        - Specific libraries for networking, parallel computing, and UI/UX
        - Cloud-based development platform
    - **Testing and Quality Assurance:**
        - Unit testing, integration testing, and user acceptance testing
        - Automated testing tools
- **Documentation:**
    - Creating user manuals, API documentation, and internal technical documentation

Infrastructure Costs:

- **Deployment:**
    - Cloud-based server
    - Domain name and SSL certificate
    - Load balancer
- **Hosting:**

    - Monthly or annual fees for cloud server or other hosting services
    - Bandwidth costs depending on user activity

Maintenance Costs:

- Bug Fixes: Addressing issues reported by users
- Feature Enhancements: Implementing new features and functionality
- Security Updates: Maintaining security patches and updates for libraries and frameworks
- Version Control: Managing code changes and releases

Additional Costs:

- Project Management: Planning, scheduling, and coordinating development activities
- Legal and Regulatory Compliance: Ensuring compliance with data privacy regulations
- Third-Party Services: APIs, libraries, or other paid services
- Marketing and Promotion: Advertising and promoting the application to attract users

Cost Estimation:

Due to the project's scope and varying factors, providing a definitive cost estimate is difficult. However, here's a rough breakdown:

- Development: $5,000 - $20,000+
- Infrastructure: $500 - $2,000+ per month
- Maintenance: $1,000 - $5,000+ per month

Cost Optimization Strategies:

- **Open-source libraries and frameworks:**
    - Utilize freely available libraries and frameworks for various functionalities, reducing licensing costs.
- **Cloud-based development and hosting:**
    - Leverage cloud platforms for development and deployment to reduce infrastructure costs and maintenance overhead.

- **Agile development methodology:**
    - Focus on rapid prototyping and iterative development to ensure resource efficiency and early feedback.
- **Community-driven development:**
    - Encourage contributions from open-source communities to leverage shared resources and expertise.

Overall, the cost of developing and maintaining a distributed image processing system using cloud computing Application will depend on various factors like project complexity, team size, and chosen technologies. Implementing cost-optimization strategies can significantly reduce expenses and ensure project viability.

Project plan:

Phase 1: Project Planning and Design (2-3 weeks)

Tasks:

1. Define project scope, objectives, and requirements.
2. Research cloud computing technologies and select the appropriate platform (AWS, Google Cloud, Azure, etc.).
3. Design system architecture, including components, interactions, and data flows.
4. Determine technologies for parallel processing (MPI, OpenCL).
5. Create a detailed project plan with tasks, responsibilities, and timelines.
6. Draft user stories based on gathered requirements.
7. Document project plan and design decisions.

Responsibilities:

- Mohamed Amr, Youssef Emad, Salma Nasreldin: System design, technology selection.
- Mohamed Ibrahim: Diagrams, user stories.

Timelines:

- Weeks 1-2: Define scope, objectives, and requirements; research and select technologies; design system architecture.
- Week 3: Finalize project plan, document design decisions, and user stories.

Phase 2: Development of Basic Functionality (2-3 weeks)

Tasks:

1. Implement basic image processing operations (filtering, edge detection, color manipulation).
2. Set up cloud environment and provision virtual machines.
3. Develop worker threads for processing tasks.
4. Implement image upload functionality.
5. Develop user interface for basic operations.
6. Manual testing of implemented functionality.

Responsibilities:

- All team: Implementation of basic functionalities, GUI coding.
- Mohamed Amr, Youssef Emad: Cloud setup, guidance on system integration, manual testing.
- Salma Nasreldin: Progress tracking, issue resolution, testing connection between vms and local machines.
- Mohamed Ibrahim: diagrams updating, GUI designing.

Timelines:

- Weeks 4: Implement basic image processing operations and cloud setup.
- Weeks 5-6: Develop worker threads, image upload functionality, and user interface.

Phase 3: Development of Advanced Functionality (2-3 weeks)

Tasks:

1. Implement advanced image processing operations (e.g., feature extraction, object recognition).
2. Develop distributed processing functionality using MPI or OpenCL.
3. Implement scalability features to add more virtual machines dynamically.

4. Incorporate fault tolerance mechanisms to handle node failures.
5. Conduct integration testing of advanced functionality.

Responsibilities:

- To be discussed

Timelines:

- Weeks 7-8: Implement advanced image processing operations and distributed processing.
- Weeks 8-9: Incorporate scalability and fault tolerance features, conduct integration testing.

Phase 4: Testing, Documentation, and Deployment (2-3 weeks)

Tasks:

1. Conduct thorough testing of the entire system, including unit, integration, and system testing.
2. Document system design, codebase, and user instructions.
3. Prepare deployment scripts and configurations.
4. Deploy the system to the cloud environment.
5. Perform final system testing and validation.

Responsibilities:

- To be discussed

Timelines:

- Weeks 10-11: Testing and documentation.
- Weeks 12: Deployment, final testing, and validation.

Sequence diagram:

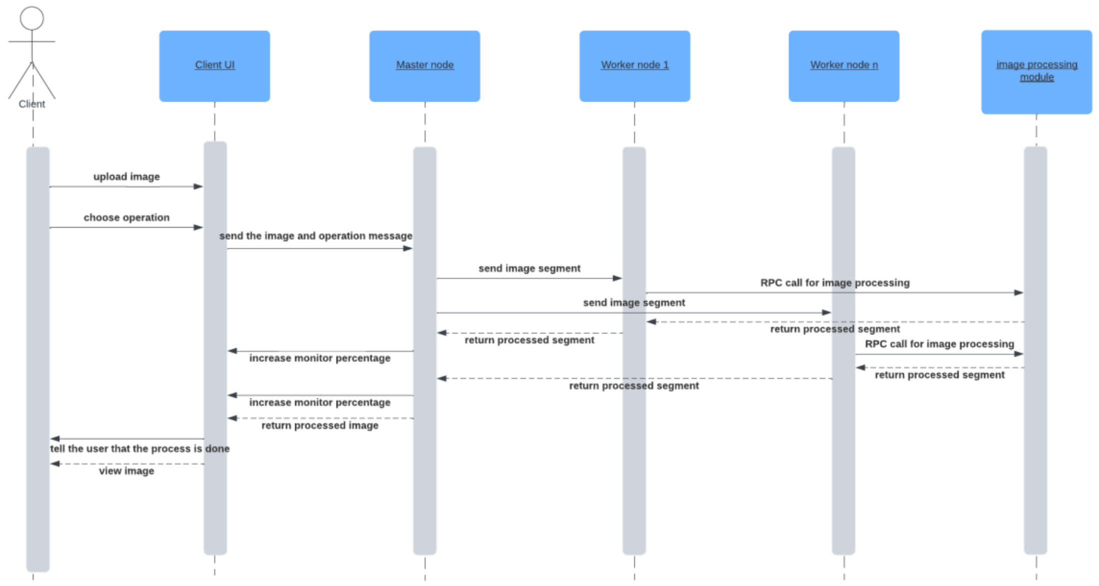

Figure 3 sequence diagram

Components diagram:

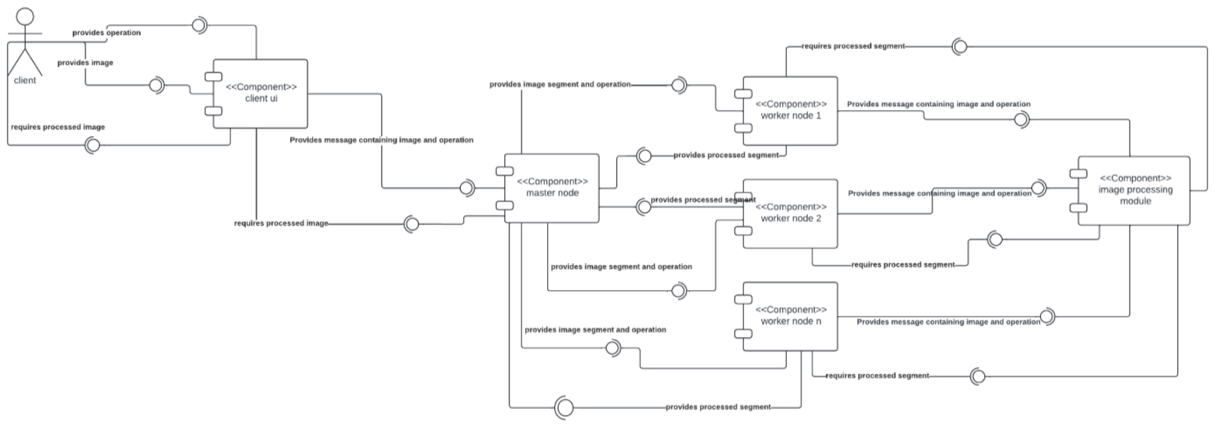
client
provides operation
provides image
requires processed image
<<Component>> client ui
Provides message containing image and operation
requires processed image
<<Component>> master node
provides image segment and operation
provides processed segment
provides processed segment
provides image segment and operation
provides image segment and operation
provides processed segment
requires processed segment
<<Component>> worker node 1
Provides message containing image and operation
<<Component>> worker node 2
Provides message containing image and operation
requires processed segment
<<Component>> worker node n
Provides message containing image and operation
requires processed segment
<<Component>> image processing module

Figure 4 components diagram

Infrastructure diagram:

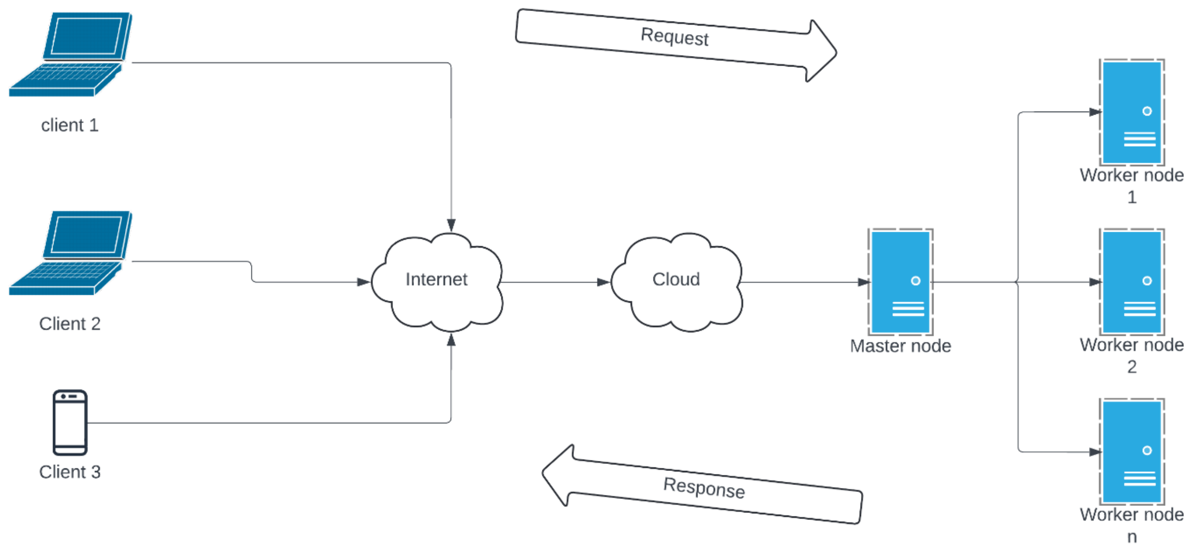

Figure 5 infrastructure diagram

Network diagram:

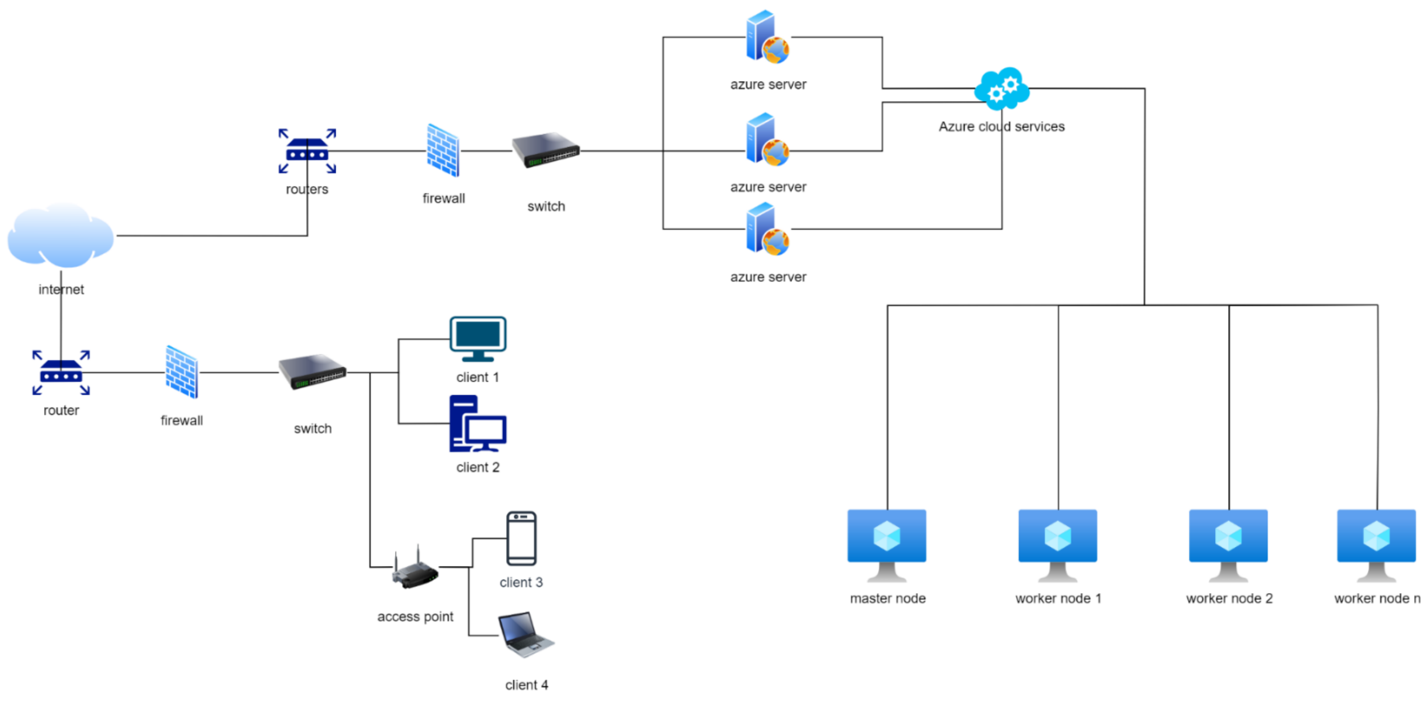

Figure 6 network diagram

End-user guide: Distributed Image Processing System

1. Introduction

Welcome to the Distributed Image Processing System! This user guide will walk you through the steps to upload an image, choose an image processing operation, and process the image using the system's graphical user interface (GUI).

2. Getting Started

- **System Requirements:** Ensure that you have a stable internet connection.
- **Accessing the System:** Open the GUI application.

3. Uploading an Image

- **Step 1:** Click on the "Upload" button to select an image file from your device.
- **Step 2:** Choose the image file you want to process from your local storage and click "Open" to upload it to the system.
- **Step 3:** Once the upload is complete, the selected image will be displayed on the GUI.

4. Selecting Image Processing Operation

- **Step 1:** Choose the type of image processing operation you want to perform from the dropdown menu.
- **Step 2:** Available operations may include:
    - Basic operations such as filtering, color manipulation, etc.
    - Advanced operations like edge detection.
- **Step 3:** After selecting the desired operation, the system will display a preview of the processed image on the GUI.

5. Processing the Image

- **Step 1:** Once you have selected the image processing operation, click on the "Convert" button to initiate the processing.
- **Step 2:** The system will distribute the processing task across multiple virtual machines in the cloud for parallel execution.
- **Step 3:** Once the processing is complete, the processed image will be displayed on the GUI, and you can download it to your device.

6. Conclusion

Congratulations! You have successfully processed an image using the Distributed Image Processing System. Feel free to explore other image processing operations and functionalities offered by the system.

Additional Tips:

- If you encounter any issues or have questions about the system's functionality, refer to the system documentation or contact your system administrator for assistance.
- Ensure that you have the necessary permissions to access and use the system features effectively.

Thank you for using the Distributed Image Processing System. We hope you find it useful for your image processing needs!

Conclusion:

In conclusion, the "Distributed Image Processing System using Cloud Computing" project represents a powerful solution to the growing demand for efficient image processing capabilities across various industries and domains. By harnessing the power of cloud computing and distributed systems, the system enables faster, more scalable, and fault-tolerant image processing workflows, benefiting researchers, healthcare professionals, media creators, security agencies, e-commerce platforms, government organizations, and more.

Moving forward, the project lays a solid foundation for future enhancements and extensions, paving the way for further innovation in the field of distributed image processing. With ongoing development and refinement, the system has the potential to continue making significant contributions to advancing image processing technologies and addressing real-world challenges effectively.

References:

https://learn.microsoft.com/en-us/azure/?product=popular

https://learn.microsoft.com/en-us/azure/azure-portal/

https://docs.python.org/3/library/socket.html

https://realpython.com/python-sockets/

https://docs.python.org/3/howto/sockets.html

https://www.techtarget.com/searchapparchitecture/definition/Remote-Procedure-Call-RPC

https://miro.com/diagramming/what-is-a-uml-diagram/

https://www.geeksforgeeks.org/python-network-programming/

https://konfuzio.com/en/cv2/#:~:text=The%20cv2%20module%20is%20the,commonly%20used%20functions%20in%20cv2.