Distributed Systems: Concepts and Design

Chapter 14 Exercise Solutions

14.1 Why is computer clock synchronization necessary? Describe the design requirements for a system to synchronize the clocks in a distributed system.

14.1 Ans.

See Section 14.1 for the necessity for clock synchronization.

Major design requirements:

i) there should be a limit on deviation between clocks or between any clock and UTC;

ii) clocks should only ever advance;

iii) only authorized principals may reset clocks (See Section 11.6.2 on Kerberos)

In practice (i) cannot be achieved unless only benign failures are assumed to occur and the system is synchronous.

14.2 A clock is reading 10:27:54.0 (hr:min:sec) when it is discovered to be 4 seconds fast. Explain why it is undesirable to set it back to the right time at that point and show (numerically) how it should be adjusted so as to be correct after 8 seconds has elapsed.

14.2 Ans.

Some applications use the current clock value to stamp events, on the assumption that clocks always advance.

We use E to refer to the 'errant' clock that reads 10:27:54.0 when the real time is 10:27:50. We assume that H advances at a perfect rate, to a first approximation, over the next 8 seconds. We adjust our software clock S to tick at rate chosen so that it will be correct after 8 seconds, as follows:

$S = c(E - Tskew) + Tskew$, where $Tskew$ = 10:27:54 and c is to be found.

But $S = Tskew+4$ (the correct time) when $E = Tskew+8$, so:

$Tskew+ 4 = c(Tskew + 8 - Tskew) + Tskew$, and c is 0.5. Finally:

$S = 0.5(E - Tskew) + Tskew$ (when $Tskew$ ð E ð $Tskew+8$).

14.3 A scheme for implementing at-most-once reliable message delivery uses synchronized clocks to reject duplicate messages. Processes place their local clock value (a 'timestamp') in the messages they send. Each receiver keeps a table giving, for each sending process, the largest message timestamp it has seen. Assume that clocks are synchronized to within 100 ms, and that messages can arrive at most 50 ms after transmission.

(i) When may a process ignore a message bearing a timestamp T, if it has recorded the last message received from that process as having timestamp T' ?

(ii) When may a receiver remove a timestamp 175,000 (ms) from its table? (Hint: use the

receiver's local clock value.)

(iii) Should the clocks be internally synchronized or externally synchronized?

14.3 Ans.

i) If T ð T' then the message must be a repeat.

ii) The earliest message timestamp that could still arrive when the receiver's clock is r is r - 100 - 50. If this is to be at least 175,000 (so that we cannot mistakenly receive a duplicate), we need r -150 = 175,000, i.e. r = 175,150.

iii) Internal synchronisation will suffice, since only time *differences* are relevant.

14.4 A client attempts to synchronize with a time server. It records the round-trip times and timestamps returned by the server in the table below.

Which of these times should it use to set its clock? To what time should it set it? Estimate the accuracy of the setting with respect to the server's clock. If it is known that the time between sending and receiving a message in the system concerned is at least 8 ms, do your answers change?

Round-trip (ms)	*Time (hr:min:sec)*
22	*10:54:23.674*
25	*10:54:25.450*
20	*10:54:28.342*

14.4 Ans.

The client should choose the minimum round-trip time of 20 ms = 0.02 s. It then estimates the current time to be 10:54:28.342 + 0.02/2 = 10:54:28.352. The accuracy is ± 10 ms.

If the minimum message transfer time is known to be 8 ms, then the setting remains the same but the accuracy improves to ± 2 ms.

14.5 In the system of Exercise 14.4 it is required to synchronize a file server's clock to within ±1 millisecond. Discuss this in relation to Cristian's algorithm.

14.5 Ans.

To synchronize a clock within ± 1 ms it is necessary to obtain a round-trip time of no more than 18 ms, given the minimum message transmission time of 8 ms. In principle it is of course possible to obtain such a round-trip time, but it may be improbable that such a time could be found. The file server risks failing to synchronize over a long period, when it could synchronize with a lower accuracy.

14.6 What reconfigurations would you expect to occur in the NTP synchronization subnet?

14.6 Ans.

A server may fail or become unreachable. Servers that synchronize with it will then attempt to synchronize to a different server. As a result, they may move to a different stratum. For example, a stratum 2 peer (server) loses its connection to a stratum 1 peer, and must thenceforth use a stratum 2 peer that has retained its connection to a stratum 1 peer. It becomes a stratum 3 peer.

Also, if a primary server's UTC source fails, then it becomes a secondary server.

14.7 An NTP server B receives server A's message at 16:34:23.480 bearing a timestamp 16:34:13.430 and replies to it. A receives the message at 16:34:15.725, bearing B's timestamp 16:34:25.7. Estimate the offset between B and A and the accuracy of the estimate.

14.7 Ans.

Let $a = T_{i-2} - T_{i-3}$ = 23.48 - 13.43 = 10.05; $b = T_{i-1} - T_i$ = 25.7- 15.725 =9.975.

Then the estimated offset o_i = (a+b)/2 = 10.013s, with estimated accuracy = $\pm d_i/2$ = ± (a-b)/2 = 0.038s (answers expressed to the nearest millisecond).

14.8 Discuss the factors to be taken into account when deciding to which NTP server a client should synchronize its clock.

14.8 Ans.

The main factors to take into account are the intrinsic reliability of the server as a source of time values, and the quality of the time information as it arrives at the destination. Sanity checks are needed, in case servers have bugs or are operated maliciously and emit spurious time values. Assuming that servers emit the best time values known to them, servers with lower stratum numbers are closest to UTC, and therefore liable to be the most accurate. On the other hand, a large network distance from a source can introduce large variations in network delays. The choice involves a trade-off between these two factors, and servers may synchronize with several other servers (peers) to seek the highest quality data.

14.9 Discuss how it is possible to compensate for clock drift between synchronization points by observing the drift rate over time. Discuss any limitations to your method.

14.9 Ans.

If we know that the drift rate is constant, then we need only measure it between synchronization points with an accurate source and compensate for it. For example, if the clock loses a second every hour, then we can add a second every hour, in smooth increments, to the value returned to the user. The difficulty is that the clock's drift rate is liable to be variable – for example, it may be a function of temperature. Therefore we need an adaptive adjustment method, which guesses the drift rate, based on past behaviour, but which compensates when the drift rate is discovered to have changed by the next synchronisation point.

14.10 By considering a chain of zero or more messages connecting events e and e' and using induction, show that $e \rightarrow e' \Rightarrow L(e) < L(e')$.

14.10 Ans.

If e and e' are successive events occurring at the same process, or if there is a message m such that $e = send(m)$ and $e' = rcv(m)$, then the result is immediate from LC1 and LC2. Assume that the result to be proved is true for all pairs of events connected in a sequence of events (in which either *HB1* or *HB2* applies between each neighbouring pair) of length N or less ($N \geq 2$). Now assume that e and e' are connected in a series of events $e_1, e_2, e_3, .., e_{N+1}$ occurring at one or more processes such that $e = e_1$ and $e' = e_{N+1}$. Then $e \rightarrow e_N$ and so $C(e) < C(e_N)$ by the induction hypothesis. But by LC1 and LC2, $C(e_N) < C(e')$. Therefore $C(e) < C(e')$.

14.11 Show that $V_j[i] \leq V_i[i]$ (NB erratum in this exercise in first printing)

14.11 Ans.

Rule VC2 (p. 609) tells us that p_i is the 'source' of increments to $V_i[i]$, which it makes just before it sends each message; and that p_j increments $V_j[i]$ only as it receives messages containing timestamps with larger entries for p_i. The relationship $V_j[i] \leq V_i[i]$ follows immediately.

14.12 In a similar fashion to Exercise 14.10, show that $e \rightarrow e' \Rightarrow V(e) < V(e')$.

14.12 Ans.

If e and e' are successive events occurring at the same process, or if there is a message m such that $e = send(m)$ and $e' = rcv(m)$, then the result follows from VC2–VC4. In the latter case the sender includes its timestamp value and the recipient increases its own vector clock entry; all of its other entries remain at least as great as those in the sender's timestamp.

Assume that the result to be proved is true for all pairs of events connected in a sequence of events (in which either *HB1* or *HB2* applies between each neighbouring pair) of length N or less ($N \geq 2$). Now assume that e and e' are connected in a series of events $e_1, e_2, e_3, .., e_{N+1}$ occurring at one or more processes such that $e = e_1$ and $e' = e_{N+1}$. Then $e \rightarrow e_N$ and so $V(e) < V(e_N)$ by the induction hypothesis. But by VC2–VC4, $V(e_N) < V(e')$. Therefore $V(e) < V(e')$.

14.13 Using the result of Exercise 14.11, show that if events e and e' are concurrent then neither $V(e) \leq V(e')$ nor $V(e') \leq V(e)$. Hence show that if $V(e) < V(e')$ then $e \rightarrow e'$.

14.13 Ans.

Let *e* and e' be concurrent and let *e* occur at p_i and e' at p_j. Because the events are concurrent (not related by happened-before) we know that no message sent from p_i at or after event *e* has propagated its timestamp to p_j by the time e' occurs at p_j, and *vice versa*. By the reasoning for Exercise 10.11, it follows that $V_j[i] < V_j[i]$ and $V_i[j] < V_j[j]$ (strict inequalities) and therefore that neither $V(e) \leq V(e')$ nor $V(e') \leq V(e)$.

Therefore if $V(e) < V(e')$ the two events are not concurrent – they must be related by happened-before. Of the two possibilities, it obviously must be that $e \rightarrow e'$.

14.14 Two processes *P* and *Q* are connected in a ring using two channels, and they constantly rotate a message *m*. At any one time, there is only one copy of *m* in the system. Each process's state consists of the number of times it has received *m*, and *P* sends *m* first. At a certain point, *P* has the message and its state is 101. Immediately after sending *m*, *P* initiates the snapshot algorithm. Explain the operation of the algorithm in this case, giving the possible global state(s) reported by it.

14.14 Ans.

P sends msg m

P records state (101)

P sends marker (see initiation of algorithm described on p. 406)

Q receives m, making its state 102

Q receives the marker and by marker-receiving rule, records its state (102) and the state of the channel from P to Q as {}

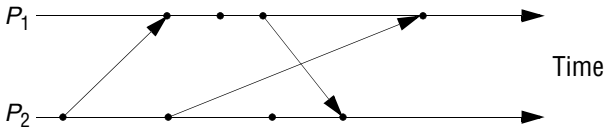
Q sends marker (marker-sending rule)

(Q sends m again at some point later)

P receives marker

P records the state of the channel from Q to P as set of messages received since it saved its state = {} (marker-receiving rule).

14.15

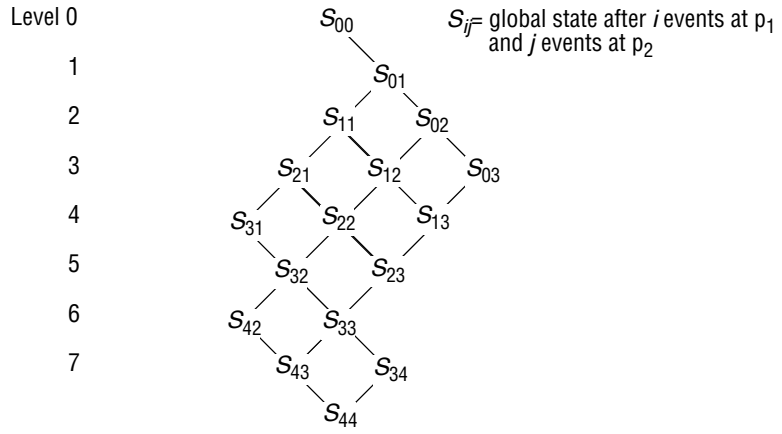


The figure above shows events occurring for each of two processes, p_1 and p_2. Arrows between processes denote message transmission.

Draw and label the lattice of consistent states (p_1 state, p_2 state), beginning with the initial state (0,0).

14.15 Ans.

The lattice of global states for the execution of Figure of exercise 11.15



14.16 Jones is running a collection of processes $p_1, p_2, \ldots, p_N$. Each process p_i contains a variable v_i. She wishes to determine whether all the variables $v_1, v_2, \ldots, v_N$ were ever equal in the course of the execution.

(i) Jones' processes run in a synchronous system. She uses a monitor process to determine whether the variables were ever equal. When should the application processes communicate with the monitor process, and what should their messages contain?

(ii) Explain the statement *possibly* ($v_1 = v_2 = \ldots = v_N$). How can Jones determine whether this statement is true of her execution?

14.16 Ans.

(i) communicate new value when local variable v_i changes;

with this value send: current time of day *C(e)* and vector timestamp *V(e)* of the event of the change, *e*.

(ii) *possibly* (...): there is a consistent, potentially simultaneous global state in which the given predicate is true.

Monitor process takes potentially simultaneous events which correspond to a consistent state, and checks predicate $v_1 = v_2 = \ldots = v_N$.

Simultaneous: estimate simultaneity using bound on clock synchronization and upper limit on message propagation time, comparing values of *C* (see p. 415).

Consistent state: check vector timestamps of all pairs of potentially simultaneous events e_i, e_j: check $V(e_i)[i] \geq V(e_j)[i]$.