Report : Loggy - a logical time logger

Zhanbo Cui

September 29, 2021

1 Introduction

In this homework, I constructed a logger with Lamport clock and vector clock. We know that in an asynchronous system, because the round-trip times are hardly to control, the clocks in different node often inaccurate. If we want to order the events in a node, a good approach is using logical time. Furthermore, logical time can help us achieve the happened before order in the send-receive scenario.

2 Main problems and solutions

2.1 Lamport clock

The key point of Lamport clock is to set one counter with an initial value 0 for each process. The time is updated in two cases:

1. update the time before sending, then attach it with the message:

```
...
NewTime = time:inc(Name, MyTime),
Message = {hello, rand:uniform(100)},
Selected ! {msg, NewTime, Message},
...
```

Pass a time value into the *inc* function, and return a value increased 1.

2. update the time when receiving a message, the new time is increased 1 based on the maximum value between the timestamp attached with message and the node's time:

```
...
NewTime = time:inc(Name, time:merge(MyTime,Time)),
...
```

In the *merge* function, return the maximum value between *MyTime* and *Time*.

2.2 How loggy print log safely with Lamport?

Now every node equipment a incremented clock, and every message sent to loggy attached with timestamp. It is easy to sort all messages after they have arrived, but we want to print them during execution! The tutorial has told us we need a *hold-back queue* and a *clock* to track all node.

Firstly, we initialize a clock within a fixed number of tuples. Each tuple records the timestamp from the sender! For example, if the clock in loggy is [{A,0},{B,1}], after it receives a new message from node A with timestamp {A,1}, the clock will be updated to [{A,1},{B,1}].

```
    ...
%%when a new log message arrives it should update the clock
Newlogclock = time:update(From,Time,Logclock),
%%add the message to the hold-back queue
Added_queue = lists:append(Holdback_queue,[{From,Time, Msg}]),
%%go through the queue to find messages that are now safe to print
New_queue = checkQueue(Added_queue, Newlogclock),
    ...
```

Now, we have a lot of message with timestamp in the *hold-back queue* and a loggy clock. we know in a send-receive scenario it is not only the order per process should be guaranteed, but also the "happend-before" order. **Loggy only print the message when the timestamp of this message in the queue smaller or equal the (minimum value of time recorded + 1) in the clock**. It easy to understand, because the timestamp in the "receive message" is always larger than the corresponding "send message", if a "send message" doesn't arrive but the "receive message" has arrived, the timestamp of "receive message" is greater than the value of time in the tuple of "sender process" by 2, it will be considered unsafe and stay in the queue until a "send message" arrive to update the clock.

2.3 Vector clock

In vector clock, every node and loggy use a vector to record the logical time. The principle of time updating is a little different with the Lamport clock:

1. Update the **own time tuple** before sending, then attach it with the message

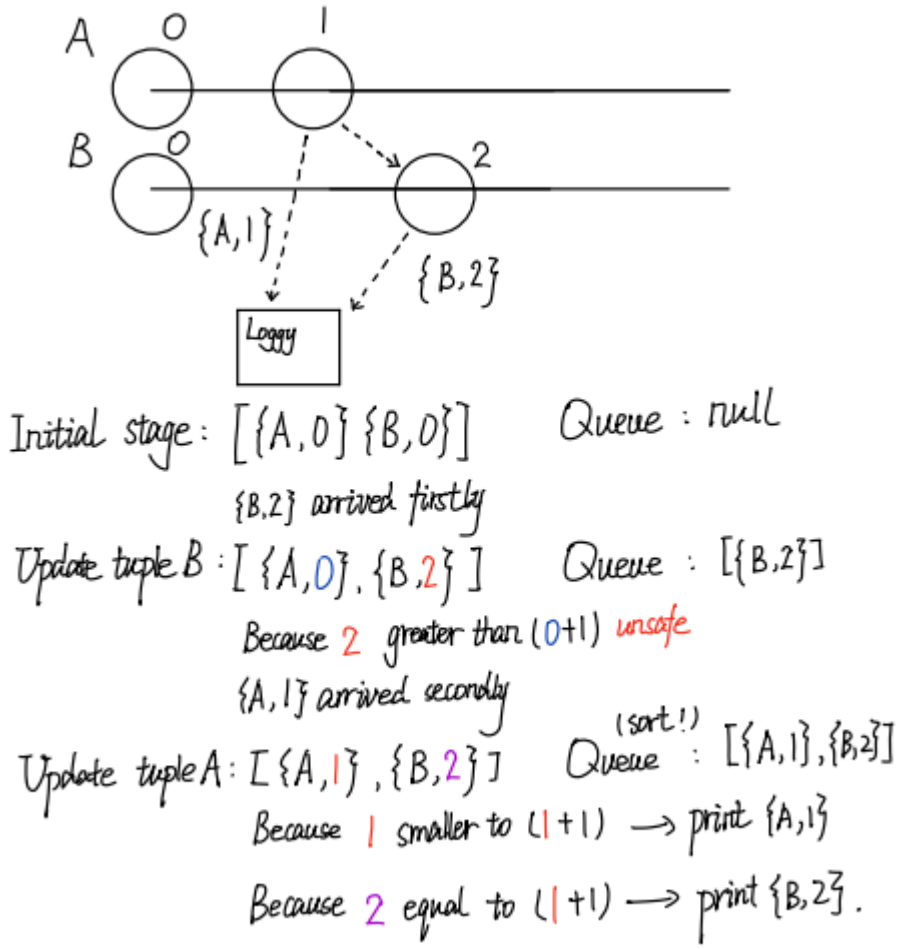

Figure 1: Demo for Lamport Clock

2. Update the **every time tuple with maximum between the timestamp of received message and own time, then increase 1 on own time tuple**

3. The loggy only update the tuple of clock correspond to the sender.

2.4 How loggy print log safely with Vector

I won't repeat what I have talk about queues. I will state how I safely print the messages in the queue. It is easier than in Lamport clock, we just need to compare every tuple of time in the received message with the tuple of the clock. If every timestamp is smaller or equal to the time in the clock, It is

safe.

3 Evaluation

Print the log in two kinds of clocks separately and output their maximum queue length.

3.1 Log with Lamport clock

Figure 2: Log of Lamport Clock

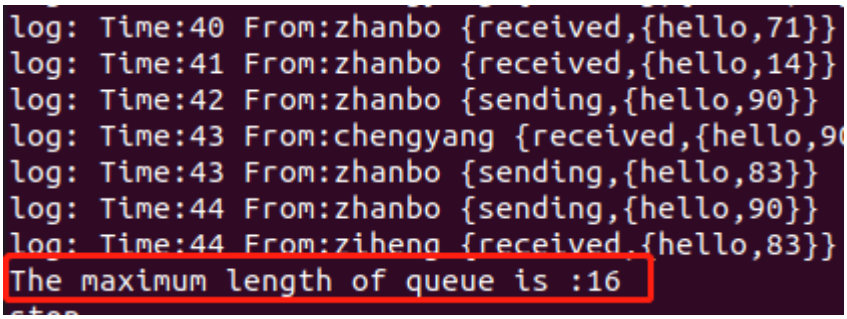

Figure 3: The maximum length of queue of Lamport Clock

3.2 Log with Vector clock

```
5> vecttest:run(50,50).
0 loop: Current Clock:[{zhanbo,0}]
1 loop: Current Clock:[{ziheng,0},{zhanbo,0}]
Output Timestamp:[{ziheng,0}] From:ziheng
{sending,{hello,13}}

Output Timestamp:[{zhanbo,0},{ziheng,0}] From:zhanbo
{received,{hello,13}}

2 loop: Current Clock:[{chengyang,0},{ziheng,0},{zhanbo,0}]
3 loop: Current Clock:[{tianyu,0},{chengyang,0},{ziheng,0},{zhanbo,0}]
4 loop: Current Clock:[{tianyu,0},{chengyang,0},{ziheng,0},{zhanbo,1}]
Output Timestamp:[{zhanbo,1},{ziheng,0}] From:zhanbo
{sending,{hello,15}}

Output Timestamp:[{chengyang,0},{zhanbo,1},{ziheng,0}] From:chengyang
{received,{hello,15}}

5 loop: Current Clock:[{tianyu,0},{chengyang,0},{ziheng,0},{zhanbo,2}]
6 loop: Current Clock:[{tianyu,0},{chengyang,1},{ziheng,0},{zhanbo,2}]
Output Timestamp:[{chengyang,1},{zhanbo,1},{ziheng,0}] From:chengyang
{sending,{hello,9}}

Output Timestamp:[{tianyu,0},{chengyang,1},{zhanbo,1},{ziheng,0}] From:tianyu
{received,{hello,9}}
```

Figure 4: Log of Vector Clock

```
80 loop: Current Clock:[{tianyu,18},{chengyang,18},{ziheng,22},{zhanbo,19}]
81 loop: Current Clock:[{tianyu,18},{chengyang,19},{ziheng,22},{zhanbo,19}]
82 loop: Current Clock:[{tianyu,18},{chengyang,20},{ziheng,22},{zhanbo,19}]
The maximum length of queue is :4
```

Figure 5: The maximum length of queue of Vector Clock

4 Conclusion

We can see that both clocks can be used to sort the logs and output them safely, the only difference is the maximum length of the queue of Vector clock is smaller than the length of queue of Lamport clock. Meanwhile, the log of the Vector clock appears more ordered, which is understandable since Vector clock provides more information on timestamp for sorting.