Loggy: a logical time logger

Chuan Su

September 25, 2018

1 Introduction

In a distributed system environment, we cannot rely on physical clock/time to determin the order of events ocurring across it.

However with logical clock we are able to find out the **partial ordering** of any arbitary pair of events ocurred in different processes if there was a message sent between them, which is also known as happens-before relation in distributed systems.

In this assignment our task is to implement a logging procedure to print the event messages concurrently sent from a set of worker processes in order.

2 Main Problem and Solution

With the initial logger server implementation event messages that are printed to stdout are unordered. /"Unordered"/ does not imply the total ordering of all the messages but partial ordering of certain message pairs.

For any arbitary message pair (messages with unique identifier) the `sent` event message should always be printed before `receive`. In our improvement of loggy implementation, Lamport Time, an approach of Logical Time was adopted to solve this ordering issue, that is:

- Sender node: Before each message is sent out, sender node's counter is incremented by 1. The value of the counter is sent with the message to the receiver node.
- Receiver node: On recieving a message, receiver node computes the max value between the sender and its own counter and then increments it by 1.

The counter on each node is called *Lamport Time* and messages sent between Lamport nodes *are tagged with Lamport Timestamp.*

With the understanding of Lamport Time, we addressed the message disordering problem by

- On recieving a message, instead of writing message directly Loggy server put the message into a message queue.
- Loggy server maintains a central clock which keeps track of the Lamport timestamp received from each worker node.
- Loggy server loops through each message in the queue and print out the messages whose timestamp is less or equal to the minst node timestamp in the contral clock.

It is remarkable that Lamport Time approach doesnot solve the issue of total ordering of each message but guarantees that for each message pair the `send event` is logged before (happens before) the `receive event` regardlees the physical time they arrives at loggy server.

In addition, we performed an evaluation on the maximum message queue size that Loggy have maintained during the logging procedure. Through repeated tests with adjusting the values for `sleep` and `jigger` variable, the result we found was 35.

3 Vector Clocks

With Lamport clock we are unable to conclude that an event *happens-before* another by only looking at their lamport timestamps, which is not very helpful in the circumstances of a distributed system.

Vector clocks were invented to overcome this shortcoming to enable us to identify the *happens-before* relation of adjacent events through logical timestamps.

In our loggy server implementation we provides the approach of Vector clock:

On receiveing a message, Loggy put the message into its message queue as in Lamport approach. Likewise Loggy server maintains the central clock for each node's logical timestamp.

One difference between the vector clock appraoch and lamport time approach lies on syncronizing logical timestamp between server clock and node/worker clock. As the timstamp tagged in lamport message is a single

value which is straightforward to merge into server clock while vector timestamp includes not only the Time from itself but the clock of every other worker, which requires Loggy to extract the coresponded Time value from the received vector first.

Another difference is how Loggy assess messages /"Safty"/ (safe to print the message in order) from its message queue. In our vector clock approach instead of calculating the minimal Time among nodes in the central clock Loggy simply compares the central clock with each message timestamp in the queue and filters out the ones that are less or equal to the central clock, which are the safe messages to print out.

The message safty evaluation is done by the following code snippet:

- `Vj` is the central clock that loggy maintains
- `Vi` is the vector timestamp sent from worker nodes.

```
%% V <= V' if V[j] <= V'[j] for j = 1,2...,N
leq(Vi, Vj) ->
  V = lists:zipwith(fun({P, Ti}, {P, Tj}) -> {P, Ti =< Tj} end, Vi, Vj),
  lists:all(fun({_, Leq}) -> Leq end, V).
```

Now running the test program `test:run(5,20)` we can easily find out that the messages are written to stdout in order.

4 Conclusion

The concept of Logical Time was easy to understand at first sight. However it requires efforts of evaluation to be able to grasp the total idea, especially the ground that Lamport Time was proposed and further Vector Clocks.