Distsys Small Exercise 1
Clocks

Kalle V.M. Viiri

November 2, 2016

1 Lamport clocks

Simulate the Lamport clock algorithm for three processes A, B, and C. Present each process's local clocks given the following order of events:

A snd B;
B ticks 6;
A ticks 3;
B rcv A;
B snd C;
C ticks 4;
C rcv B;
C snd A;
A ticks 16;
A rcv C;
C ticks 1;

1.1 Solution

The values of each clock after each event are in the following table:

Event	**A**	**B**	**C**
Initial	19	40	28
A snd B	20	40	28
B ticks 6	20	46	28
B rcv A	20	47	28
B snd C	20	48	28
C ticks 4	20	48	32
C rcv B	20	48	49
C snd A	20	48	50
A ticks 16	36	48	50
A rcv C	51	48	50
C ticks 1	51	48	51

The final values for the three clocks are A = 51, B = 48, C = 51.

2 More Lamport clocks

Prove the following two properties for Lamport's cloks:

1. If $e \to e'$, then $L(e) < L(e')$

2. $L(e) < L(e')$ does not necessarily imply $e \to e'$

2.1 Solution

When $e \to e'$, any possible sequence of events between e and e' is composed of local events within the same process, or send and receive events between different processes. Local events always increment the local clock, so therefore clearly preserve the clock condition. Send and receive events also preserve the clock condition: a message is always sent before it is received, and the receiving process always sets its clock one greater than the received value. Therefore, if $e \to e'$, it also must be true that $L(e) < L(e')$.

The second claim can be proven by counterexample: suppose e and e' occur on two distinct processes that never communicate with each other (or a common intermediary process). Then e and e' are concurrent, but may have any clock values without regard to the other, so $L(e) < L(e')$ is possible even if $e \nrightarrow e'$.

A slightly less trivial counterexample from the first exercise is events 7 and 8 (C snd A and A ticks 16, respectively). These events have no causal relationship to each other, and while event 8 has a lower timestamp (36 versus 50) that doesn't guarantee it happening before C sent a message to A.

3 Vector clocks

Assume that when P2's clock was $(4, 5, 1)$ it received a message from P1. The timestamp of the message was $(8, 3, 2)$. Given that knowledge, answer the following questions:

1. What can we tell about the overall system?

2. What did P2 learn about the history of its colleagues?

3. Show that the relation $V_j[i] \leq V_i[i]$ always holds.

4. Show that $e \rightarrow e' \Rightarrow V(e) < V(e')$.

5. Solve Lamport clock problem in Q1 using vector clocks.

3.1 Solution

What can we tell about the overall system? Since there are three elements in each vector, there are three processes in total.

What did P2 learn about the history of its colleagues? That there has been previous message traffic from P3 to P1 (for $V_1[3]$ is incremented to two).

Show that the relation $V_j[i] \leq V_i[i]$ always holds. The only way $V_j[i]$ can be increased is through receiving a greater value from another process, as only process i can increment it directly. This means no process other than i can have the greatest value for $V[i]$.

Show that $e \rightarrow e' \Rightarrow V(e) < V(e')$. By the definition of e happening before e', there has to be a sequence of events between e and e' composed of local events within the same process, or send and receive events in between. Local events always increment the local clock, so local events preserve the clock condition. Send and receive also preserve the condition: the received message always gets a greater vector than the sent vector was (seen above). Therefore for any sequence of events between e and e' results in $V(e) < V(e')$.

Solve Lamport clock problem in Q1 using vector clocks. The table below shows the states of each vector after each action:

Event	**A**	**B**	**C**
Initial	(19, 0, 0)	(0, 40, 0)	(0, 0, 28)
A snd B	(20, 0, 0)	(0, 40, 0)	(0, 0, 28)
B ticks 6	(20, 0, 0)	(0, 46, 0)	(0, 0, 28)
B rcv A	(20, 0, 0)	(20, 47, 0)	(0, 0, 28)
B snd C	(20, 0, 0)	(20, 48, 0)	(0, 0, 28)
C ticks 4	(20, 0, 0)	(20, 48, 0)	(0, 0, 32)
C rcv B	(20, 0, 0)	(20, 48, 0)	(20, 48, 33)
C snd A	(20, 0, 0)	(20, 48, 0)	(20, 48, 34)
A ticks 16	(36, 0, 0)	(20, 48, 0)	(20, 48, 34)
A rcv C	(37, 48, 34)	(20, 48, 0)	(20, 48, 34)
C ticks 1	(37, 48, 34)	(20, 48, 0)	(20, 48, 35)

The final values for the clocks are: A = (37, 48, 34), B = (20, 48, 0) and C = (20, 48,

35).

4 Clock synchronization

Consider a scenario with multiple wall clocks showing different times. Each clock is controlled by its own computer which is connected to the rest of the network as well as the Internet. How would you synchronize the individual clocks? The goal is to have them be as closely synchronized to each other as possible.

In your answer, consider the merits of Cristian's algorithm, Berkeley algorithm, and NTP, and make a case for each of them why that algorithm would be the best one for this scenario. You can make additional assumptions about the operating environment, but please state them explicitly.

4.1 Solution

Cristian's algorithm. Cristian's algorithm requires an agreed-upon time server, so one of the clocks has to be designated as the master. To synchronize clocks, each servant clock would poll the master for the time. The master replies, and the servant clocks set their own clocks assuming that the master's timestamp occured halfway through the round trip.

In reality, the master's reply could've happened at any point during the round trip. For example, if RTT (round-trip time) is ten seconds, it could be that the initial query took nine seconds to reach the master and the reply took only one second to get back to the servant. In this case, the supposed time is four seconds off the actual time of the master. The results turn out the best when the RTT is very short, since the RTT is a strict upper bound on the offset of the time after synchronization. Therefore it's a recommended solution, but only if network latency is reasonably and consistently low.

Berkeley algorithm. In the Berkeley algorithm, a master is first elected (for example by using the Bully algorithm). The master first establishes the time of the other clocks by first polling their times in a similar manner to Cristian's algorithm. The master chooses the average of the time estimates as its time, ignoring any clear outliers, and tells each other clock to shift backwards or forwards to achieve the desired time. If a backward shift is needed, the receiving clock is usually slowed down to correct instead of shifted right away, because time progressing backwards could break software relying on normal properties of time applying.

Since the Berkeley algorithm picks the average time of each servant clock as the official timestamp, it sticks closer to the original times than Cristian. Any deviation from the eventual "official" time is, like in Cristian's algorithm, a result of an uneven round trip length between master polling each servant and the servants replying. As the reply is structured as difference to the timestamp received, not the resulting time, the RTT for actually setting the clocks has no effect on the clock synchronization.

NTP. Network Time Protocol (NTP) would mean connecting the entire set of clocks to an NTP server, or preferably, a set of such servers. Each clock would query the available NTP servers for a time, and use statistical analysis to get the correct time. To use the publicly available NTP clocks (high-precision atomic clocks) the clocks need internet connection, which according to the assignment is available.

One alternative is to create one's own NTP network inside the classroom where the clocks are. This would mean deciding a hierarchy between the clocks. Preferably just one clock would be on the zeroth stratum (meaning "official time"), since we can't assume these clocks to have the precision of high-end atomic clocks and thus two zeroth stratum clocks would possibly deviate from each other. The other clocks would be formed into a hierarchical tree (strata) according to the NTP protocol so each layer gets its time synchronized with the stratum "above" (closer to stratum zero) of its own.

NTP treats the received timestamps more carefully than Cristian and Berkeley algorithms: instead of just round trip time, each querying node also receives "processing time" stamps indicating how long the time server took to process the time request. These are further processed statistically by the receiving node to mitigate inaccurate timestamps. This procedure refines NTP's results to be much more accurate than Cristian and Berkeley algorithms, which is why I would prefer NTP in the described scenario.