Report Loggy: a logical time logger

Anestos Nikolaos - Ektoras

October 1, 2014

1 Introduction

In this exercise, we implement a logging system that uses Lamport timestamps to synchronize it's nodes. Main goal of this exercise is to understand how logical time could solve issues caused by asynchronous nodes in a distributed system.

2 Main problems and solutions

We have a set of workers which send messages to each other. Each time a worker sends or receives a message, he informs the logger and the logger prints it to stdout.

Each worker waits for an arbitrary amount of time to receive a message. If he timeouts, he sends a message to another random worker. After that he waits for some random time (jitter) and informs the logger about that message. If we introduce a large value in Jitter, we can see that the logger prints the events in an abnormal order. He first prints the event from the worker receiving the message and then the one from the worker sending it. This is of course not the right order things happened in our system and we need to fix it.

3 Evaluation

Running the skeleton code, one can see almost immediately (using a high Jitter value on run function) that the logger has trouble printing the events in the correct order. In fact, since Jitter reflects on the delay the worker has between sending a message and informing the logger, we can see several messages being received before sent! To change that we have to introduce logical time to the system.

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

*E*xample of events logged in the wrong order.

3.1 Adding logical time to the workers

At first we have to make our workers aware of time.

We start by adding a variable (LamportClock) to the worker loop function, starting the loop with 0 to this variable and adding plus one for each message we send. When we receive a message, we check the "local time" and the time the message was sent and if it the latter is greater, we update our clock to that time plus one.

3.2 This is not enough

Even though workers are now updating correctly their "local time", we still haven't solved anything because the logger still prints messages in a FIFO order. If the Jitter is high enough, the logger violates the "Happened Before" order of messages.

```
log: paul 6 {received,{hello,john,95}}
log: ringo 3 {sending,{hello,ringo,80}}
log: ringo 4 {received,{hello,paul,69}}
```

*I*ntroducing Lamport Timestamp to the workers.

3.3 Updating Logger to understand logical time

Since we don't want to print messages unless we are safe to do so, we need to store them.

3.3.1 The lazy way

I queued the messages using a list which contains a tuple with information about the message the logger receives (From, Time and Msg). At first, i printed messages only when the logger stops (sort the list, print it recursively and we are done). Of course this is a simple solution, we need the logger to be able to print messages while still running.

3.3.2 A more sofisticated solution

We need the logger print messages while still available for receiving more, so it should have knowledge of each worker's time and decide which messages to print from the queue.

To solve this, i updated a name-timestamp (key-value) map of the workers each time the logger receives a message. For example when worker A sends a message to the logger, the logger updates the timestamp in the map to A worker's clock. This way the logger will always know which messages are safe to print.

Having all four Lamport timestamps, we know that it it safe to print all messages with timestamp lower than the lowest of the four. For example if all four workers have sent messages with timestamp greater than 5, the logger knows that it is safe to print all messages in the queue with timestamp lower or equal to 5.

```
log: john 1 {sending,{hello,john,58}}
log: paul 1 {sending,{hello,paul,69}}
log: nick 1 {sending,{hello,nick,24}}
log: george 2 {received,{hello,john,58}}
log: ringo 2 {received,{hello,paul,69}}
log: george 3 {received,{hello,nick,24}}
log: ringo 3 {sending,{hello,ringo,80}}
```

*S*ave your head from the King.

3.3.3 Give it some thought

The next problem is to decide when to print messages (or check if we have something to print).

We can tell the logger to check for safe to print messages each time it receives a message. The workload of the logger process is then depending on the number of workers and messages they send.

A second solution might be to add a small timeout to the logger, and ask it to check for safe to print messages.

3.4 What about Vector Clocks?

I tried to implement the Vector Clocks solution in this problem. It is a bit tricky and i wasn't able to finish it in time to write this report. My main concern is should the logger keep track of all clocks and remember which message from each worker he printed last?

I hope that the seminar will give me a slight push to complete this task and understand what advantages or disadvantages we might have.

4 Conclusions

This exercise helped me understand why logical time is important in a distributed system.

We know at this point, that messages with the same time stamp might not really happen in the same order that they are logged. What matters in this system though, is causality; we should never violate the happened before order.