Report 3: Loggy

Nagasudeep Vemula

September 25, 2019

1 Introduction

In this assignment we learn how to work with logical time in distributed systems with the help of a practical example.
We implement a logging procedure that is used to send processes from one place to another,which is not as straightforward as it seems when needed in an ordered fashion as each system will have its own time and this will lead to varied timestamps.
The processes are tagged with the Lamport time stamp of the workers and these events need to be ordered before being written to the stdout to manage flow of information.This solution can be done by using concepts of logical time which I have explained in the following sections.

2 Main problems and solutions

We are given the basic implementation of Loggy which spawns some processes and passes messages between them.Our task is to extend the above implementation by introducing the notion of Logical/Lamport Time which is a concept used to address the problem of synchronization in distributed systems.The solution also involves using a hold back queue in the module logger which keeps the messages back and prints them in time in the correct order.

2.1 Lamport Time

The algorithm of Lamport timestamps is a simple algorithm used to determine the order of events in a distributed computer system. As different nodes or processes will typically not be perfectly synchronized, this algorithm is used to provide a partial ordering of events with minimal overhead. (https://en.wikipedia.org/wiki/Lamport_timestamps, n.d.)

In order to implement the above algorithm we will need the processes to keep a track on their timestamp.We do this with a personal counter which can be updated on occurrence of the following events:-

- Receiving a message: We get the time counter from the message and compare it with the personal counter on receiving the message.In case the incoming time is greater than our time then we update the timer to take this new value and then increment our clock.
- Sending a message to another process.

The worker module is modified with the loop function adding a new argument in order to accommodate the above solution using the counter.Below is the code of the same:

```
loop(Name, Log, Peers, Sleep, Jitter,Counter)->
  Wait=random:uniform(Sleep),
  receive
    {msg,Time,Msg} ->
      NewTime= time_fin:inc(Name,time_fin:merge(Time,Counter)),
      Log ! {log, Name, NewTime, {received, Msg}},
      loop(Name, Log, Peers, Sleep, Jitter,NewTime);

    stop->
      ok;

    Error->

      Log ! {log, Name, time, {error, Error}}

  after Wait ->
    Selected = select(Peers),
    Time = time_fin:inc(Name,Counter),
    Message = {hello, random:uniform(100)},
    Selected ! {msg, Time, Message},
    jitter(Jitter),
    Log ! {log, Name, Time, {sending, Message}},
    loop(Name, Log, Peers, Sleep, Jitter,Time)
  end.
```

2.2 The Hold Queue

The hold queue is implemented to track the messages received and print them in the correct order of time. Lamport time helps us to order the messages but the printing of the message at the right time still has to be managed.Using an ordered list that can be updated,the following modification was made in the logger_fin module:-

```
loop(Queue, Clock) ->
  receive
    {log, From, Time, Msg} ->
      NewClock = time_fin:update(From, Time, Clock),
      NewQueue = queue_add(From, Time, Msg, Queue),
      NextQueue = log(NewQueue, NewClock),
      loop(NextQueue, NewClock);

    stop ->
      EmptyQueue = print_all(Queue),

      ok
  end.
```

Additionally we use the function safe(Time,Clock) to check if it is safe to log an event that happened at a given time.This is also implemented in the logger module as follows:-

```
log(Queue, Clock) ->
  lists:foldr(
    fun(Elem, List) ->
      {Name, Time, Msg} = Elem,
      Safe = time_fin:safe(Time, Clock),
      if (Safe) ->
        log_print(Name, Time, Msg),
        List;
        true -> [Elem | List]
      end
    end,
    [], Queue).
```

Together the above 2 implementations will form a hold queue that will make the messages of the ordered events print at the right time.

3 Evaluation

3.1 The Basic Logger

The first test is done using the implementation of loggy provided to us and it is easily observed that the message order is not correct with several of the messages are printed as received before being sent.

```
log: na ringo {received,{hello,57}}
log: na john {sending,{hello,57}}
log: na john {received,{hello,77}}
log: na paul {sending,{hello,68}}
log: na ringo {sending,{hello,77}}
log: na ringo {received,{hello,68}}
log: na ringo {received,{hello,58}}
log: na john {received,{hello,20}}
log: na paul {sending,{hello,20}}
```

It is seen from the first 5 printed messages itself that they are being displayes as received before sending.

3.2 Lamport Time Implemented but no Hold Queue

The following test is to show the result when the lamport time is implemented and with the counter now we can identify the messages printed in incorrect order.This is enabled by checking the Logical Time.

```
log: 3 george {received,{hello,16}}
log: 3 ringo {sending,{hello,68}}
log: 3 john {received,{hello,77}}
log: 3 paul {sending,{hello,16}}
```

We can see by checking the time for the above portion that the messages are in the wrong order.

3.3Lamport Time with Queue

The last test with the queue will print the messages in correct order with the logical time being displayed.

```
log: 1 george {sending,{hello,58}}
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 {received,{hello,58}}
log: 5 john {received,{hello,20}}
```

4.Conclusion

This assignment teaches us in a practical manner some of the issues that can arise in distributed systems in the concepts of time and information sharing.It also shows that even with all the benefits offered,concurrency does have its share of problems.I also got to learn about the concept of Lamport Time.