Project Report

# Title

Developing an N-Node Distributed System with Vector Clock Implementation

# Introduction

The aim of this project is to develop a distributed system that implements a Vector clock. The system is designed to be n-node, and it is composed of three scripts: generic.py, node.py, and sockets.py. The project is divided into two parts. In Part 1, the distributed system is implemented using a Vector clock. The clock values are printed before and after every message sent or received from/to another system. In Part 2, the three scripts are explained in detail.

Part 1: Implementing Distributed System using Vector Clock

The distributed system is designed to use a Vector clock for synchronization. The Vector clock allows the system to maintain a logical ordering of events. The clock values are updated for every message sent or received by the system. The implementation ensures that before and after clock values are printed for every message exchange.

Part 2: Scripts for Distributed System

The three scripts that make up the distributed system are generic.py, node.py, and sockets.py. The scripts are explained in detail below.

# generic.py:

The primary purpose of this script is to take the number of systems (i.e., NODE\_COUNT) from the user and find the available ports to create socket servers. The script mimics multiple systems of distributed systems in a single machine. The current implementation assumes that there are three nodes in the system. However, users can change the number of nodes by modifying the NODE\_COUNT constant. After finding ports for different systems, the script automatically spawns separate systems in console mode.

# node.py:

This script is the child system script that governs how systems behave on messages received or sent. The script is executed by generic.py, but it can also be manually executed by providing the required command-line arguments. The script is divided into two parts: sender and receiver. To accommodate these two functions, multithreading is used. The sender runs on an infinite for loop in the main thread while the receiver runs in a separate thread. The primary purpose of the sender is to take the message and system ID to send the message to as input and then send the message to the specified system. The receiver's primary purpose is to create a server using sockets that listens to a specified port for any incoming messages.

# sockets.py:

This script file consists of different helper utility functions for node.py. It has functions to create a server for a specific port, handle vector clock changes for the system, and send requests to a specific port/system with a message. Another helper function to print a help message and a dummy internal process for the system are also included in this file.

# Learnings

The project provided insights into using sockets to communicate with other systems and how Vector Clocks in distributed systems work and can be implemented.

# Difficulties Encountered

To see if the port is open, we first utilized the following steps:

1. Get a port number
2. Try using that port to connect (as a sender)
3. It is usable if we can connect to it; else, it is 3.
4. If not, select the following port and repeat steps 1-4.

However, the process took 2-5 seconds for each clock cycle from stages 1-4, making it a time-consuming procedure. It was not the best option.

These are the steps we used to optimize it:

1. First, obtain any port number.
2. Second, attempt to bind the socket to that port (as a listener/server).
3. It is usable if we can connect to it; else, it is 3.
4. If not, select the following port and repeat steps 1-4.

That cut the time it took to find all of the open ports in half, from 2-5 seconds each port.

# Conclusion

The project was tested and implemented as defined in the project statement. The code for the project is in the source\_code directory, distributed in different files as mentioned above. The README file is in the same project directory.