Report : Loggy - a logical time logger

Vasileios Giannokostas , vasgia@kth.se

October 1, 2014

1 Introduction

In this seminar , we learnt about the logical time in a distributed system. Because we cannot synchronize perfectly the clocks in a distributed system , we cannot use the physical time. Lamport defined the Lamport time Stamp to introduce the ordering and priority of 2 or more procedures. We can perceive this time Stamp as a number that given in each process. The comparison of two events considering their Lamport timestamp is easier now. We can infer that an event A that happened before the event B has lower timestamp that B (L(A)<L(B)). In contrast, we cannot infer that an event with a lower timestamp happened before another with a higher , this happens because the two events may are concurrent.

After this seminar we will able to :

- Understand the necessity to use logical time.
- Explain the Lamport timestamp logic and the happened-before relation.
- Understand the vector clocks and how they impove the Lamport timestamp logic.
- Practise in Erlang and functional programming are also benefits.

2 Main problems and solutions

*A*fter the small part of theory , it's time to implement the logical time to an example. The task is composed of two basic parts , the worker module and the logger module. We create many workers that send messages to each other and also send a report to the logger to print out the messages.

Our task is to implement the logical time to the workers and logger in order to avoid the inconsistency of messages and give them ordering. With the given code and without upgrades the logger receives and prints messages in the wrong order, for instance the logger prints out that a worker A receives a message M from a worker B , and after this message the logger prints out that the B sends the M to A.

log: ringo na {received,{hello,57}
log: george na {sending,{hello,57}}

In order to "deteriorate" the problem and show that the logger doesn't print with the right order, we introduce the jitter value that is the delay between the sending a message to a worker and the sending the same message to the logger. This random delay makes the fact that a "sending" message arrives to the logger after the "received" message.

3 Evaluation

We implement the logical time to our task and made some improvements to our system.

The easy way: The first step was to introduce timestamps in every message.The logger saves every message for later and considering their timestamps prints them in the right order at the end of the program. We notice that the logger prints out the right order of the messages.

Additional improvement: The Lamport logic works in our system but we can also improve this to print out on fly. The logger checks it's queue if it's safe to print out any message every time that a message arrives to it, the logger print out only if it's sure that a message must be print and it continues to receive other messages and put them to the queue.

How do we know if messages are safe to print?

We check the minimum timestamp of all workers(Min) and print out the messages with timestamp equal or lower to the Min.

Did you detect entries out of order and if so how did you detect them?

I detected entries out of order only in the initial version(without implemented the logical time). I detected them by noticing that a received message is printed earlier that a sending message (the unique random number

works as an Id to detect the messages). In the upgraded versions the logger prints the messages properly.

4 Conclusions

*T*his seminar taught us the necessity to give logical time to machines. We understood that the physical time is almost impossibly to implement to the machine because the machines have different physical times. We implemented a simple example that show us the power of the logical time to cover the vulnerability of the systems in the time. Also another seminar practicing in Erlang and functional programming is also good.