Distributed System SE 1

Hou, Jue

014695647

jue,hou@helsinki.fi

1. Lamport clocks

Simulation:

A

B

C

19

40

28

46

47

23

32

48

39

49

50

49

20

48

2. More Lamport clocks

Proof:

If then :

Case 1: Both and happened on . In this case, the clock of former one is obviously larger than the latter one.

Case 2: Case 2: happened on and happened on . If and was local event on each process and both happened after or before sending and receiving message on each side, they could be concurrent, which does not meet the condition . If happened before sending event and happened after receiving, . x denotes for the time of local event on after receiving. If was the receiving event, x=1. And the property that if then holds for these reasons.

does not necessarily imply holds:

On the other hand, as what is showed in Q1, after B ticked 6, A ticked 3. And the clock of A was 23 and the clock of B was 46, which means that even though ticking for A happened after ticking for B, the clock of B was still larger than A. For this reason, the property that does not necessarily imply holds.

In conclusion, both of the properties of Lamport’s clock hold.

3. Vector clocks

Solution:

1. There are three processes working and communicating with each other. The local timestamp recorded by P2 for itself was 5 before receiving the message. The timestamp recorded by P1 for itself was 8 when sending the message. The timestamp recorded by P3 for itself was at least 2. P3 once sent a message of timestamp to P2. And then it sent another message of timestamp to P1.
2. When sending the message, P1 was currently 8 and P3 was recorded as 2.
3. Assume that and is two processes working and communicating with each other. According to the vector clock algorithm, unlike or which will be incremental for local events on each process, will not increase until there was a message sending from . After receiving the message of timestamp from , updated the value with . denotes for the vector in the message.

If there was no further local event on after sending message, .

If there had already been some local events on , which also means that , then .

For these reasons, the relation always holds.

1. Case 1: Both and happened on . There would be and . N denotes for the number of processes. According to the definition for , .

Case 2: happened on and happened on . If and was local event on each process and both happened after or before sending and receiving message on each side, they could be concurrent, which does not meet the condition . If was a local event happened before sending message and happened after receiving, and . x denotes for the time of local events happened between and receiving message event. So . If was the receiving event, . Either way, .

In conclusion,

1. Simulation:

A

B

C

(19, 0, 0)

(0, 40, 0)

(0, 0, 28)

(0, 46, 0)

(19, 47, 0)

(22, 0, 0)

(0, 0, 32)

(19, 46, 33)

(23, 46, 33)

4. Clock synchronization

Solution:

Assumption for all the scenarios: No information will lost when message is passed through network and the Internet.

1. If computers are physically close to each other and no computer will crush, I will use Christian's algorithm. Also, in this scenario, clocks are only required to be closely synchronized instead of being “correct” if compared to things like UTC. Since Christian's algorithm can provide a relatively easy solution for synchronization. And physically closing to each other means that the RTT is very small, which also means that the precision of time is good. Also the computer selected to be the time server will not crush so that the algorithm will always get every clock to the almost same time.
2. If computers will crush in a sudden but still they are relatively close to each other, I will apply Berkeley algorithm. As the same as the first scenario, clocks do not need to be “correct”. Because the master computer will frequently request for the time from slave computers and return an average time. The slave will react accordingly. Also, some extremely large or small time will get ignore so that the average time is under control. Even though the master crushed, the next master will still make the right average time according to the algorithm.
3. If the computers are widely separate or clocks need to be “correct”, I will apply NTP. NTP forms a hierarchical structure. The first layer of NTP gets the time from a solid source. If clocks are required to be “correct”, this source can be set to GPS satellites or any other requested source. The nodes in each layer will refer to the best candidate and synchronize to it. Other candidates will also be taken into consideration to minimize errors. Even though some nodes become unavailable, other nodes will give estimation according to the past. So NTP will get the computers located in different place as closely synchronized as possible.