Report 3: Loggy: A Logical Time Logger

Luxiang Lin

September 24, 2024

1 Introduction

Through this assignment, I was expected to gain a deeper understanding of the concept of logical time, which plays a crucial role in distributed systems. As part of the homework, I have implemented both the Lamport Timestamp algorithm and the Vector Clock, two fundamental methods for tracking and maintaining the causal order of events in a distributed environment. These implementations have helped me learn how logical time is used to coordinate processes without relying on physical clocks.

2 Main problems and solutions

The logic of the Lamport algorithm is to have each node manage its own clock, updating it every time it sends or receives a message, or performs a local action. Additionally, the logger should maintain a holdback queue to store any messages that are not yet safe to process. By following these rules, we can ensure that if $e_1 \to e_2$ (i.e., event e_1 causally affects event e_2), then e_1 must occur before e_2.

3 Evaluation

3.1 First Try

In the first experiment, I ran the test without applying the Lamport algorithm, and the output appeared to be unordered and inconsistent:

```
5> test:run(1000, 500).
log: 2 c {received,{hello,57}}
log: 1 a {sending,{hello,57}}
log: 4 a {received,{hello,77}}
log: 1 b {sending,{hello,68}}
log: 3 c {sending,{hello,77}}
log: 4 c {received,{hello,68}}
```

```
log: 5 a {received,{hello,20}}
log: 5 c {received,{hello,58}}
log: 2 b {sending,{hello,20}}
log: 6 a {sending,{hello,84}}
```

The incorrect ordering in the output can be easily detected. For instance, 'log: 2 c {received,{hello,57}}' appears before 'log: 1 a {sending,{hello,57}}'. The final log '{hello,57}' shows that this is a pair of 'send' and 'receive' actions, making it obvious that these two outputs are in the wrong order. Another instance is 'log: 4 a {received,{hello,77}}' appearing before 'log: 3 c {sending,{hello,77}}'.

3.2 Lamport Time

Next, I carried out a test to observe the output order after implementing the Lamport algorithm. Here are the results:

```
3> test:run(1000, 500).
log: 1 d {sending,{hello,58}}
log: 1 b {sending,{hello,68}}
log: 1 a {sending,{hello,57}}
log: 2 b {sending,{hello,20}}
log: 2 c {received,{hello,57}}
log: 3 b {sending,{hello,16}}
log: 3 c {sending,{hello,77}}
log: 4 c {received,{hello,68}}
log: 4 a {received,{hello,77}}
log: 5 c {sending,{hello,20}}
log: 5 a {received,{hello,20}}
log: 6 d {received,{hello,20}}
log: 6 a {sending,{hello,84}}
log: 6 c {received,{hello,58}}
```

Now that the outputs with the same identification code are in the correct order, the message sequence appears correct. However, due to the limitations of the Lamport algorithm, the program cannot determine the order of concurrent messages.

3.3 Vector Clock

Finally, I implemented the vector clock and conducted a test. Below is the output of the vector clock:

```
5> vtest:run(1000, 500).
log: [{d,0},{c,0},{b,0},{a,1}] a {sending,{hello,57}}
log: [{d,0},{c,0},{b,1},{a,0}] b {sending,{hello,68}}
log: [{d,0},{c,0},{b,2},{a,0}] b {sending,{hello,20}}
log: [{d,0},{c,0},{b,3},{a,0}] b {sending,{hello,16}}
log: [{d,0},{c,1},{b,0},{a,1}] c {received,{hello,57}}
log: [{d,0},{c,2},{b,0},{a,1}] c {sending,{hello,77}}
log: [{d,0},{c,2},{b,0},{a,2}] a {received,{hello,77}}
log: [{d,0},{c,2},{b,2},{a,3}] a {received,{hello,20}}
log: [{d,0},{c,2},{b,2},{a,4}] a {sending,{hello,84}}
log: [{d,0},{c,2},{b,2},{a,5}] a {sending,{hello,7}}
log: [{d,0},{c,2},{b,4},{a,5}] b {received,{hello,7}}
```

The order of the outputs is correct, indicating that the implementation of the vector clock was successful.

4 Conclusions

Through this assignment, I successfully implemented both the Lamport Timestamp algorithm and the Vector Clock, gaining practical experience in managing logical time in distributed systems. These implementations have deepened my understanding of how logical time is used to coordinate distributed processes.