Loggy - a logical time logger - Report HW3

Malte Berg

September 24, 2025

1 Introduction

In this homework assignment, the primary topic covered has been:

- Understanding Lamport time

2 Main problems and solutions

The beginning of the `time` module was very easy to implement. The function `zero/0` does just what it looks like, returns a 0.

Next, the `inc/2` function takes in a name of a worker and a Lamport time for that worker which should be increased. While the name has no true purpose in this implementation, it is implemented in case one wants to improve the program further. The function returns a tuple {`Name, T+1`} where `Name` is the name of the worker, and `T+1` is the time `T` incremented by 1.

For the `merge/2` function, the BIF `max/2` was used, which returns the larger of the two Lamport times `Ti` and `Tj` given as the parameters.

`leq/2` was also simple to implement, returning the result of the comparison `Ti =< Tj`, which is true if the first parameter `Ti` is less than or equal to the second parameter `Tj` and false if it is more.

The `clock/1` function took some time to figure out the best approach, but having the `foldl/3` function from the `lists` library fresh in memory from the previous homework served to be a quick way of implementing the function as follows:

```
clock(Nodes) ->
    lists:foldl(fun(Node, Clock) -> [{Node, zero()} | Clock] end, [], Nodes).
```

This returns a clock consisting of tuples with the name of the worker and a 0 for its initial Lamport time. This method is dynamic, so it will work no matter the amount of workers.

The `update/3` function consists of a case expression finding the result of calling `lists:keyfind/3` on the given worker node. With the found Node

and its Lamport time, the current Lamport time is compared with `leq/2` with the given new time. If the new time is bigger, i.e. `leq/2` returns `true`, the Lamport time in the clock is updated with `lists:keyreplace/4`. If it returns false, the old clock is returned without change.

Finally for the `time` module, the function `safe/2` creates a list with all the Lamport times of the clock with the help of the `map/2` function from the `lists` library, mapping the tuple to just extract the Lamport time. This list is then fed into `lists:min/2` to get the smallest time, or the lowest Lamport time of the different workers. The given time `Time`, the first parameter of the function, is the compared to the minimum time that was found. The result of the comparison is then sent back, as it is safe to print the message as there is no Lamport time in the clock smaller than the given time.

3 Evaluation

The implementation of the `time` module as well as code written to update the logger implementation successfully makes the printing occur in order of the Lamport times. While the logs are ordered after their logical time/Lamport time, it is not sure that they are ordered after their real time occurences. Due to differing jitters and sleeps, and the fact that their is randomness involved in both of those times, there will be times where an event could have occurred not in the order of the Lamport times, as these are only updated when sending and receiving.

Testing with sleep times of 1, 10, 100, and 1000 ms as well as jitter times of 0, 1, 100, and 1000, my testing got a clear result that with a higher jitter value, a higher percentage of the total messages sat in the queue at one time than with lower jitter times, seen in Table 3. The queue length, seen as a percentage of the total amount of messages is also significantly decreased when the sleep duration decreases, getting the best results with a low sleep and jitter time.

Sleep (ms)	Jitter (ms)	Max queue length	Total messages	Max/Total (%)
1000	0	14	62	22.6
1000	1	25	62	40.3
1000	100	15	66	22.7
1000	1000	15	31	48.4
100	0	27	628	4.3
100	1	25	602	4.2
100	100	28	370	7.6
100	1000	22	78	28.2
10	0	30	4772	0.6
10	1	57	3986	1.4
10	100	37	683	5.4
10	1000	17	81	21.0
1	0	41	20000	0.2
1	1	39	9984	0.4
1	100	27	768	3.5
1	1000	22	80	27.5

Table 1: Queue metrics: percentage of total messages in queue at one time

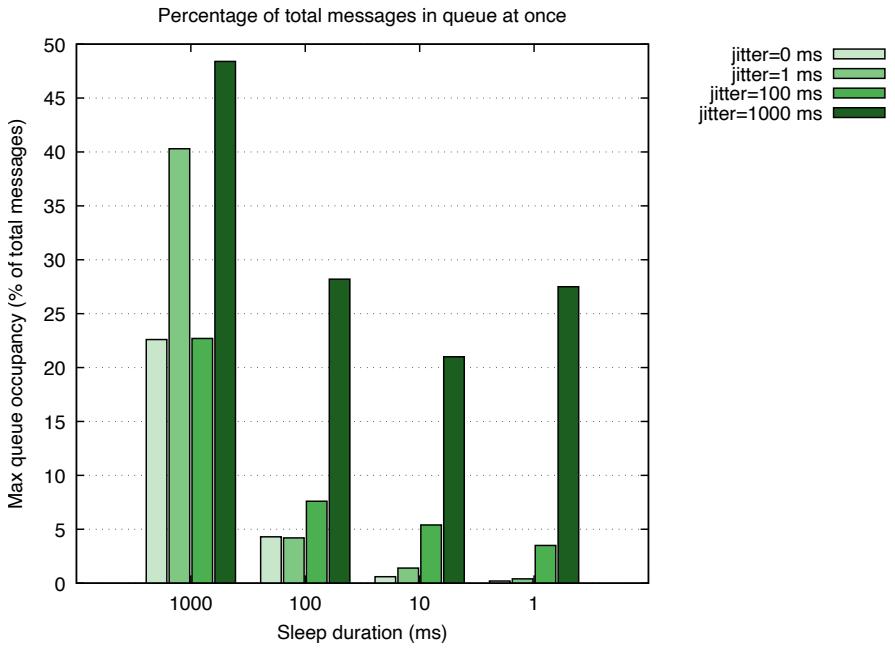

Figure 1: Bar graph showing max queue occupancy percentage