

UNIVERSITY OF TEXAS-ARLINGTON



UNIVERSITY OF
TEXAS
ARLINGTON

PROJECT-2 REPORT

Submitted in partial fulfilment of the requirement for the course-work
CSE-5306-003 DISTRIBUTED SYSTEMS

SUBMITTED BY

JAYACHANDRA JARAJAPU (1001964536)

PRUDHVI GURRAM (1001955767)

DECLARATION

I PRUDHVI GURRAM bearing UTA ID: 1001955767 and JAYACHANDRA JARAJAPU bearing UTA ID: 1001964536, of the Department of Computer Science, University of Texas at Arlington Confirm that we have neither given nor received unauthorized assistance on this work.

Prudhvi Gurram

Jayachandra Jarajapu

10/10/2022

INTRODUCTION:

The Requirement of the project is to implement a vector clock for a distributed system using Python RPC's or sockets. In this project we have implemented using RPC method. There will be two threads in RPC, one for sending the message to other nodes, the other thread for listening messages to its communication port. We have implemented a logical clock to timestamp message received from or sending to other nodes in the distributed system. The number of nodes we used in our network is 3 and all three will never fail or leave the distributed system.

IMPLEMENTATION OF N-NODE DISTRIBUTED SYSTEM:

In this project, we were told to design a N-Node distributed system which should be able to send and receive messages to other devices in the network. We have targeted three nodes in our distributed system. In python we have a package called SimpleJSONRPCServer which will be able to create a server using localhost and port number of that host. Once the host is created, we have created two threads for this RPC server, one for sending messages to other two nodes and the other thread for listening messages to the communication port. The threads are present in the nodes of the distributed system and each node is connected to an Ip address and port.

The client makes an RPC call and sends a message to the JSON server and this server will send the message to other node in the network and the receiving node will receive the message and send back the updated clock to the node with results. To perform this operation, the nodes will use the same Ip address where the nodes are open and send and receiving messages between nodes. If the existing node is not connected or failed to receive the message then the remaining nodes won't timestamp the messages sent to the targeted node.

The program has been implemented using RPC programming in python and we have used threading in python as well.

WORKFLOW:

- Initially we are considering the vector clock of all the devices as zero and then incrementing the clocks by one after any operation is performed.
- we implemented a vector clock for all the three nodes present in the distributed system. To simplify the timestamp of the devices, we have implemented random integer defining a range to generate a logical clock for these devices.

- We have implemented a class where we can import this class across all nodes and get the timestamp of the messages.
- The three nodes are implemented in separate python scripts and each node will have its own port number which is connected to the same Ip address over the network
- If the node is sending message to itself then the vector clock is incremented by one and if node is targeting other nodes, then it will increment its clock and also updating the targeted node clock.
- The device makes an synchronous RPC call every time to send or receive messages to other nodes in the network.
- We have created two threads for each node where one thread will serve to send messages and the other thread to listen messages to it's communication port.
- Each node will take the current device vector clock from the vectorclock class and it is called for each node using this class
- Once the device is started the thread will be created and started using threading class and is targeted to send messages to another node
- While sending messages we will enter the message and the targeted node as a command line argument and based on the input the RPC will make a call to another node in the network
- Once the device sends the message to the other node, it will update its vector clock and executed the RPC call.
- The receiving node will wait for the message and prints the vector clock before receiving the message
- For receiving the message, we have used register function which will receive the message from the incoming vector and thus printing the message
- After receiving the message, it will also increment Its vector clock confirming that the message is received and the vector clock is updated.

ISSUES ENCOUNTERED:

- Implementing a logical clock requires chronological order of timestamping the messages in the distributed network and we tried to consider global time clock where each device will be in a separate time zone and timestamp messages but we had constraints while updating the clock every time so we created a class where we initialize the vectors to zero and get a random integer for that device.

- To create a communication between the nodes we tried to implement sockets but we had encountered issues where the socket is dropping the calls between the nodes so we implemented a SimpleJSONRPCServer which will bind the host and port and this server will fork between listen and accept between the nodes in the network.
- Vector clocks require syncing vectors of all processes at the same time and we had issues syncing all the process and in order to alleviate the situation we have using `list(map(zip))` instead of iterating each element of the list position in different lists, this function does the all operations in a single code.

Team Collaboration:**Prudhvi Gurram:**

- Learnt about vector clocks and how it can be implemented in python and update the nodes for each operation performed
- Worked on implementing vector clock and initializing threading for performing operations between the nodes
- Learnt about register function and list mapping and how it can be used to sync the vector clock between the nodes.
- Prepared the ReadMe file and part of report for the project

Jayachandra Jarajapu:

- Worked on implementing RPC threading for sending the messages and listening to the communication port for all devices.
- Learnt about JSONRPC library which will help us to learn host an RPC server and later can be used by threading
- Learnt how to manage hosted servers in sending and listening to the messages in the distributed network.
- Prepared the readme file and part of report for the project.



KNOWLEDGE WE GAINED:

- After implementing the project, we gained knowledge about how vector clocks work and how it processes the events over a distributed network
- Vector clocks have an issue where we need to send entire vector to each process to for every message sent in order to keep the vector clock sync and we have learnt how to implement them for all nodes in the network
- We have learnt about python list mapping to check for the latest vector clock in each list and update the latest vector clock in the network

References:

Honour, Joe. "Distributed Systems: Physical, Logical, and Vector Clocks." *Medium*, Level Up Coding, 30 Mar. 2020, <https://levelup.gitconnected.com/distributed-systems-physical-logical-and-vector-clocks-7ca989f5f780>.

Mssaxm, Mssaxm. "Simple Pre-Forked PYTHON JSON-RPC Server." *Corps of Engineers*, 20 Jan. 2013, <https://axialcorps.wordpress.com/2013/01/16/simple-pre-forked-python-json-rpc-server/>.

Hanford, Patrick. "Understanding Map, Filter, and Zip in Python." *DEV Community* , DEV Community , 1 Jan. 2020, <https://dev.to/codespent/understanding-map-filter-and-zip-in-python-3ifn>.