Report HW3: Loggy – a logical time logger

Lakshmi Srinidh Pachabotla

September 25, 2024

1. Introduction

Using Lamport clocks to synchronize logical time, the Loggy system is a distributed logging method designed for handling messages from several worker processes. Employees communicate through a logger process that records the events and makes sure the messages are arranged appropriately according to their logical timestamps. Despite the possibility of out-of-order message delivery due to the distributed configuration, the system attempts to guarantee that messages are displayed in the correct sequence, representing logical time. To manage out-of-order occurrences, this exercise explores the actual application of distributed systems principles such as Lamport clocks, message ordering, and holdback queues. The main goal is to emulate a distributed logging system that is important to causality and event sequencing in real life.

2. Main Problems faced and Solutions

During the development of Loggy, several challenged were faces, which are handled carefully to get the required output:

- The main problem faced while implementing was to ensure that each worker maintained its Lamport clock, incrementing the clock with each send and receive event. To solve this problem, a “time” module was implemented which is used to manage the Lamport clocks with the help of some operations like “inc”, “merge”, and “safe”. Before delivering and after receiving messages, employees adjusted

their clocks. To verify whether a message should be in holdback queue, or it should be logged immediately, logger used “safe” functions.

```
% Increment the Lamport time by 1
inc(_Name, T) -> T + 1.
```

Figure 1: inc operation

```
% Merge two Lamport clocks (take the max)
merge(Ti, Tj) -> max(Ti, Tj).
```

Figure 2: merge operation

```
% Check if a time is safe to log
safe(Time, Clock) ->
    % Safe if all nodes have logged up to the given time
    lists:all(fun({_, NodeTime}) -> Time =< NodeTime end, Clock).
```

Figure 3: safe operation

- The second problem is with the message arrival times. Some messages times may vary, and some may arrive out of order. This required implementing a holdback queue to temporarily store messages that could not be logger immediately in a safe manner. To counter this problem, the logger was updated to manage a queue of incoming messages processing and printing them only when it was safe. The holdback queue ensured that messages with earlier timestamps were handled first, guaranteeing that no message was printed before all previous ones had been logged.

3. Evaluation

The system’s final test showed that Loddy effectively preserved message order and accurately handled Lamport clocks. A group of workers (John, Paul, George, and Ringo)

exchanged messages, and the logger printed them in the correct sequence according to their logical timestamps.

- The clocks of each worker were appropriately set up and increased with each transmit and receive event. To prevent a message from printing ahead of a previous message with a lesser timestamp, the logger appropriately managed the Lamport clocks. John and George's messages, for instance, were printed in the right manner with none of them written out of order.
- Messages that were unable to be printed right away were stored in the holdback queue, just as planned. The logger handled the waiting list in the right sequence when it seemed safe for printing the messages (that is, when there were no more messages to be processed). The final product made this clear by printing the occurrences in a logical temporal sequence that increased with each passing moment.
- The setup was tested with varying jitter levels to mimic message delivery unpredictability and network delays. The Lamport clock system and holdback queue allowed the logger to reliably retain the right message sequence even in these settings.

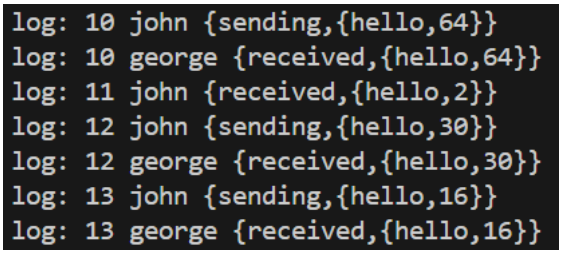

Figure 4: Generated Output

- Here, the logs from two distinct workers John and George are presented in a sequence that makes sense, demonstrating how well the holdback queue controlled message delivery.

4. Conclusion

In summary, the Loddy system demonstrates a practical application of using logical time to sequence events in a distributed system. By carefully managing message queues, it ensures that events are processed in the correct order. This implementation provides basis for exploring more advanced challenges in distributed systems, such as those involving vector clocks or other logical time mechanisms.