Loggy - a logical time logger

Chrysoula Dikonimaki

September 30, 2021

1 Introduction

This report describes Loggy, a logical time logger implemented in Erlang.

2 The simple Loggy

First of all, I ran test:run setting jitter=0. We can see that the order in the ideal case that jitter is 0 is that first will be printed the 'sending' message and then will be printed the 'received' message (see Figure 1). However, jitter changes this order, as we can see in Figure 2. Decreasing the jitter will reduce the number of messages received in wrong order.

3 Lamport Time

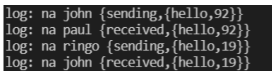
3.1 First Implementation

In Figure 3 we can see a case that the log entries were printed in a wrong order. For example, hello, 57 has been sent from john to ringo and the received message has been printed before the sent message. It is always true that for each worker the messages has been printed in the right order but we want logger to print all messages in the right order.

3.2 Final Implementation

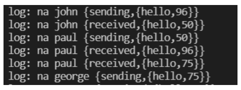
3.2.1 Time Module Description

Functions:



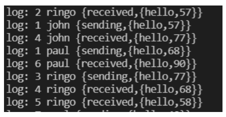
```
log: na john {sending,{hello,92}}
log: na paul {received,{hello,92}}
log: na ringo {sending,{hello,19}}
log: na john {received,{hello,19}}
```

Figure 1: Test with jitter=0



```
log: na john {sending,{hello,96}}
log: na john {received,{hello,50}}
log: na paul {sending,{hello,50}}
log: na paul {received,{hello,96}}
log: na paul {received,{hello,75}}
log: na george {sending,{hello,75}}
```

Figure 2: Test with jitter=1000



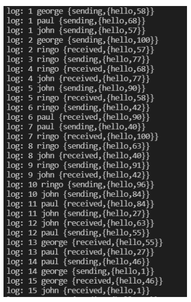
```
log: 2 ringo {received,{hello,57}}
log: 1 john {sending,{hello,57}}
log: 4 john {received,{hello,77}}
log: 1 paul {sending,{hello,68}}
log: 6 paul {received,{hello,90}}
log: 3 ringo {sending,{hello,77}}
log: 4 ringo {received,{hello,68}}
log: 5 ringo {received,{hello,58}}
```

Figure 3: Lamport Time, Test with jitter=1000

- clock: we will represent our clock as a list of tuples, one tuple per worker node. We will set the time of all tuples (the second entry of each tuple) to 0.
- update: replaces the time for a specific worker to the updated time iff the new time is greater than the old one.
- safe: we find the min time and check if the message time is less than the min time. If it's true it means that it is safe to log the message.

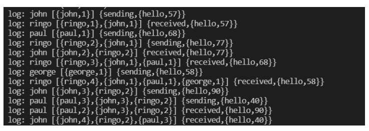
3.2.2 Test

At our final implementation all logs are printed in the right order. We can see it in Figure 4. However, we can't be sure that the events have happened in the order shown because some events are concurrent. Concurrent events are events from different processes that it is impossible to know if they have the happened-before relationship.



```
log: 1 george {sending,{hello,58}}
log: 1 paul {sending,{hello,68}}
log: 1 john {sending,{hello,57}}
log: 2 george {sending,{hello,100}}
log: 2 ringo {received,{hello,57}}
log: 3 ringo {sending,{hello,77}}
log: 4 ringo {received,{hello,68}}
log: 4 john {received,{hello,77}}
log: 5 john {sending,{hello,90}}
log: 5 ringo {received,{hello,58}}
log: 6 ringo {sending,{hello,42}}
log: 6 paul {received,{hello,90}}
log: 7 paul {sending,{hello,40}}
log: 7 ringo {received,{hello,100}}
log: 8 ringo {sending,{hello,63}}
log: 8 john {received,{hello,40}}
log: 9 ringo {sending,{hello,91}}
log: 9 john {received,{hello,42}}
log: 10 ringo {sending,{hello,96}}
log: 10 john {sending,{hello,84}}
log: 11 paul {received,{hello,84}}
log: 11 john {sending,{hello,27}}
log: 12 john {received,{hello,63}}
log: 12 paul {sending,{hello,55}}
log: 13 george {received,{hello,55}}
log: 13 paul {received,{hello,27}}
log: 14 paul {sending,{hello,46}}
log: 14 george {sending,{hello,1}}
log: 15 george {received,{hello,46}}
log: 15 john {received,{hello,1}}
```

Figure 4: Lamport Time Final Implementation, Test with jitter=1000



```
log: john [{john,1}] {sending,{hello,57}}
log: ringo [{ringo,1},{john,1}] {received,{hello,57}}
log: paul [{paul,1}] {sending,{hello,68}}
log: ringo [{ringo,2},{john,1}] {sending,{hello,77}}
log: john [{john,2},{ringo,2}] {received,{hello,77}}
log: ringo [{ringo,3},{john,1},{paul,1}] {received,{hello,68}}
log: george [{george,1}] {sending,{hello,58}}
log: ringo [{ringo,4},{john,1},{paul,1},{george,1}] {received,{hello,58}}
log: john [{john,3},{ringo,2}] {sending,{hello,90}}
log: paul [{paul,3},{john,3},{ringo,2}] {sending,{hello,40}}
log: paul [{paul,2},{john,3},{ringo,2}] {received,{hello,90}}
log: john [{john,4},{ringo,2},{paul,3}] {received,{hello,40}}
```

Figure 5: Vector Clock Implementation, Test with jitter=1000

3.3 Vector Clocks

Vector clocks are useful because we can understand if two events are concurrent. The result of testing the vector clock implementation is shown in Figure 5.

3.4 Maximum number of entries

I create a new function in the test file called run2 which runs the run function N times and get the average max entries the queue had while running. The results of some tests are shown in Table below.

Sleep	Jitter	Average max queue size
100	1000	15.2
100	100	26.2
100	10	24.2
1000	10	13.7

Results of running the experiment multiple times.

4 Conclusions

When it comes to distributed systems, it's difficult to determine the order that the events happen especially, when they don't send messages one another.