Report 3: Loggy - a logical time logger

Yining Hou

September 27, 2023

1 Introduction

In this report, I present a logical time logger by using Lamport and Vector Clock algorithm.

In this seminar, the main topic is about how to receive log events from a set of workers in distributed systems. The events are tagged with time stamp of the worker, and the events must be ordered before being written to stdout.

2 The Worker

A set of workers send logs to the logger every time they send or receive a message.

When a worker receives a message, it should update its timer to the greater of its internal clock and the timestamp of the incoming message.

```
NewClock = time:inc(Name, time:merge(Time, Clock)),
```

When a worker sends a message, it should first increase its timer and send the message to the logger after a small delay.

```
Time = time:inc(Name, Clock),
```

3 The Logger

Accept events and print them on the screen in the right order.

The logger have a clock that keeps track of the timestamps of the last messages seen from each worker. It should also have a holdback queue where it keeps log messages that are still unsafe to print. When a new log message arrives, it should update the clock, add the message to the holdback queue and then go through the queue to find messages that are now safe to print.

```
NewClock = time:update(From, Time, Clock),
NewQueue = lists:dropwhile(fun({F, T, M}) ->
```

```
    case time:safe(T, NewClock) of
        true ->
          log(F, T, M),
          true;
        false ->
          false
        end
                                end,
        insert({From, Time, Msg}, Queue))
```

4 Lamport Time

Introduce logical time to the worker process. It should keep track of its counter and pass this along with any message it sends to other workers.

`safe()` is used to check whether it's safe to print a message in the log.

```
clock(Nodes) ->
  [{Node, zero()} || Node <- Nodes].

update(Node, Time, Clock) ->
  lists:keyreplace(Node, 1, Clock, {Node, Time}).

safe(Time, Clock) ->
  lists:all(fun({_, T}) -> leq(Time, T) end, Clock).
```

Start 4 workers and the logs printed like this:

```
1> test:run(500,500).
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}}
```

5 Vector Clock

Represent a vector clock by a list of process names and values.

Only change the implementation of `zero()`, `inc()`, `merge()` and `leq()` in the time module to be compatible with the vector clock. I implemented the rest part in a more general way so I can reuse them easily.

The structure of the `Time` for each worker and the `Clock` for the logger should look like this:

- Time - [{Node, Time}...]

- Clock - [{Node, [{Node, Time}...]}...]

Start 4 workers and the logs printed like this:

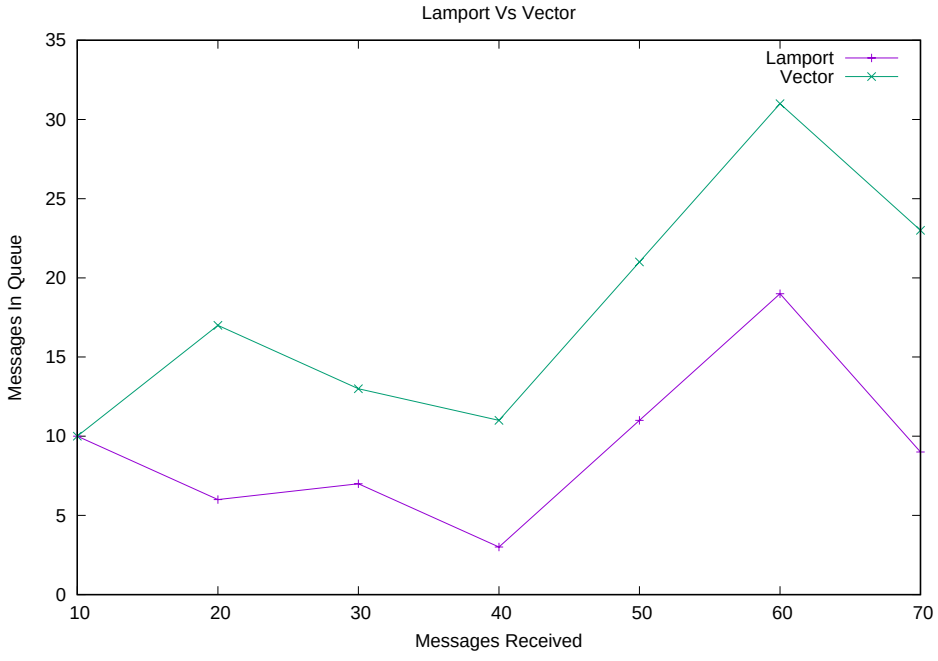
```
1> test:run(500,500).
log: [{john,1}] john {sending,{hello,57}}
log: [{ringo,1},{john,1}] ringo {received,{hello,57}}
log: [{ringo,2},{john,1}] ringo {sending,{hello,77}}
```

6 Test

Experiment and test with the logger.

At first there is no clock and logs are printed randomly. I experiment with the jitter and find that if I set sleep time longer than jitter, it will decrease the number of wrong entries. Try to imagine that every worker sleeps for a large amount of time before sending and receiving messages and the jitter just has little influence compared to the long waiting time.

To compare the Lamport algorithm and the Vector algorithm, I record the messages in queue when different numbers of messages have been received.



7 Conclusions

Loggy works well and two algorithms have been implemented and compared.

I experienced to build a logger and got familiar with the lamport clock and vector clock. It's pretty straight forward to implement each function and module, just step by step. The interesting part is how to do the erlang programming in a more elegant way, such as the use of `lists:all` in `safe()` and `lists:dropwhile` when deleting messages in the holdback queue.