

**CSE354: Distributed Computing**

*Project Title:*

Distributed Image Processing System using Cloud Computing

# Phase 1: Project Planning and Design

*Group(46) :*

*Eslam Mostafa Mohamed Haider 20P4797*

*Belal Mahmoud Amer Mahmoud 19P2374*

*Abdelrahman Mostafa 20P6060*

*Zeyad Essam Elsayed 20P5728*

*Submitted to:*

*Prof. Ayman M. Bahaa-Eldin*

Contents

[Phase 1: Project Planning and Design 1](#_Toc165222898)

[# Project proposal 3](#_Toc165222899)

[1. Introduction 3](#_Toc165222900)

[2.Project objectives 3](#_Toc165222901)

[3. Project Scope 4](#_Toc165222902)

[4.Project Requirements 4](#_Toc165222903)

[5.Project Deliverables 5](#_Toc165222904)

[#Design document 7](#_Toc165222905)

[6. System Architecture Design 7](#_Toc165222906)

[7. Project Plan and timelines 10](#_Toc165222907)

[8. User Stories 13](#_Toc165222908)

## # Project proposal

In this part of the report, we aim to define the project scope, and objectives of the project.

### 1. Introduction

This project aims to develop a distributed image processing system using cloud computing technologies. The system will be implemented in Python, leveraging cloud-based virtual machines for distributed computing. The application will use OpenCL or MPI for parallel processing of image data.

The system aims to harness the power of cloud-based virtual machines to distribute image processing tasks seamlessly across a network of computational nodes. By utilizing the computational resources of the cloud, the system endeavors to accelerate image processing workflows, enhance scalability, and ensure fault tolerance. This report provides an in-depth analysis of the system's architecture, features, implementation details, and potential applications.

### 2.Project objectives

*The primary objectives of this project are to:*

***1-******Scalability****: Design a system that can handle a large number of image processing requests efficiently by scaling resources dynamically.*

***2-Performance****: Ensure quick processing times for uploaded images by distributing the workload across multiple computing nodes.*

***3-Reliability****: Create a reliable system that can handle failures gracefully and ensure high availability of services.*

***4-Cost-effectiveness:*** *Optimize resource usage to minimize operational costs while meeting performance requirements.*

***5-Security:*** *Implement robust security measures to protect user data and prevent unauthorized access to the system.*

***6-Flexibility:*** *Design the system to accommodate future changes and additions in image processing algorithms or features.*

***7-Resource Utilization****: Optimize resource utilization to minimize idle resources and maximize cost efficiency.*

***8-Interoperability****: Ensure compatibility with different image formats and processing libraries to support diverse use cases.*

### 3. Project Scope

*The scope of the project involves designing and implementing a distributed image processing system leveraging cloud computing technologies. This system will allow users to upload images to a cloud-based platform, where they will be processed concurrently using multiple computing nodes. The processed images will then be available for download or further analysis.*

***1-Integration:*** *Determine how the system will integrate with existing applications or systems, if applicable.*

***2-Geographical Considerations:*** *Define whether the system needs to be deployed globally or in specific regions, considering data sovereignty and latency requirements.*

***3-Regulatory Compliance:*** *Ensure the system complies with relevant data protection regulations (e.g., GDPR, HIPAA) and industry standards.*

***4-User Feedback:*** *Incorporate mechanisms for collecting user feedback to continuously improve the system.*

### 4.Project Requirements

**1-Cloud Infrastructure:** Select a cloud service provider (e.g., AWS, Azure, Google Cloud) and set up the necessary infrastructure for hosting the image processing system.

Image Upload and Storage: Develop functionality for users to upload images to the cloud storage.

**2-Image Processing:** Implement distributed image processing algorithms that can be parallelized across multiple computing nodes.

**3-Monitoring and Logging:** Set up monitoring and logging tools to track system performance, resource usage, and errors.

**4-Fault Tolerance:** Implement mechanisms for fault tolerance and automatic recovery to ensure the system remains operational even in the face of failures.

**5-Security Measures:** Implement encryption for data in transit and at rest, role-based access control, and other security measures to protect the system from unauthorized access and data breaches.

**6-Scalability:** Design the system to scale horizontally to handle increased load as the number of image processing requests grows.

**7-API and User Interface:** Develop an intuitive user interface and API for users to interact with the system, upload images, and retrieve processed results.

**8-Resource Orchestration**: Implement a resource orchestration mechanism (e.g., Kubernetes) for efficient management of computing resources.

**9-Load Balancing:** Design load balancing mechanisms to evenly distribute processing tasks among available nodes.

**10-Data Persistence:** Choose appropriate data storage solutions (e.g., object storage, databases) for storing uploaded images and processed results.

**11-Version Control:** Implement version control for image processing algorithms and configurations to track changes and facilitate rollback if needed.

**12-Cost Monitoring:** Set up tools for monitoring and analyzing cloud usage to identify cost-saving opportunities and optimize resource allocation.

**13-Data Backup and Recovery:** Establish backup and recovery procedures to protect against data loss and ensure data integrity.

**14-Performance Benchmarking:** Define performance metrics and conduct benchmarking tests to evaluate the system's efficiency and identify areas for optimization.

**15-Task Distribution:** Design a mechanism for distributing image processing tasks efficiently among available computing nodes.

### 5.Project Deliverables

**1-System Architecture Diagram:** A high-level overview of the distributed image processing system, showing components, interactions, and data flow.

**2-Technical Specifications:** Detailed specifications for each component of the system, including APIs, data formats, and communication protocols.

**3-Prototype:** A functional prototype demonstrating key features of the system, such as image upload, processing, and retrieval.

**4-Documentation:** Comprehensive documentation covering installation, configuration, usage, and troubleshooting of the system.

**5-Testing Plan:** A plan for testing the system to ensure it meets performance, reliability, and security requirements.

**6-Deployment Plan:** Guidelines for deploying the system in a production environment, including scaling strategies and best practices for maintenance and monitoring.

**7-Training Materials:** Develop training materials and documentation for users and administrators to onboard and operate the system effectively.

**8-Community Engagement:** Foster a community around the system by providing forums, documentation, and support channels for users and developers.

**9-Continuous Improvement Plan**: Outline strategies for continuously improving the system based on feedback, performance metrics, and emerging technologies.

**10-Comprehensive Testing:** Conduct thorough testing, including unit tests, integration tests, and end-to-end tests, to ensure the reliability and functionality of the system under various scenarios.

**11-Scalability Testing:** Perform scalability testing to validate the system's ability to handle increased load and scale resources accordingly.

By defining these elements clearly in the planning and design phase, you'll have a solid foundation for developing your Distributed Image Processing System using Cloud Computing.

## #Design document

In this part of the report, we aim to specify system’s functionalities, architecture, components, and Design the system architecture and select the appropriate technologies.

And Create a detailed project plan with tasks, responsibilities, timelines ,and User Stories

### 6. System Architecture Design

This system utilizes a 3-tier architecture to achieve distributed image processing using cloud computing. Here's a breakdown of the tiers:

**Tier 1: Presentation Tier**

* **Components:** user interface (UI)
* **Technology:** Python GUI (**Tkinter**)
* **Reason:**
* **Functionality:**
  + Provides a user interface for uploading images.
  + Allows users to select desired image processing algorithms.
  + Displays progress of the processing task.
  + Enables downloading of processed images.

**Tier 2: Business Logic Tier**

 **Components:**

 **Processing Manager (Python, Cloud SDK):**

* **Job Coordinator:** Receives user requests and uploaded images.
* **Algorithm Selector:** Interacts with the Algorithm Library to select the chosen processing algorithm.
* **Task Distributor:** Splits images into chunks and distributes tasks to the Worker Supervisor.
* **Result Aggregator:** Collects results from worker nodes and assembles the final processed image.
* **User Interface Coordinator:** Handles user interactions like progress updates and download requests.

 **Worker Supervisor (Python, Cloud SDK, OpenCL):**

* **Virtual Machine Manager:** Provisions and manages virtual machines in the cloud.
* **Task Scheduler:** Schedules image processing tasks on available worker nodes.
* **Worker Health Monitor:** Monitors worker node health and handles failures by reporting back to the Job Coordinator for task reassignment.
* **Technology:** Python with cloud APIs and libraries (Cloud SDK, OpenCL)
* **Reason:** Cloud providers offer APIs and libraries (like Cloud SDK) specifically designed to interact with their services. These tools simplify tasks like provisioning virtual machines, interacting with storage services, and managing resources within the cloud environment.

OpenCL leverages GPUs for efficient image processing tasks, while MPI facilitates communication and task distribution across multiple virtual machines. Choosing between them depends on the type of processing and hardware resources available in the cloud environment.

* **Functionality:**
  + Processing Manager:
    - Receives user requests and uploaded images.
    - Splits the image into smaller chunks for parallel processing.
    - Interacts with the worker manager to distribute tasks to virtual machines.
    - Collects results from worker nodes and assembles the final processed image.
    - Handles user interactions like progress updates and download requests.
  + Worker Manager:
    - Provisions and manages virtual machines in the cloud.
    - Sends image processing tasks (chunks) to worker nodes.
    - Monitors worker node health and handles failures by reassigning tasks.

**Tier 3: Data Tier**

* **Components:** Cloud storage service (Cloud Storage)
* **Technology:** Cloud provider's storage service APIs
* **Reason:** Cloud storage services offered by major providers (e.g., Cloud Storage, S3 buckets) provide scalable and reliable storage for user images, temporary processing data, and final processed images. They are cost-effective and integrate seamlessly with cloud compute services.
* **Functionality:**
  + Stores uploaded user images.
  + Stores temporary processing data on worker nodes (if needed).
  + Stores the final processed image after completion.

**Communication Flow:**

1. User interacts with the UI, uploading an image and selecting processing options.
2. Presentation tier sends the image data and user request to the processing manager.
3. Processing manager splits the image and interacts with the worker manager.
4. Worker manager provisions virtual machines (if needed) and distributes image chunks for processing.
5. Worker nodes perform the assigned image processing tasks using OpenCL or MPI for parallelization.
6. Processed image chunks are sent back to the processing manager.
7. Processing manager assembles the final image and stores it in the cloud storage.
8. Processing manager updates the UI with the processing status and allows downloading the final image.

**Benefits of this Architecture:**

* **Scalability:** Easy to add more virtual machines for increased processing power.
* **Fault Tolerance:** System can recover from failures by reassigning tasks.
* **Distributed Processing:** Efficiently utilizes cloud resources for parallel image processing.
* **Clear Separation of Concerns:** Each tier handles specific functionalities.

**Component Diagram: -**

A diagram of a computer network

Description automatically generated with medium confidence

**Sequence Diagram: -**

A diagram of a graph

Description automatically generated with medium confidence

**Network Diagram: -**

A diagram of a diagram

Description automatically generated

**Infrastructure Diagram: -**

A diagram of a network

Description automatically generated with medium confidence

### 7. Project Plan and timelines

**Phase 1: Two to three weeks of project planning and design**

**Describe the project's goals, requirements, and scope (Week 1)**

**Tasks:**

Have a launch meeting to go over the objectives and scope of the work.

Establish the goals and success standards for the project.

Collect and record specific needs from interested parties.

**Week 1: Design System Architecture and Choose Technologies**

**Tasks:**

**System Architect:**

Create the general architecture of the system, including its parts and functions.

Select MPI or OpenCL for parallel processing.

**Engineers in Software:**

Investigate and choose a cloud computing platform (such as GCP, AWS, or Azure).

Ascertain picture data storage options.

**Establish a thorough project schedule (Week 2)**

**Tasks:**

**Manager of the Project:**

Divide the project into manageable stages, tasks, and subtasks.

Establish deadlines and assign roles.

Keep track of the duties, deadlines, and tasks for every stage.

Specify deadlines and checkpoints.

**Acceptance Criteria and User Stories (Week 2)**

**Tasks:**

**Product Owner:**

Refine user stories based on gathered requirements.

Define acceptance criteria for each user story.

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

**Implement Basic Image Processing Operations (Week 3)**

**Tasks:**

**Software Engineers:**

Develop functionality to upload images.

Implement basic image processing algorithms such as filtering and color manipulation.

**Set Up Cloud Environment and Virtual Machines (Week 3)**

**Tasks:**

**System Administrator:**

Provision cloud-based virtual machines.

Configure networking and security settings**.**

**Develop Worker Thread for Processing Tasks (Week 4)**

**Tasks:**

**Software Engineers:**

Create worker thread for executing image processing tasks.

Integrate worker thread with cloud environment for distributed computing.

----------------------------------------------------------------------------------------------------

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

**Implement Advanced Image Processing Operations (Week 5)**

**Tasks:**

**Software Engineers:**

Implement advanced image processing algorithms such as edge detection.

Test and optimize algorithms for performance.

**Develop Distributed Processing Functionality (Week 5-6)**

**Tasks:**

**Software Engineers:**

Implement distributed processing logic using selected technology (MPI or OpenCL).

Handle task distribution and coordination among virtual machines.

**Implement Scalability and Fault Tolerance Features (Week 6-7)**

**Tasks:**

**System Architect:**

Design mechanisms for scaling the system dynamically.

Implement fault tolerance features such as task reassignment.

**Phase 4: Deployment, Documentation, and Testing (two to three weeks)**

**Perform Extensive Testing (Week 8)**

**Tasks:**

**Quality Control Group:**

Evaluate systems, integrate systems, and test units.

Determine and fix errors and problems.

**Record User Instructions, Code, and System Design (Week 9)**

**Technical Writers' Tasks:**

Record the codebase, algorithms, and architecture of the system.

Write troubleshooting manuals and user instructions.

**Ensure Operational and Deploy System to Cloud (Week 10)**

**System Administrator tasks:**

Install the completed system in a cloud environment.

Make one last check to make sure the system is working.

**Milestones:**

Phase 1: Project Planning and Design Completed

Phase 2: Development of Basic Functionality Completed

Phase 3: Development of Advanced Functionality Completed

Phase 4: Testing, Documentation, and Deployment Completed

### 8. User Stories

**1-As a user**, I want to upload an image to the system for processing, so that I can apply various image processing operations.

**Acceptance Criteria:**

The system should provide a user interface or API endpoint for uploading images.

Supported image formats should include common formats such as JPEG, PNG, and GIF.

Users should receive confirmation once the image has been successfully uploaded.

**2-As a user**, I want to select the type of image processing operation to be performed, so that I can customize the processing according to my requirements.

**Acceptance Criteria:**

The system should present users with a list of available image processing algorithms or operations.

Users should be able to choose one or more operations to be applied to their uploaded image.

The selected operations should be clearly displayed or indicated to the user for confirmation.

**3-As a user**, I want to download the processed image once the operation is complete, so that I can use the processed image for further purposes.

**Acceptance Criteria:**

Once the image processing task is complete, users should be provided with a download link or option.

The processed image should be available in a commonly supported image format.

Users should receive confirmation that the download is successful.

**4-As a user**, I want to monitor the progress of the image processing task, so that I can track the status and estimated completion time.

**Acceptance Criteria:**

The system should provide a progress indicator or status update for ongoing image processing tasks.

Users should be able to view the percentage completion, estimated time remaining, or any other relevant progress information.

If there are any errors or issues during processing, users should be notified appropriately.

A blue and white logo

Description automatically generated

**CSE354: Distributed Computing**

*Project Title:*

Distributed Image Processing System using Cloud Computing

# Phase 2: Development of Basic Functionality

*Group(46)*

*Submitted to:*

*Prof. Ayman M. Bahaa-Eldin*

**The code**

import tkinter as tk

from tkinter import filedialog

import threading

import queue

import cv2  # OpenCV for image processing

class WorkerThread(threading.Thread):

    def \_\_init\_\_(self, task\_queue):

        super().\_\_init\_\_()

        self.task\_queue = task\_queue

    def run(self):

        while True:

            task = self.task\_queue.get()

            if task is None:

                break

            image, operation, filter\_type = task

            result = self.process\_image(image, operation, filter\_type)

            self.display\_result(result)

    def process\_image(self, image, operation, filter\_type):

        # Load the image

        img = cv2.imread(image, cv2.IMREAD\_COLOR)

        # Perform the specified operation

        if operation == 'edge\_detection':

            result = cv2.Canny(img, 100, 200)

        elif operation == 'color\_inversion':

            result = cv2.bitwise\_not(img)

        # Apply the selected filter

        if filter\_type == 'grayscale':

            result = cv2.cvtColor(img, cv2.COLOR\_BGR2GRAY)

        elif filter\_type == 'blur':

            result = cv2.GaussianBlur(img, (5, 5), 0)

        # Add more filters as needed...

        return result

    def display\_result(self, result):

        # Display the processed image

        cv2.imshow('Processed Image', result)

        cv2.waitKey(0)

        cv2.destroyAllWindows()

def select\_image():

    file\_path = filedialog.askopenfilename()

    entry.delete(0, tk.END)

    entry.insert(0, file\_path)

def process\_image():

    image\_path = entry.get()

    operation = var.get()

    filter\_type = filter\_var.get()

    task\_queue.put((image\_path, operation, filter\_type))

    start\_worker\_thread()

def start\_worker\_thread():

    WorkerThread(task\_queue).start()

root = tk.Tk()

root.title("Image Processing")

frame = tk.Frame(root)

frame.pack(padx=10, pady=10)

label1 = tk.Label(frame, text="Select Image:")

label1.grid(row=0, column=0, sticky="w")

entry = tk.Entry(frame, width=50)

entry.grid(row=0, column=1, padx=5, pady=5)

button = tk.Button(frame, text="Browse", command=select\_image)

button.grid(row=0, column=2, padx=5, pady=5)

label2 = tk.Label(frame, text="Select Operation:")

label2.grid(row=1, column=0, sticky="w")

var = tk.StringVar()

var.set("edge\_detection")

option1 = tk.Radiobutton(frame, text="Edge Detection", variable=var, value="edge\_detection")

option1.grid(row=1, column=1, sticky="w")

option2 = tk.Radiobutton(frame, text="Color Inversion", variable=var, value="color\_inversion")

option2.grid(row=2, column=1, sticky="w")

label3 = tk.Label(frame, text="Select Filter:")

label3.grid(row=3, column=0, sticky="w")

filter\_var = tk.StringVar()

filter\_var.set("grayscale")

filter\_option1 = tk.Radiobutton(frame, text="Grayscale", variable=filter\_var, value="grayscale")

filter\_option1.grid(row=3, column=1, sticky="w")

filter\_option2 = tk.Radiobutton(frame, text="Blur", variable=filter\_var, value="blur")

filter\_option2.grid(row=4, column=1, sticky="w")

process\_button = tk.Button(frame, text="Process Image", command=process\_image)

process\_button.grid(row=5, column=1, padx=5, pady=10)

task\_queue = queue.Queue()

root.mainloop()

**how we implemented the image processing**

Let's break down the image processing implementation:

1. Selecting an Image: When the user clicks the "Browse" button, a file dialog is displayed using filedialog.askopenfilename(). This allows the user to select an image file from their file system. The selected file path is then inserted into the entry widget (entry) to display the selected file path.
2. Processing Image: When the user clicks the "Process Image" button, the process\_image() function is called. This function retrieves the selected image file path from the entry widget (entry.get()), along with the selected operation (edge detection or color inversion) and filter type (grayscale or blur).
3. Worker Thread: The start\_worker\_thread() function is then called to start a worker thread (WorkerThread). This worker thread handles the actual image processing. The selected image file path, operation, and filter type are put into a queue (task\_queue) and processed by the worker thread.
4. Worker Thread Execution: The WorkerThread class inherits from threading.Thread and overrides the run() method. In the run() method, the thread continuously retrieves tasks from the queue (task\_queue) and processes them until it receives a None task, indicating that there are no more tasks to process. For each task, the selected image is loaded using OpenCV (cv2.imread()), and the specified operation and filter are applied to the image using OpenCV functions (cv2.Canny(), cv2.bitwise\_not(), cv2.cvtColor(), cv2.GaussianBlur()).
5. Displaying the Processed Image: The processed image is displayed using cv2.imshow(). The GUI remains blocked until the user closes the displayed image window (cv2.waitKey(0)), after which the processed image window is closed (cv2.destroyAllWindows()).

**How the worker thread worked**

Let's break down how the worker thread works in this script:

1. **Initialization**: The **WorkerThread** class inherits from **threading.Thread**. In its **\_\_init\_\_** method, it initializes the task queue (**task\_queue**) passed as an argument.
2. **Run Method**: The **run()** method of the **WorkerThread** class overrides the **run()** method of the **threading.Thread** class. This method defines what the thread will do when it starts running.
3. **Processing Tasks**: Within the **run()** method, there is an infinite loop (**while True:**) that continuously checks for tasks in the task queue (**self.task\_queue**). It retrieves tasks using **self.task\_queue.get()**.
4. **Handling Task Completion**: If the retrieved task is **None**, it means there are no more tasks to process, so the loop breaks (**if task is None: break**). Otherwise, it proceeds to process the task.
5. **Image Processing**: For each task, which consists of an image file path (**image**), an operation (**operation**), and a filter type (**filter\_type**), the **process\_image()** method is called. This method loads the image using OpenCV (**cv2.imread()**), performs the specified operation and filter using OpenCV functions (**cv2.Canny()**, **cv2.bitwise\_not()**, **cv2.cvtColor()**, **cv2.GaussianBlur()**), and returns the processed image.
6. **Displaying the Result**: The processed image (**result**) is then displayed using OpenCV's **cv2.imshow()** function.
7. **Loop Continuation**: After processing a task, the thread continues to check for more tasks in the task queue. This loop continues until a **None** task is encountered, at which point the thread exits.

**screenshot of the output**

**A screenshot of a computer

Description automatically generated**

Figure 1:The Gui

A lake with rocks and mountains in the background

Description automatically generated

Figure 2:The image that we will edit but before editing

A lake with rocks and mountains in the background

Description automatically generated

Figure 3:Processed image after processing

A screen shot of a computer

Description automatically generated

Figure 4:Many threads have been created