Loggy: a logical time logger

Mikael Karlsson

September 25, 2019

1 Introduction

In this seminar the task was to implement a logging procedure that receives log events from a set of workers. Each event are tagged with a Lamport time stamp of the worker and each event must be ordered before written to stout.

2 Main problems and solutions

I first implemented a Logger without any timestamps that accepts events and print them on the screen. I did a test with sleep = 2000 and jitter = 2000 just to observed whats happening without any logical clock. The result was:

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

Then a tried to set sleep = 2000 , jitter = 0, and the result was:

```
log: na john {sending,{hello,57}}
log: na ringo {received,{hello,57}}
log: na ringo {sending,{hello,77}}
log: na john {received,{hello,77}}
.
.
```

So when there is a "jitter delay" the messages is printed in a incorrect order. One way to solve this is to implement the Lamport algorithms.

The Lamport algorithm works as follows, for each message a worker sends, it increments its own clock by one. The message being sent has the structure:

```
{msg, Time, Message}
```

where Time is the workers clock, and Message is just the string 'hello' combined with a random integer. For every message a worker receives, it takes the highest number between its own timestamp and the timestamp that comes with the message and increments it by one and pass the message to the logger which is responsible for printing the messages in a correct order.

The logger has two important data structures, a queue where all incoming messages is stored and a list containing all the workers and their latest timestamp.

```
Message queue = [{ringo,2,{received,{hello,57}}},{paul,2,{sending,{hello,20}}}]
Worker list =  [{george,1},{paul,2},{ringo,5},{john,6}]
```

The algorithm iterates through the message queue and compare the timestamp for each message to lowest timestamp in the worker list. If the message timestamp is less or equal to the lowest timestamp in the worker list, the message is removed from the message queue and printed out to stout.

3 Evaluation

For evaluation a run several test too se that the messages are printed in the correct order. One test was to set sleep = 1000 and jitter = 2000, the result was:

```
log: 1 george {sending,{hello,58}}
log: 1 john {sending,{hello,57}}
log: 1 paul {sending,{hello,68}}
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 ringo {received,{hello,58}}
log: 5 john {sending,{hello,90}}
log: 6 ringo {sending,{hello,42}}
log: 6 paul {received,{hello,90}}
log: 7 paul {sending,{hello,40}}
log: 8 john {received,{hello,40}}
log: 9 john {received,{hello,42}}
```

.
.

As we can see, the messages are printed in the correct order.

4 Conclusions

I think this seminar displayed the issue of keeping track of time in a distributed system in a pedagogic way. It was interesting to see how the Lamport algorithm solves this problem and that it actually works in a practical example.