PROJECT REPORT

COMPUTER SYSTEMS PRORAMMING (IT227)

TOPIC: Master-Slave Client Server System

Team Members:

(202401026) – BHAKTI SUDHIR

(202401122) – MRUNALI PARMAR

(202401129) – NISARG

(202401133) – PALAK PATEL

PROF: AMIT MANKODI

Dhirubhai Ambani University

Gandhinagar, Gujarat

Problem Statement

This project simulates a simple **master-slave system** where a master computer gives tasks like word count and PDF-to-text conversion to slave computers. The goal is to show how dividing work among multiple machines can make data processing faster and more efficient. It helps in understanding basic distributed computing concepts. The system also shows how tasks can be managed and results collected from different machines.

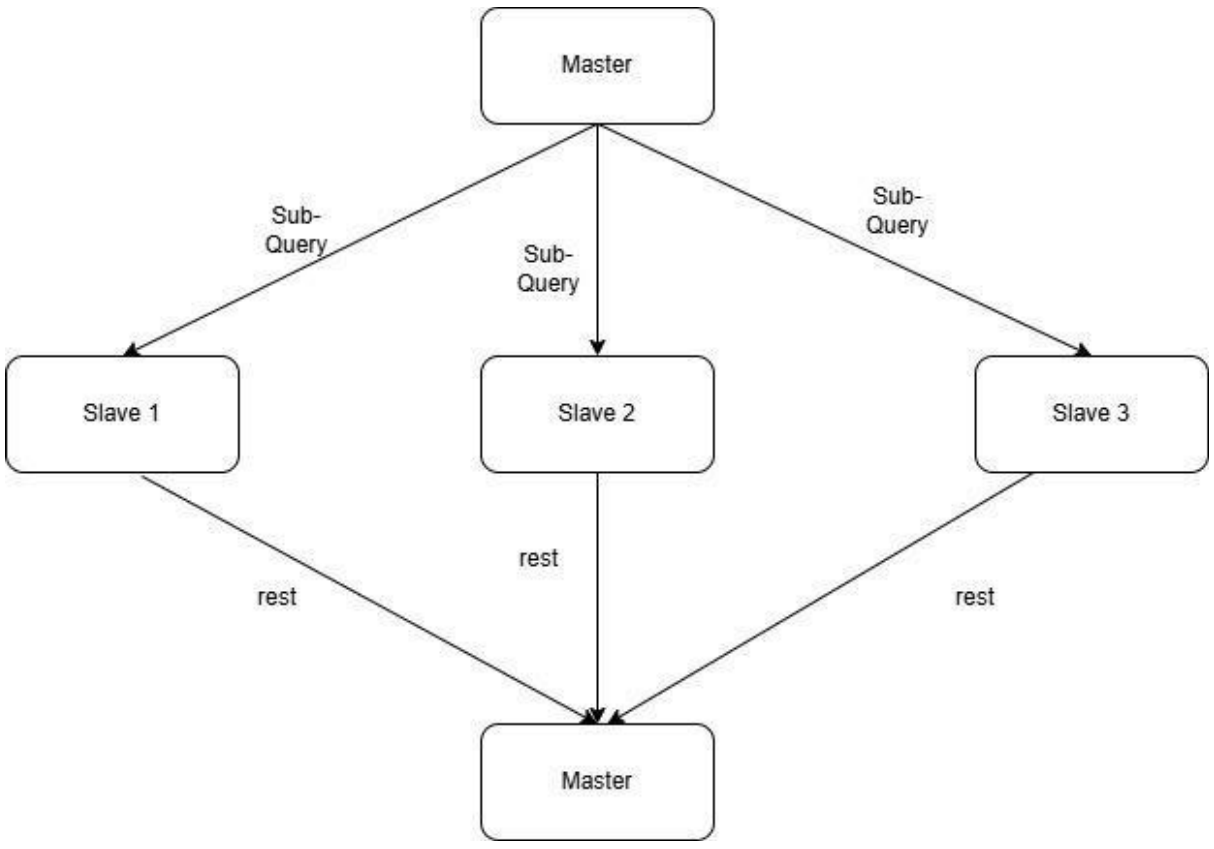
System Overview

The system consists of a client application with a graphical interface that allows users to submit tasks. These tasks are sent to a central master server, which is responsible for managing and distributing the workload. The master server assigns each task to one of several slave servers, which perform the actual computation.

To make data storage and file sharing simple, the system uses the Google Drive API. This lets both the client and servers easily upload the files they need for processing and access the results when they're ready. By connecting everything through Google Drive, the system stays organized, scales easily, and keeps file management smooth and hassle-free.

System Architecture

High-Level Architecture Diagram



System Architecture and Components

1. Client Application (client.py):

- Provides a GUI interface via Tkinter.
- Handles file selection and upload.
- Sends operation commands to the server using socket communication.

- Uses Google Drive API to handle file upload/download.

2. Primary Server (primary_server.py):

- Accepts client requests.
- Coordinates tasks and delegates them to available slave servers.
- Sends back results to the client.

3. Secondary Server (secondary_server.py):

- Takes over if the primary server fails.
- Maintains synchronization with the primary server.

4. Slave Nodes (slave1.py, slave2.py, slave3.py):

- Perform the actual word count or other processing tasks.
- Receive file paths from the server and return results.

5. Google Drive Integration:

- Upload and store processed results using client_secrets.json and token.pickle.

Distributed System Concepts Used

1. Fault Tolerance: Achieved through a secondary server that mirrors the primary.
2. Replication: Slave servers provide redundancy.

3. Transparency: Client is unaware of which slave performs the task.
4. Scalability: Additional slaves can be added with minimal configuration.
5. Concurrency: Multiple clients can be served simultaneously through threading.

Implementation Details

- Language: Python
- GUI Framework: Tkinter
- Networking: socket, threading
- Cloud Integration: Google Drive API (googleapiclient, google-auth-oauthlib)
- File Handling: MIME types, Pickle
- Authentication: OAuth2 with client_secrets.json and token.pickle

Challenges Faced and Solutions

1. Socket disconnection or primary server failure - Implemented a secondary server that monitors the primary and takes over if needed.
2. Google Drive API authentication issues - Used persistent token storage with pickle and automatic refresh using Request().
3. Concurrent request handling - Used Python's threading module on the server side.
4. Handling large files - File uploads are streamed using efficient buffering and Google Drive's MediaFileUpload.
5. Slave discovery - solved using hardcoded IPs and ports (scope for improvement).

6. File synchronization - managed using Google Drive API and token authorization.

Results and Performance Analysis

- **Latency:** File upload and word count were completed within seconds for small files.

- **Throughput:** System supports multiple clients due to multi-threading.

- **Accuracy:** Word count returned correct results across varied test documents.

- **Fault Recovery:** Seamless transition from primary to secondary server observed during simulated failure.

Future Improvements and References

- Add dynamic slave discovery using a Name Server.
- Implement logging and monitoring for better error tracking.
- Improve UI/UX of the client for better user interaction.