Report 3: Loggy - A Logical Time Logger

Ziheng Zhang

September 28, 2021

1 Introduction

In this project, I used Lamport Clocks and Vector Clocks to implement a logical time logger which can track sender and receiver information in correct order.

2 Main problems and solutions

2.1 Lamport Clocks

Lamport Clocks focus on using timestamps to keep the order of messages. In my implementation of Lamport Clocks, I used a massage queue to make sure that messages are in the correct order. Specifically, in the message queue, elements are sorted by the third keyword which is Timestamp, and if the timestamp is the smallest among other messages in the queue, it means the message carrying this timestamp can be printed. Otherwise, the message will be pushed into the queue and wait for the next printed time.

This implementation ensures that if an event e happened before the other event e', then the Logical Clocks $L(e)$ is smaller than $L(e')$. It can be written mathematically as $e \rightarrow e' \Rightarrow L(e) < L(e')$.

2.2 Vector Clocks

Vector Clocks let each process keep its own vector clock v_i to keep track of the message's order. Most parts of implementation is the same as Lamport Clocks, and the main differences are concentrated in the file `vector_logger`. I used a message queue to contain all received messages without sorted. When the logger needs to print messages, it iterates all elements in the message queue, and check whether the message ready for output is safe or not. If it is safe, the message will be printed. Otherwise, it will be held on.

This implementation also ensures that if $e \rightarrow e' \Rightarrow V(e) < V(e')$.

3 Evaluation

The results of this assignment is shown below.

```
1> test:run(100, 10).
log: 1 john {sending,{hello,57}}
log: 1 paul {sending,{hello,68}}
log: 2 ringo {received,{hello,57}}
log: 2 paul {sending,{hello,20}}
log: 3 ringo {sending,{hello,77}}
log: 3 paul {sending,{hello,16}}
log: 4 john {received,{hello,77}}
log: 4 ringo {received,{hello,68}}
log: 5 ringo {sending,{hello,20}}
log: 5 john {received,{hello,20}}
log: 6 george {received,{hello,20}}
log: 6 john {sending,{hello,84}}
log: 6 ringo {sending,{hello,97}}
log: 7 george {received,{hello,84}}
log: 7 john {sending,{hello,7}}
log: 8 george {received,{hello,16}}
log: 8 paul {received,{hello,7}}
log: 8 john {sending,{hello,23}}
log: 9 george {received,{hello,97}}
log: 9 paul {sending,{hello,60}}
log: 10 george {sending,{hello,100}}
log: 10 paul {sending,{hello,79}}
```

Figure 1: The result of Lamport Clocks.

```
2> vector_test:run(100, 10).
log: john{sending,{hello,57}}[1,0,0,0]
log: ringo{received,{hello,57}}[1,0,1,0]
log: ringo{sending,{hello,77}}[1,0,2,0]
log: john{received,{hello,77}}[2,0,2,0]
log: paul{sending,{hello,68}}[0,1,0,0]
log: ringo{received,{hello,68}}[0,1,1,0]
log: ringo{sending,{hello,20}}[0,1,2,0]
log: george{received,{hello,20}}[0,1,2,1]
log: paul{sending,{hello,20}}[0,2,0,0]
log: john{received,{hello,20}}[1,2,0,0]
log: john{sending,{hello,84}}[2,2,0,0]
log: george{received,{hello,84}}[2,2,0,1]
log: paul{sending,{hello,16}}[0,3,0,0]
log: george{received,{hello,16}}[0,3,0,1]
log: ringo{sending,{hello,97}}[0,1,3,0]
log: george{received,{hello,97}}[0,1,3,1]
log: john{sending,{hello,7}}[3,2,0,0]
log: paul{received,{hello,7}}[3,3,0,0]
log: george{sending,{hello,100}}[0,1,3,2]
log: ringo{received,{hello,100}}[0,1,4,2]
log: ringo{sending,{hello,20}}[0,1,5,2]
```

Figure 2: The result of Vector Clocks.

4 Conclusions

In this assignment, I leaned the basic knowledge of Lamport Clocks and Vector Clocks and also master the skills of implementing these two algorithms in `Erlang`.

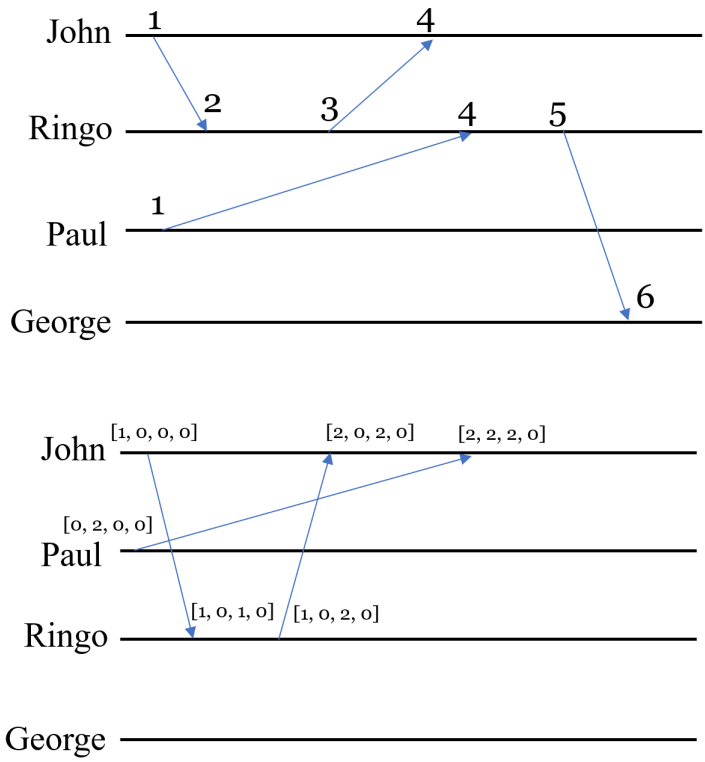

Figure 3: The demonstration of two different locks.