Report 3: loggy

Xiaoying Sun

September 23, 2020

1 Introduction

In this exercise I learnt how to use logical time and implement a logging procedure that receives log events from a set of workers. The events were tagged with the Lamport time stamp of the worker and had been ordered before written to stdout.

2 Main problems and solutions

2.1 Functions

The following are the new functions used in this exercise.

$$random : seed(Seed, Seed, Seed)$$

Seeds random number generation with $Seed$ in the process dictionary and returns the old state.

$$random : uniform(Sleep)$$

For a specified integer $Sleep >= 1$, returns a random integer uniformly distributed between 1 and $Sleep$, updating the state in the process dictionary.

$$lists : keysort(2, [From, Time, Msg|Queue])$$

Sort the tuple list $From, Time, Msg$ by the 2_{nd} value of the tuple, and finally return the sorted new tuple list.

$$lists : foldl(fun(,I)- > I + 1end, 0, Queue)$$

Each element in the $Queue$ is executed from left to right and returns the sum of calculation results of all elements.

$$\{OutputQueue, BufferQueue\} = lists : splitwith(Pred, AppendedQueue)$$

Based on an assertion $Pred$ divides one list into two lists, $\{OutputQueue, BufferQueue\}$. The elements in $OutputQueue$ are composed of those that return true in the Pred function call, while $BufferQueue$ is the remaining elements of the $AppendedQueue$, which the assertion determines as false.

2.2 Can't load module 'logger'

The following error occurred at my first compiling the code from tutorial.

```
Can't load module 'logger' that resides in sticky dir

{error,sticky_directory}
```

The reason is that the name of the custom module conflicts with the built-in module. Solution is to change 'logger' to 'loggy'.

3 Evaluation

3.1 Without logical clock

In the beginning, we didn't introduce any kinds of logical clock. The result is shown in figure 1.

```
2> test:run(1000,100).
log:cc 2 {received,{hello,57}}
log:aa 1 {sending,{hello,57}}
log:aa 4 {received,{hello,77}}
log:cc 3 {sending,{hello,77}}
log:cc 4 {received,{hello,68}}
log:bb 1 {sending,{hello,68}}
log:dd 6 {received,{hello,20}}
log:aa 5 {received,{hello,20}}
log:cc 5 {sending,{hello,20}}
log:bb 2 {sending,{hello,20}}
```

Figure 1: Without logical clock

Obviously, the printed results are out of order. So we tried to import Lamport's clock to the worker process.

3.2 Lamport's clock

In this part of the exercise, a logical time stamping mechanism was added to the worker processes.

```
2> test:start().
[<0.86.0>,<0.87.0>,<0.88.0>,<0.89.0>,<0.90.0>]
--------------------
log: aa 1 {sending,{hello,23}}
--------------------
log: bb 1 {sending,{hello,91}}
--------------------
log: bb 2 {sending,{hello,24}}
--------------------
log: aa 2 {sending,{hello,96}}
--------------------
log: dd 2 {received,{hello,23}}
--------------------
log: bb 3 {sending,{hello,8}}
--------------------
log: cc 3 {received,{hello,96}}
--------------------
log: cc 4 {received,{hello,91}}
--------------------
log: cc 5 {sending,{hello,90}}
--------------------
log: cc 6 {sending,{hello,9}}
--------------------
log: aa 6 {received,{hello,90}}
--------------------
log: aa 7 {sending,{hello,5}}
--------------------
log: dd 7 {received,{hello,9}}
--------------------
log: aa 8 {received,{hello,8}}
--------------------
```

Figure 2: Lamport's clock

What was the final log telling us?

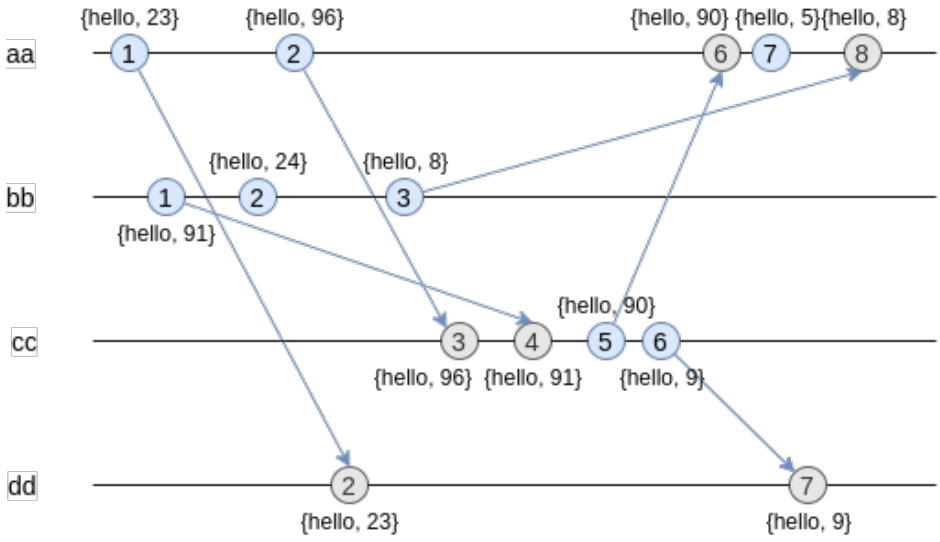

Figure 3: View from final log

It is hard to predict the maximum size of the message buffer queue since it mainly depends on the time drift between the involved worker processes.

3.3 Vector clock

How would things be different with vector clocks? With vector clocks, we assume that

1) We know the number of processes in the group.

2) Each process knows its position in the vector.

Instead of a single number, our timestamp is now a vector of numbers, with each element corresponding to a process. In the example below, the vector elements correspond to the processes $[A, B, C]$.

```
C1 happened before B2 = true (should be true)
C1 happened before B2C1 = true (should be true)
B2A1 happened before A2B3 = false (should be false)
ok
```

Figure 4: Vector clock

What was the final log telling us?

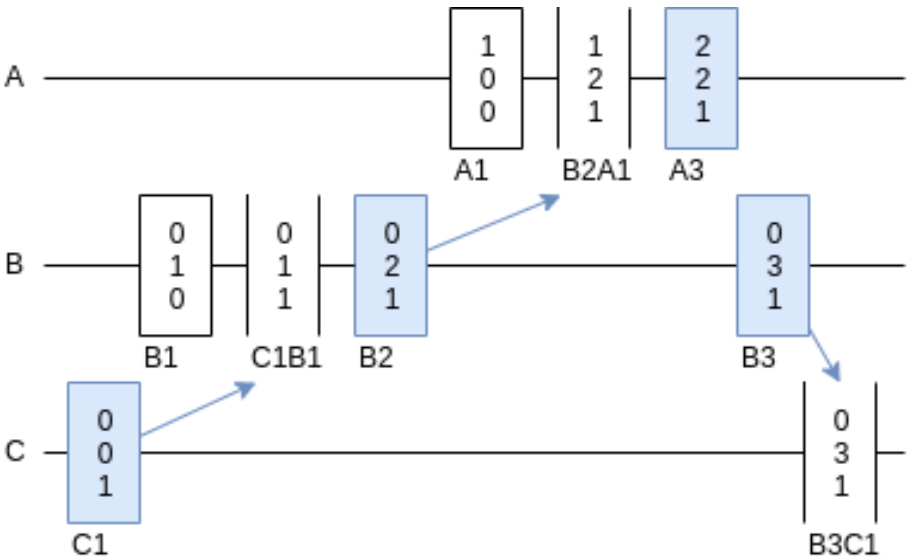

Figure 5: View2 from final log

4 Conclusions

From the above, we successfully implemented Lamport's clock and vector clock. Honestly, I didn't finish the part of vector clock all by my own. Special thanks to Justin Sheehy and Andy Gross. The further work direction should be,

1) How could we implement vector clock without knowing the total numbers of process in advance?

2) how to simulate the vector algorithm in one step?

5 Links

- http://www.vs.inf.ethz.ch/publ/papers/logtime.pdf
- https://www.cs.rutgers.edu/ pxk/417/notes/logical-clocks.html
- http://techtricks.co.in/programming/design-implementation-of-totally-ordered-multicast-using-lamport-logical-clockssynchronization
- https://erlang.org/doc/man/lists.html