Distributed Systems, Project 2

CSE 5306, Summer 2022

Contributors:

Rishitha Patel - 1001863136, rxp3136@mavs.uta.edu

Keerthi Chittineni - 1001967726, kxc7726@mavs.uta.edu

I have neither given nor received unauthorized assistance on this work.

Student Name & Sign	Date
Rishitha Patel (1001863136)	7/16/2022
Keerthi Chittineni (1001967726)	7/16/2022

Requirements:

In this programming project, you will develop an n-node distributed system that implements a vector clock. The distributed system uses a logical clock to timestamp messages sent/received among the nodes. You can use any programming language. To simplify the design and testing, the distributed system will be emulated using multiple processes on a single machine. Each process represents a machine and has a unique port number for communication.

Implement the vector clock for your distributed system. You can create two threads for each process, one for sending messages to other nodes and one for listening to its communication port. Communication among nodes can be done using RPC or using sockets. Once a process sends a message, it should print its vector clock before and after sending a message. Similarly, once a process receives a message, it should print its vector clock before and after receiving the message. You can assume that the number of processes (machines) is fixed (equal to or larger than 3) and processes will not fail, join, or leave the distributed system.

Concepts covered:

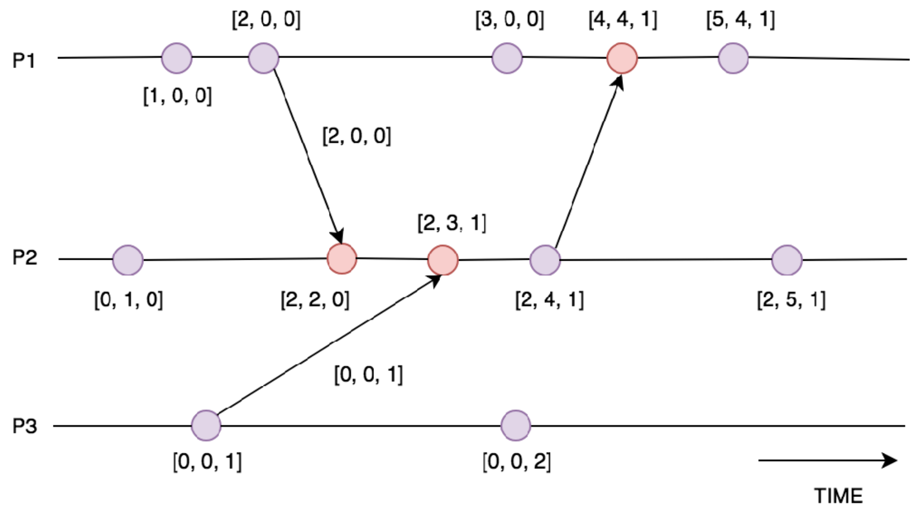
1. In this we have implemented a vector clock for the distributed system
2. In this we have used three ports for the communication.
3. We use a vector of integer values to represent the timestamp.

How it works:

1. Open the command prompt for vector_clock and run **python app.py** command.
2. After the connection we can see all the time stamps that when the messages are sent and received

Vector Clock Implementation:

Vector clock is used to assign timestamps for events in a distributed system. It also gives a partial ordering of the events. A vector of integer values is used to represent the timestamp. If we have N processes in the group, then each process will have a vector with N elements.



1. Here in the vector clock, before executing any event Pi increases its own counter by executing VCi[i]=VCi[i]+1. In the above figure, when P1 is at [1, 0, 0], it executes some event and its vector clock becomes [2, 0, 0].

2. In the second step, when Pi sends a message to Pj, the timestamp of the message is set to VCi after step 1 executes i.e.; the message timestamp is nothing but the vector clock of Pi. So after that, P1 updates the message timestamp and sends the message to P2.

3. After message is received, Pj updates each element of its own vector clock to the maximum of the current value and received value that is at first the vector clock P2 was[0, 1, 0], after receiving the message with timestamp [2, 0, 0], the vector clock at P2 becomes[max(0,2),max(1,0),max(0,0)]=[2, 1, 0], while delivering the message, P2 increases its own counter, so the final vector clock becomes [2, 2, 0].

The following algorithm is used to maintain the timestamp vector:

- Assume that initially all clocks are zero.
- Before each local atomic event, the local clock value for a process is incremented at least once.
- It includes a copy of its own timestamp vector, each time a process sends a message.
- It increments its own logical clock in the vector by one and updates each element in its vector by taking the maximum of the value in its own vector clock , each time a process receives a message.

Output of Vector clock implementation:

```
C:\Windows\System32\cmd.exe - python  app.py
Microsoft Windows [Version 10.0.22000.795]
(c) Microsoft Corporation. All rights reserved.

C:\Users\Rishitha Patel\OneDrive\Desktop\vector_clock>python app.py
*****starting nodes******
server is serving on port 3900
server is serving on port 3901
server is serving on port 3902
127.0.0.1 - - [15/Jul/2022 23:32:06] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2022 23:32:08] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2022 23:32:10] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2022 23:32:12] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2022 23:32:14] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2022 23:32:16] "POST /RPC2 HTTP/1.1" 200 -
127.0.0.1 - - [15/Jul/2022 23:32:16] "POST /RPC2 HTTP/1.1" 200 -
****** triggering a new message event *******
process node on port 3901 is sending message to node on port 3900
vector clock for 3901: [5, 5, 5]
process node on port 3900 received a message
vector clock for 3900: [5, 5, 5]
vector clock for 3900: [6, 6, 5]
127.0.0.1 - - [15/Jul/2022 23:32:20] "POST /RPC2 HTTP/1.1" 200 -
process node on port 3901 sent a message
vector clock for 3901: [5, 6, 5]
127.0.0.1 - - [15/Jul/2022 23:32:20] "POST /RPC2 HTTP/1.1" 200 -
```

Learned from the implementation of the vector Clock:

The vector clock algorithm defines the order between two events whenever inter-process communication creates a causal link between the two events.

In vector clock algorithm, by tracking the logical clock of each process in the system, we make it possible compare and form a globally consistent snapshot of system state.

The vector clock algorithm is useful for applications like garbage collection, or rolling back errors by reversing the order of execution.

Issues encountered while implementing:

At first, we faced problems during RPC implementation due to which we couldn't establish communication between nodes.

And we could only print the vector clock for sending a message, but after working on proxy and port index we could implement everything successfully.

References:

https://medium.com/geekculture/all-things-clock-time-and-order-in-distributed-systems-logical-clocks-in-real-life-2-ad99aa64753

https://www.geeksforgeeks.org/vector-clocks-in-distributed-systems