

Excercise Sheet 8

Solution

Lecture Distributed Systems

Winter Term 2025/26

Exercise 1: Clock Synchronization

- a) The client should select the request with the minimum round trip time of $20ms = 0.02s$. If it can set its clock immediately after the response arrives, it would set its clock to $t_{Server} + T_{round}/2$, i.e. $11:51:25.232 + 0.02/2 = 11:51:25.242$. Since the client can usually only determine the answer with the minimum round trip time later (after some requests), it must proceed as follows: Suppose the local time of the client is t . It then sets the clock to $t + (t_{server} + T_{round}/2 - t_{arrival}) = t + (t_{server} + T_{round}/2 - (t_{send} + T_{round})) = t + 25.242 - 24.734 = t + 0.508$, i.e. the clock is advanced by $508ms$.

The accuracy is $\pm 10ms$.

If the minimum transfer time is $8ms$, then the setting remains the same, but the accuracy improves to $\pm 2ms$.

- b) To synchronize the clock with an accuracy of $1ms$, if a transmission time of at least $8ms$ is given, the maximum round trip time is $T_{round} = 2 \cdot 1ms + 2 \cdot 8ms = 18ms$.

Exercise 2: Happened-Before Relation, Lamport and Vector Time

- a) From the fact that $L(a_3) < L(c_2)$, it can **not** be concluded that $a_3 \rightarrow c_2$ holds. However, from $L(a_3) < L(c_2)$ it follows that $L(c_2) \not< L(a_3)$ and therefore $c_2 \not\rightarrow a_3$, so either a_3 (in the sense of Lamport's *happened before* relation) was before c_2 or the two events are concurrent. A statement about the sequence in real time is not possible.
- From $L(a_1) = L(b_1)$ it can be concluded (since $L(a_1) \not< L(b_1)$) that $a_1 \not\rightarrow b_1$, so either b_1 was before a_1 or the two events are concurrent. A statement about the sequence in real time is not possible.
- From $L(b_2) < L(c_1)$ it can **not** be concluded that $b_2 \rightarrow c_1$ holds. Since $L(c_1) \not< L(b_2)$, $c_1 \not\rightarrow b_2$ follows, so either b_2 was before c_1 or the two events are concurrent. Again, no statement can be made about the sequence in real time.
- b) $V(a_3) = (3, 1, 0) < (4, 1, 2) = V(c_2)$, i.e. $a_3 \rightarrow c_2$. a_3 must therefore have taken place in real time before c_2 .
- $V(a_1) = (1, 0, 0) \not< (0, 1, 0) = V(b_1)$ and $V(b_1) = (0, 1, 0) \not< (1, 0, 0) = V(a_1)$, i.e. $a_1 \not\rightarrow b_1$ and $b_1 \not\rightarrow a_1$, i.e. a_1 and b_1 are concurrent. Their order in real time can be arbitrary.
- $V(c_1) = (0, 0, 1) < (0, 2, 1) = V(b_2)$, i.e. $c_1 \rightarrow b_2$. c_1 must therefore have taken place in real time before b_2 .
- c) For the vector time: $V(a) < V(b) \Leftrightarrow a \rightarrow b$.
- For the Lamport time only $a \rightarrow b \Rightarrow L(a) < L(b)$ applies, but not vice versa. However, the implication $a \rightarrow b \Rightarrow L(a) < L(b)$ is equivalent to $L(a) \not< L(b) \Rightarrow a \not\rightarrow b$, so that certain conclusions can be drawn about the *happened before* relation (see subtask a).

Exercise 4: Snapshot Algorithm According to Chandy/Lamport

1. P sends m .
2. P initiates the snapshot, stores the state 101, and sends a marker to Q .
3. Q receives m , its state is 102.

4. Q receives the marker and stores its state 102. Thus Q has already received all markers, the recorded state of the message channel from P to Q is empty.
5. Q sends a marker to P
6. Q sends m back to P
7. P receives the marker
8. P stores the state of the message channel from Q to P . This is empty because P did not receive a message before the marker.

Determined state: $P : 101, Q : 102$, both channels are empty.

Note: Q could also execute step 6 immediately after step 3. In this case the state would be: $P : 101, Q : 102$, channel $Q \mapsto P$ contains message m .

Exercise 5: Bully Algorithm

- a) All higher numbered processes get two ELECTION messages. They also answer both messages with OK, so that both initiators cancel the election. If a process receives an ELECTION message but already holds an election, it ignores this (redundant) message. The higher numbered processes ignore the second ELECTION message and perform their election as usual.

- b) First, note that this is unwanted behavior if there is no direct advantage in using a higher numbered process, because a re-election is usually wasteful. However, the numbering of processes can reflect their relative utility (for example, higher numbered processes running on faster machines). In this case, the benefit of re-election can be worth the cost of it. The cost of re-election also includes repeated communication, i.e. the rounds in which ELECTION messages are sent, and the transmission of status messages from coordinator to coordinator.

To avoid re-election, a recovering process could only send a requestStatus message to the next successive lower numbered processes. This would allow them to find out if another process has already been selected and only elect themselves if they receive a negative response. The algorithm can then work as before: If the process just recovered discovers that the coordinator is failing or is receiving an ELECTION message, it sends a coordinator message to the remaining processes.

- c) With the prerequisites mentioned in the question, we cannot guarantee that a unique process can be chosen at any time.

Instead, we can accept it as sufficient to form subgroups of processes that agree with their coordinator as such and allow several such subgroups to exist at once.

For example, if a network is divided into two, we could form two largest possible subgroups, each of which would choose the process with the highest identifier among its members as the coordinator. However, in the event that the partitions are merged again, it should be ensured that the two groups become one with only one coordinator.

Hector Garcia-Molina's so-called invitation algorithm achieves this. He chooses a single coordinator within a subgroup whose members can communicate with each other. At regular intervals, the coordinator requests other members of the entire set of processes to try to merge with other groups. If another coordinator is found, a coordinator sends an invitation message to invite him to form a merged group. If a process suspects its coordinator's inaccessibility or failure, it performs an election, as in the bully algorithm.