Loggy: Seminar Report Template

Weherage, Pradeep Peiris

September 30, 2015

1 Introduction

Loggy is a simple logger procedure that receives log messages from multiple worker processes.

This report explains a practical use of Logical Clock in finding order for arbitrary events of worker processes, which try to simulate a distributed environment.

2 Main problems and solutions

It is an important factor in distributed systems to determine the order of events occurred across the nodes of a distributed system. The need could be more application level such as set of transaction events and their execution order to make business decisions. Or it could be more system level need such as distributed garbage collection.

The physical time is a problematic for event timestamps, due to the fact that clocks are deviated and cannot be synchronized perfectly across nodes.

Lamport Logical timestamps provides a simple mechanism for identifying the order of events in a distributed system.

In Lamport Logical clock, the state in between nodes are not maintained. If we have events that e $->$ e' then Lamport clock value should be L(e) $<$ L(e'). But L(e) $<$ L(e') do not necessary means e $->$ e' or, e and e' are concurrent.

Vector clocks resolve this problem maintaining state of all nodes in the system. That is, it stores a vector of timestamps equal in size to the number of nodes in the system.

3 Evaluation

The first implementation of Loggy simply logs the received messages from multiple arbitary Worker processes. The log message contains the Worker name, event timestamps and process message.

Our test case compose of 4 Worker processes connected with each other. They are designed to send messages with a delay of 'jitter' value. It was evaluated that higher 'jitter' value causes Loggy to receive log messages in arbitrary order.

3.1 Message Ordering with Lamport logical clock

Now worker process is introduced with Lamport Logical Time. That is, each worker process maintains a Lamport clock as the following logic.

1. Lamport value, Li is incremented before Worker Wi sending messages.

   ```
   Li := Li + 1.
   ```

2. On receiving at Worker Wj, it computes max(Li, Lj). And increment the output value prior to timestamping the received message.

   ```
   Lj := max(Lj, Li),
   Lj := Lj + 1.
   ```

Loggy is improved to log the messages in Worker's sending/receiving order with the following logic.

1. Loggy maintains a ClockMap, which contains the maximum clock value for each Worker messages.

2. Message is appended to the hold-back queue

   ```
   lists:append([{From, Time, Msg}], HoldbackQueue),
   ```

3. Loop through all messages in hold-back queue, and compared with ClockMap. The message is log out only if its clock value is less than or equal to all clock values in ClockMap.

   ```
   update(HoldbackQueue, Clock) ->
     lists:foldl(fun({From, Time, Msg}, UpdatedHoldbackQueue) ->
       case (time:safe(Time, Clock)) of
         true ->
           log(From, Time, Msg),
           UpdatedHoldbackQueue;
         false ->
           UpdatedHoldbackQueueForSort =
             lists:append([{From, Time, Msg}], UpdatedHoldbackQueue),
             lists:sort(fun({_, X, _}, {_, Y, _}) -> Y > X end,
             UpdatedHoldbackQueueForSort)
       end
   end, [], HoldbackQueue).
   ```

It is evaluated in most of the time Loggy logs the messages in correct order with Lamport Logical time implementation. But still there is a chance of incorrect order as hold-back queue grows with number of messages. This was simply solved making hold-back a sorted queue on clock values.

3.2 Message Ordering with Vector clock

The Worker can be simply replaced with Vector clock implementation.

```
inc(Node, Clock) ->
  {Node, T} = lists:keyfind(Node, 1, Clock),
  ClockUpdated = lists:keydelete(Node, 1, Clock),
  lists:append([{Node, T + 1}], ClockUpdated).

merge(Ci, Cj) ->
  lists:foldl(fun({Node, Ti}, C) ->
   {Node, Tj} = lists:keyfind(Node, 1, Cj),
   if
     Ti > Tj -> lists:append([{Node, Ti}], C);
     Ti < Tj -> lists:append([{Node, Tj}], C);
     Ti == Tj -> lists:append([{Node, Ti}], C)
   end
end, [], Ci).
```

The Loggy can be changed to follow the same execution flow as before calling Vector Clock API.

4 Conclusions

With set of Worker modules we tried to simulate a real distributed environment of processes. The messages of each Worker process is timstamped with Lamport Logical clock value. And Loggy is implemented with a logic to use this timestamp value to order the execution of events/messages.

The experiment was quite successful as we managed to order the messages of distributed processes using its Lamport Logical time. Hold-back queue growth may causes in-order of messages. This was solved with sorted Hold-back queue.