Problem 1

This problem pertains to Cristian’s clock synchronization protocol between a Slave clock P and a Master clock Q, where \( \rho \) is the drift rate of both clocks and \( \text{min} \) is the minimum message delay between P and Q.

(1) The formula for adjusting the logical clock of a slave processor by linear amortization has the form \( C(t) = (1 + m) \cdot H(t) + N \) where \( m \) and \( N \) are constants determined by synchronization. Suppose the slave’s hardware clock is always faster than the master’s clock. What is the sign (+ve or -ve) of the constant \( m \)? Justify your answer.

Answer:

Since \( m = \frac{M-L}{\alpha} \) where \( L = \) slave’s clock time and \( M = \) master’s clock time at the beginning of amortization, \( m \) will have a **negative** sign because the slave’s clock is faster than the master’s clock (\( L > M \)).

(2) Suppose the drift rate of both clocks is actually \( \rho' \) and not \( \rho \) as specified by the clock manufacturer, where \( \rho' > \rho \). As a result, the actual precision of clock synchronization is worse than the value \( D \cdot (1 + 2\rho) - \text{min} \) that would be expected from applying Cristian’s algorithm, where \( D \) is the measured single trip delay and \( \text{min} \) is a lower bound of the delay. What is the difference between the measured and actual precision in this case? Justify your answer.

Answer:

Drift rate of both clocks: \( \rho', \quad \rho' > \rho \)
Q’s clock should be in the interval \([T + \text{min}(1-\rho'), T + 2D(1+2\rho') - \text{min}(1+\rho')]\)

The precision should be

\[ e' = D(1 + 2\rho') - \text{min} \quad \leftarrow \text{actual} \]

The measured precision is

\[ e = D(1 + 2\rho) - \text{min} \quad \leftarrow \text{measured} \]

The difference between measured and actual precision,

\[ e' - e = (D(1 + 2\rho') - \text{min}) - (D(1 + 2\rho) - \text{min}) \]

\[ = \frac{D + 2D\rho' - \text{min}}{D + 2D\rho - \text{min}} \]

\[ = 2D \rho' - 2D\rho = 2D(\rho' - \rho) \]

\[ \Delta e = \rho' - \rho \]

\[ \frac{\Delta e}{\Delta \rho} = \frac{2D}{2D} = 1 \]

\[ \frac{\Delta e}{\Delta \rho} = \frac{\Delta \text{precmin}}{(D(1 + 2\rho') - \text{min})} \]

\[ = \frac{\Delta \rho'}{(D(1 + 2\rho') - \text{min})} \cdot \Delta \rho = 2D \cdot (\rho' - \rho) \]

\[ \Delta e = \rho' - \rho \]
(3) Suppose the round-trip time measurement of a slave’s synchronization query is subject not only to drift error of the slave’s clock but also to a software error such that the resulting round-trip time measurement is always 110% of the value that the slave should have used to compute the master’s time had there been no software error. Professor Moriarty claims that in this case, the resulting synchronization precision will differ from the software-error-free value by exactly 10%. Is Moriarty’s claim correct? Justify your answer.

Answer:

The real-time round-trip delay is \(2d\). When there is no software error, \(d \leq D(1+p)\) so that \(2d \leq 2D(1+p)\) and \(e = D(1+2p) - \min\).

When there is a software error, \(2d \leq 2 \cdot 1.1 \cdot D(1+p)\).

The upper bound of the interval \([T + \min(1-p), T + (2d - \min)(1+p)]\) becomes: \(T + 2.2D(1+2p) - \min(1+p)\).

\[
\begin{align*}
(2d - \min)(1+p) \\
= (2.2D(1+p) - \min)(1+p) \\
= 2.2D(1+2p) - \min(1+p)
\end{align*}
\]

Hence, the interval becomes: \(\frac{T + 2.2D + 4.4Dp \cdot \min - \min \cdot \min}{2.2D + 4.4Dp - 2 \min}\).

And the precision becomes:

\[
\frac{1}{2}(2.2D + 4.4Dp - 2 \min) = 1.1D + 2.2Dp - \min.
\]

\[
\frac{1}{2}(1.1D(1+2p) - \min) = 1.1D(1+2p) - \min
\]

Moriarty’s claim:

\(e' = 1.1(D(1+2p) - \min) = 1.1D(1+2p) - 1.1 \min\)

Thus, Moriarty’s claim is incorrect because the \(\min\) value does not increase by 10%.

Note: \(\frac{\text{Precision}}{D} = \frac{d}{D} \cdot \frac{D(1+2p) - \min}{D} = 1+2p\).

\(\Delta \text{Precision} = (1+2p) \cdot \frac{D}{D}\)

\(\Delta \text{Precision} = \text{not proportional to Precision which is } D(1+2p) - \min\).

\(\text{choice} \text{ is a } - \min \text{ term.}
Problem 2

This problem pertains to Flaviu Cristian's synchronous atomic broadcast protocol, executed by a system of 5 processors: P0, P1, P2, P3 and P4 that are connected by 7 channels: C1 to C7. The message delay between any two processors is at most 1 time unit. The processor clocks are perfectly synchronized and the fault budget is 6. You are given the following scenario. Processor P0 sends out a hop-1 message on channel 1 at time=0 and crashes. Only P1 picks up the message on channel 1 before the message is corrupted. P1 sends out a hop-2 message on channel 2 at time=1 and crashes. Only P2 picks up the hop-2 message on channel 2 before the message is corrupted. P2 sends out a hop-3 message on channel 3 at time=2 and crashes. Only P3 picks up the hop-3 message on channel 3 before the message gets corrupted.

(1) (a) What are the channel(s) on which processor P4 will see the message? (b) What is the least number of faults that must have occurred by time=3 as deduced by processor P3? For each of these faults, name the hardware components where these faults may be?

Answer:

Part (a) 
\[ c = 3 \]
\[ h = 3 \]
\[ f = 6 \]
\[ f + 1 - h = 4 \]
\[ c + 1 = 4 \]

Since \( c < f + 1 - h \), P3 forwards a hop-4 message on channel 4. P4 will see the message on channel 4.

Part (b) 
\[ c_h = 3 \]
\[ h = 3 \]
\[ f + 1 - h = 4 \]

Since P3 receives the message on highest-numbered channel \( c_h = 3 \), \( c_h \leq f + 1 - h \), P3 deduces there are at least \( h = 3 \) faults that must have occurred.

The faults may be any 3 of the following:

- processor fault (P1 or P2)
- the out-adaptor to channel 4 (may belong to P1 or P2)
- channel 4, 2, or 1
- P3's in-adaptor to channel 4
(2) Consider further the scenario of this problem. Suppose P3 picks up a hop-3 message on channel 3 before time=2, i.e., this hop-3 message is early. Nevertheless, P3 is able to authenticate this hop-3 message and verify the correctness of its contents, i.e., the only thing wrong with this hop-3 message is its early arrival that constitutes a performance fault in the communication system. P3 decides to forward it anyway as a hop-4 message on channel 4 at time=3. Will this forwarding action result in the violation of the atomicity requirement of the atomic broadcast protocol? Justify your answer. Hint: consider the simple lazy forwarding protocol.

Answer:

T = 0
h = 3
δ = 1
ε = 0

A message \((T, s, \sigma; h)\) is considered timely if it is received by a processor at its local time \(U\), where

\[ U < T + h(\delta + \varepsilon) \]

When \(P_3\) forwards a hop-4 message \((0, P_0, \sigma; 4)\) on channel 4 at time = 3, the message will arrive at \(P_4\) at time = 4 the latest. * This forwarding action does not violate the atomicity of the atomic broadcast protocol since all correct processors receive the message.

* \(P_4\) will not reject the message because it is timely.
(3) Consider again the scenario of this problem. Instead of message corruption, the message losses are due to crash failures of the channels that carry them (i.e., channels C1, C2, C3 all crash instead of just omitting the messages). Our consultant Professor Mokiarty observes that for a channel that has crashed, each further attempt to transmit on the crashed channel will not be counted as additional individual omission faults but only as a single (crash) fault against the fault budget. Mokiarty claims that this would allow an adversary to use more (omission) faults than is allowed by the fault budget to cause Cristian’s broadcast protocol to fail, since the additional omission faults will not be counted against the fault budget. Is Professor Mokiarty correct or not? Justify your answer.

Answer: When a correct processor receives a message \( m \) on highest-numbered channel \( C_i \), \( P_i \) will not forward \( m \) to any other channel before \( C_i \), but to \( C_{i+1}, \ldots, C_{i+h} \). Therefore, a crashed channel and a channel that suffers an omission fault are not considered different. This will not exceed the fault budget specified by Cristian’s protocol, and hence Prof. Mokiarty is not correct.
Problem 3

Consider the following two options for implementing the attendance list group membership protocol that differ in when a new group request is made by the group members. In option A, every group member requests a new group as soon as it can detect a failure, depending on its position in the virtual ring. Suppose group members are labelled from 1 to n with member 1 as the leader. For $k > 1$, the $k^{th}$ member of the group will request a new group at time $=(k - 1) \cdot \delta + \epsilon$ from the beginning of the check-in period if it detects a failure, where $\delta$ is an upper bound on the message delay between adjacent nodes and all clocks are synchronized to within $\epsilon$. The leader checks for failures at time $= n \cdot \delta$. In option B, every correct group member will issue a new group request at time $= n \cdot \delta + \epsilon$ from the beginning of the check-in period if it has not received the attendance list by that time. Recall that the reconfiguration latency is the longest time between a node failure and the subsequent reconfiguration that acknowledges the node failure. Assume that an atomic broadcast takes $\Delta$ time units.

(1) What is the reconfiguration latency of option B if there are at most $x$ failures? Justify your answer.

Answer:

$T = V + \pi$, first check-in period. Let $\delta = 0$. Suppose $P_n$ fails after passing the list back to $P_1$. $P_1$ thinks all processors are correct.

$T = V + 2\pi$, second check-in period. Let $\delta > 0$. At $T = V + 2\pi + (n-1) \delta + \epsilon$, the list should reach $P_n$. $P_n$ does not pass the list back to $P_1$. $P_1$ will detect $P_n$'s failure, so we kill $P_1$.

$T = V + 3\pi$, third check-in period. Let $\delta > 0$. At $T = V + 3\pi + x \delta + \epsilon$, $P_x$ expects to receive the list from $P_1$. $P_2$ will detect $P_1$'s failure, so we kill $P_2$. Then we kill $P_3, P_4, \ldots, P_{x-1}$. At $T = V + 3\pi + (x-1) \delta + \epsilon$, $P_x$ detects the failure but waits until $T = V + 3\pi + n \delta + \epsilon$ to issue the new group request.

So the reconfiguration latency of option B is

$D = 2\pi + n \delta + \epsilon + 2\Delta$

if there are at most $x$ failures (does not depend on $x$).
(2) A disadvantage of option B is that there can be as many as \( n-1 \) nearly simultaneous new group requests if the leader crashes. Suppose that the check-in period is many times longer than the time it takes a message to traverse the virtual ring, propose a way to combine the ideas of option A and option B to remedy this disadvantage.

Answer:

\[ \pi > (n-1) \delta. \]

We can combine option A and option B by letting only the lowest numbered processor to broadcast the new group request at \( \pi < n\delta + \varepsilon \) after the check-in period.

Thus, if there are \( k \) faults, let \( P_k \) broadcast the new group request first. If \( P_k \) does not broadcast the request, \( P_{k+1} \) can broadcast the request at \( (n+1)\delta + \varepsilon \) after the check-in period, and so on. \( n\delta + \varepsilon + \Delta \)

\( P_k \) can tell the other processors \( P_{k+1}, \ldots \) that it wants to broadcast the request, by passing a message. Then, if at \( n\delta + \varepsilon + \Delta \) the other processors do not see a broadcast, let \( P_{k+1} \) tell the others it wants to broadcast the request, and so on.
(3) In the attendance list protocol, if there is no processor failure, \( n \) processor-to-processor messages (datagrams) are needed to execute the protocol for each check-in period of length \( \pi \). Suppose a check-in period starts at time \( t_1 \), a processor fails during the period, and the first detection of some failure occurs subsequently at time \( t_2 \), possibly in a later check-in period. Let \( x \) be the total number of datagrams sent by all the processors in the interval \([t_1, t_2]\). How big can \( x \) be in the worst case if up to \( k \) processors may fail? Do not count the atomic broadcast messages for reconfiguration. Explain your calculation clearly.

Answer:

\( t_1 = \text{check-in period starts} \) (consider it 1st check-in period)

\( t_2 = \text{detection on some failure} \)

\( x = \text{total # of datagrams sent in } [t_1, t_2] \)

\( k \) processors may fail.

\[ \text{Suppose } \delta = 0 \text{ in the first check-in period. At time } t_1, \text{ a processor fails. Let it be } P_n, \text{ the last processor, failing after passing the list to } P_1. \]

In this period, there are \( n \) datagrams sent.

At \( t_1 + \pi \), the next check-in period starts. \( P_1 \) passes the list up to \( P_n \). In this period, there are \( n-1 \) datagrams sent, because \( P_n \) does not pass the list to \( P_1 \). Before \( P_1 \) detects the failure, kill \( P_1 \).

At \( t_1 + 2\pi \), the 3rd check-in period starts. \( P_2 \) will detect \( P_1 \)'s failure, so kill \( P_2 \). Kill \( P_3 \), ..., \( P_{k-1} \). \( P_k \) will detect the failure. In this period, there is no datagram sent.

Therefore \( x = n + (n-1) = 2n - 1 \).