

CSSE4004-Lecture 5

Synchronisation

1

Learning Objectives

After this week you should be able to describe a number of classic algorithms used for process synchronisation in DS, including:

- Synchronisation of physical clocks
- Ordering of events (logical clocks)
- Providing mutual exclusion
- Electing coordinator processes

2

Outline

- Clock Synchronisation
- Logical Clocks
- Mutual Exclusion
- Election Algorithms

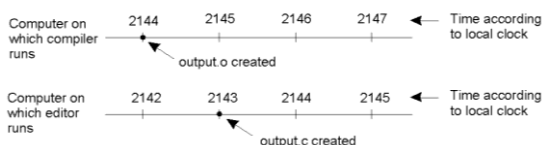
3

Clock Synchronisation

- The Problem with clocks in DS
- Physical Clocks
- Clock Synchronisation Algorithms:
 - Network Time Protocol (NTP)
 - The Berkeley Algorithm
 - Clock Synchronisation in Wireless Networks

4

Clock Synchronisation - The Problem with Clocks in DS



When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time

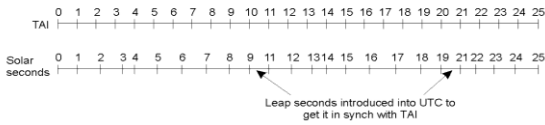
5

Clock Synchronisation - Physical Clocks (1)

- Sometimes we need the exact time in DS
- Solution: Universal Coordinated Time (UTC):
 - Based on the number of transitions per second of the Cesium 133 atom
 - At present, the real time is taken as the average of approx. 50 Cesium clocks around the world (International Atomic Time – TAI)
 - Introduces a leap second from time to time to compensate that days are getting longer
- UTC is broadcast through short wave radio and by satellite. Satellites can give an accuracy of about ± 0.5 ms.

6

Clock Synchronisation - Physical Clocks (2)



TAI seconds are of constant length, unlike solar seconds. Leap seconds are introduced when necessary to keep in phase with the sun

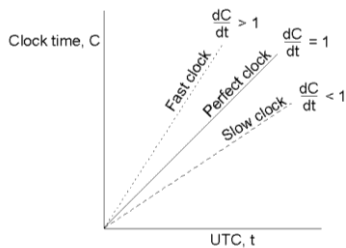
7

Clock Synchronisation Algorithms

- Suppose we have a distributed system with a UTC receiver somewhere in it we still have to distribute its time to each machine
- Assumptions:
 - Every machine has a timer that generates an interrupt H times per second.
 - There is a clock in machine p that **ticks** on each timer interrupt. Denote the value of that clock by $C_p(t)$, where t is UTC time.
 - Ideally, we have that for each machine p , $C_p(t) = t$ or, in other words, $dC/dt = 1$

8

Clock Synchronisation Algorithms



The relation between clock time and UTC when clocks tick at different rates (clocks may drift from UTC)

9

Clock Synchronisation Algorithms - Network Time Protocol (1)

- The Network Time Protocol (NTP) was developed to synchronise clocks across DS
- NTP achieves accuracy of between 1 and 50 ms
- NTP servers are divided into strata reflecting the accuracy of their clocks
- The most accurate servers are referred to as stratum-1 (and typically have direct access to a reference clock)

10

Clock Synchronisation Algorithms - Network Time Protocol (2)

- NTP operates pair-wise between servers
- For example:
 - Servers A and B probe each other for the current time
 - Both calculate:
 - relative clock offset (i.e. differences in reported time)
 - delay (message propagation delay)
 - Assume B is a stratum-2 server, and A a stratum-10
 - Lower stratum servers are assumed to be more accurate => A will try to synchronise with B's clock
 - In general, synchronising with stratum- k server makes you stratum- $k+1$
 - So, A will become stratum-3

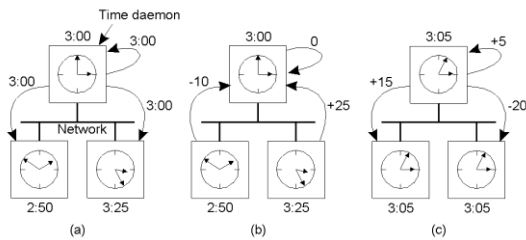
11

Clock Synchronisation Algorithms - The Berkeley Algorithm (1)

- Berkeley algorithm uses averaging approach to correct clocks (so doesn't need WWV receiver)
- Time calculations performed by time daemon
- Time daemon's clock must be manually set (periodically)
- New time is calculated as follows:
 - daemon announces its time to each node on the network
 - nodes report how far ahead/behind their clocks are
 - daemon calculates new time based on average of reported values
 - daemon tells each node how to adjust its clock

12

Clock Synchronisation Algorithms - The Berkeley Algorithm (2)



- a) The time daemon asks all the other machines for their clock values
- b) The machines answer
- c) The time daemon calculates average time and tells everyone how to adjust their clock

13

Clock Synchronisation Algorithms - Wireless Networks

- Clock synchronisation in wireless networks is problematic because:
 - Nodes cannot always contact one another
 - Nodes are resource-constrained
- Reference Broadcast Synchronization (RBS) designed for wireless sensor networks
 - offers network internal synchronisation (not necessarily to UTC time)
 - sender broadcasts reference message with timestamp
 - receivers record difference between reference message timestamp and their own clock
 - receivers store time offset (calculated using simple linear regression algorithm) for each sender

14

Logical Clocks

- The order in which events occur in the DS is often more important than the time that they occurred
- The order of events can be established using logical clocks
- Examples of approaches that use logical clocks include:
 - Lamport's algorithm
 - Vector clocks

15

Logical Clocks - Lamport's algorithm (1)

- Events in the DS can be ordered using Lamport's **happened-before** relation
- The **happened-before** relation on the set of events in a DS is the smallest relation satisfying:
 - if a and b are events in the same process, and a occurs before b , then $a \rightarrow b$ is true
 - if a is the event of a message being sent by one process, and b is the receipt of that message in another process, then $a \rightarrow b$ is also true
- This relation is transitive:
 - $a \rightarrow b$ and $b \rightarrow c$, then $a \rightarrow c$

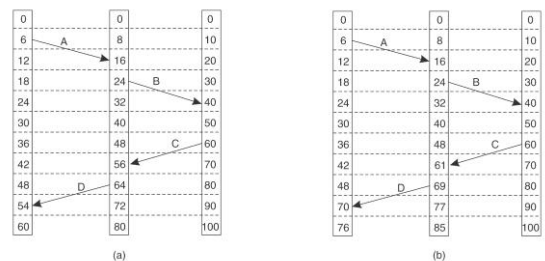
16

Logical Clocks - Lamport's algorithm (2)

- In Lamport's logical clock algorithm:
 - each event a has an associated time $C(a)$ based on the local clock
 - between any two events, the clock must tick at least once (i.e. no two events ever occur at the same time)
 - messages carry their sending time according to the sender's clock e.g., $C(b)$
 - when a message is received, its time is compared against the local clock. If the local clock is less than $C(b)$, it is set to $C(b) + 1$

17

Logical Clocks - Lamport's algorithm (3)



- a) Three processes each with their own logical clock. The clocks run at different rates.
- b) Lamport's algorithm corrects the clocks.

18

Logical Clocks

- Lamport's algorithm (4)

- Events occurring in processes that do not interact (even indirectly through third parties) are said to be concurrent
- Nothing can be said about the order of concurrent events
- For example:
 - Events x and y occur in two different process (that do not interact at all)
 - The happened-before relation cannot be applied as x and y are concurrent
 - This means that $x \rightarrow y$ is not true, and $y \rightarrow x$ is not true

19

Logical Clocks

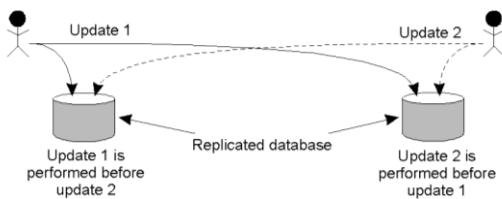
- Example: totally ordered multicast (1)

- A bank replicates its customer account database in two locations (e.g., New York and San Francisco)
- Queries are forwarded to the nearest replica
- Update operations on the customer account database need to be executed on both replicas in the same order
- A totally ordered multicast mechanism is needed to achieve this

20

Logical Clocks

- Example: totally ordered multicast (2)



- Two updates are multicast to a replicated database
- If messages are not received in the same order the database is left inconsistent

21

Logical Clocks

- Example: totally ordered multicast (3)

- Totally ordered multicast can be implemented at each database replica by:
 - ordering all update messages received according to logical clocks
 - acknowledging update messages upon receipt
 - only performing updates acknowledged by all database replicas

22

Logical Clocks

- Vector Clocks (1)

- Lamport's algorithm ensures that if the happened-before relation exists between events a and b , then $C(a) < C(b)$
- But if $C(c) < C(d)$, Lamport's algorithm doesn't guarantee that c happened before d
- The problem is that Lamport's algorithm does not capture **causality**
- Causality can be captured using vector clocks

23

Logical Clocks

- Vector Clocks (2)

- The vector clock for an event a is signified by $VC(a)$
- For two events, a and b , if $VC(a) < VC(b)$, then event a causally precedes event b
- To construct a vector clock, each process P_i maintains a vector VC_i
- $VC_i[i]$ is the value of the logical clock at P_i
- If $VC_i[j] = k$, P_i knows that at least k events have occurred at P_j .

24

Logical Clocks - Vector Clocks (3)

- For every event that occurs at P_i the vector value $VC_i[i]$ is incremented by one
- A process' vector is piggybacked onto all messages sent by that process
- Every time a message is received, the recipient process updates its own vector (VC_r)
- If we assume the message vector is VC_m , the update is performed by setting $VC_r[k] = \text{MAX}(VC_r[k], VC_m[k])$ for each k

25

Logical Clocks - Vector Clocks (4)

- The issue of total ordering and causal ordering of messages by the communication system is controversial
- Total or causal ordering can also be provided in the application (**end-to-end** argument)

26

Mutual Exclusion

- Processes in a distributed systems may want exclusive access to a shared resource
- A mutual exclusion mechanism is required to prevent corruption (or inconsistent updates) of that resource
- How to achieve mutual exclusion in DS?
- Typical solutions:
 - Via a centralised server
 - Distributed, with no topology imposed
 - Distributed, using a ring topology

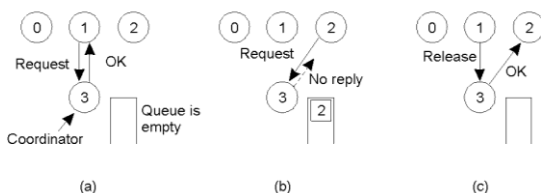
27

Mutual Exclusion - A Centralised Algorithm (1)

- Coordinator process enforces mutual exclusion over resource
- Processes must ask coordinator for permission to access resource
- Benefits of centralised approach:
 - easy to implement
 - low message overhead
 - fair (access requests are processed in order)
- Drawbacks:
 - coordinator is a single point of failure
 - coordinator can be performance bottleneck

28

Mutual Exclusion - A Centralised Algorithm (2)



- Process 1 asks the coordinator for permission to access resource. Permission is granted.
- Process 2 then asks permission to access the same resource. The coordinator does not reply.
- When process 1 releases resource, it tells the coordinator, which then replies to 2

29

Mutual Exclusion - A Distributed Algorithm (1)

- A process wanting to access a shared resource sends a message to all other processes. The message contains:
 - the requested resource's name
 - the requesting process' process id
 - the requesting process' logical time
- Recipients of the message follow one of three behaviours. If the recipient:
 - doesn't want the resource, it sends back OK
 - currently holds the resource, it ignores the message
 - wants to use the resource, then it checks if the logical time in the message is less than its own logical time. If so, it sends back OK. If not, it queues the message and sends back nothing.

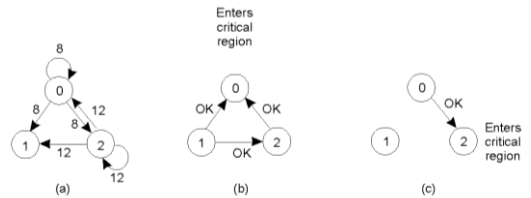
30

Mutual Exclusion - A Distributed Algorithm (2)

- To access the shared resource, a process must receive an OK from all other processes
- When a process is finished with a resource it:
 - sends OK messages to processes in its queue
 - deletes its queue
- Benefits of approach:
 - solution is fair
 - does not need a single coordinator
- Drawbacks:
 - all processes are involved in all decisions (one slow process slows down others)
 - large number of messages required
 - single point of failure replaced with n points of failure
 - processes must have accurate group membership list

31

Mutual Exclusion - A Distributed Algorithm (3)



- Two processes want to access the resource at the same time
- Process 0 has the lowest timestamp, therefore it wins
- When process 0 is done, it sends an OK, so 2 can now access the resource

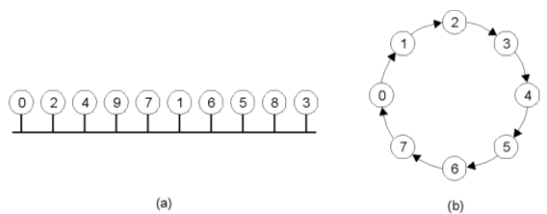
32

Mutual Exclusion - A Token Ring Algorithm (1)

- Uses a logical ring to order processes
- Processes can only access the shared resource while in possession of a token
- The token is passed on to the next node in the ring if:
 - the current token holder does not want to access the shared resource
 - is the token holder is finished accessing the shared resource
- The token circulates around the ring in one direction
- Benefits of this approach:
 - algorithm is simple
- Drawbacks:
 - token must be regenerated if lost
 - crashed processes can stop circulation of token
 - potentially have to wait for every other process to use token before it is your turn

33

Mutual Exclusion - A Token Ring Algorithm (2)



- An unordered group of processes on a network
- A logical ring constructed in software

Only the process with the token may access the shared resource

34

Mutual Exclusion - Comparison of Algorithms

Algorithm	Messages per entry/exit	Delay before entry (in message times)	Problems
Centralised	3	2	Coordinator crash
Distributed	$2(n-1)$	$2(n-1)$	Crash of any process
Token ring	1 to ∞	0 to $n-1$	Lost token, process crash

A comparison of three mutual exclusion algorithms

35

Election Algorithms

- Many distributed algorithms require that one of the processes acts as a coordinator
- An election is used to dynamically select coordinator
- Election algorithms needed so that at end of election all processes agree on coordinator
- Common election algorithms are:
 - the Bully Algorithm
 - a Ring Algorithm
- Different algorithms needed for:
 - Wireless network environments
 - Large-scale distributed systems

36

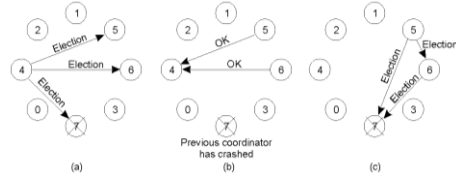
Election Algorithms - The Bully Algorithm (1)

- When process P notices coordinator is non-responsive, it initiates an election
- Election conducted as follows:
 - P sends an ELECTION message to all processes with higher process numbers.
 - If no one responds, P wins the election and becomes coordinator.
 - If one of the higher-ups answers, it takes over. P's job is done.

37

Election Algorithms - The Bully Algorithm (2)

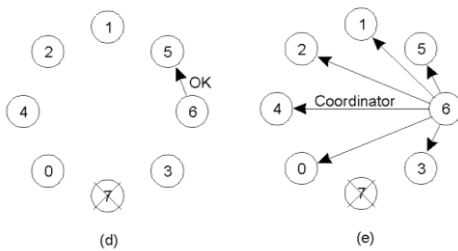
In following example assume Process 7 (previous coordinator) has crashed.



- Process 4 holds an election
- Process 5 and 6 respond, telling 4 to stop
- Now 5 and 6 each hold an election

38

Election Algorithms - The Bully Algorithm (3)



- Process 6 tells 5 to stop
- Process 6 wins and tells everyone

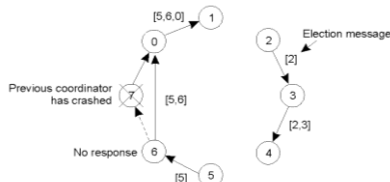
39

Election Algorithms - A Ring Algorithm (1)

- Assumes processes are logically ordered (each node knows its successor in ring)
- When process P notices coordinator is non-responsive, it initiates election
- Election conducted as follows:
 - P sends ELECTION message (containing P's process num.) to successor
 - Recipients add own process num. to message and pass to their successor
 - Message gets back to P, who changes message type to COORDINATOR
 - COORDINATOR message circles ring again
 - Process with highest process num. in COORDINATOR message becomes coordinator

40

Election Algorithms - A Ring Algorithm (2)



- For example:
 - process 7 crashes
 - 5 notices 7 crashed, so sends ELECTION to 6 (its successor)
 - 2 also notices 7 crashed, so it also sends ELECTION (to 3, its successor)
- If the successor is down, it has to be skipped.
- At each step, the sender adds its own process num.
- Coordinator is the process with the highest number (i.e. 6).

41

Election Algorithms - for Wireless Environments (1)

- Previously described election algorithms need:
 - reliable message passing
 - stable network topology
- These aren't always present in wireless environments
- The following election protocol for wireless environments attempts to overcome these problems

42

Election Algorithms - for Wireless Environments (2)

- Node that calls election becomes *source node*
- Source node sends ELECTION message to all neighbours
- The first time a node receives ELECTION message it:
 - marks sender as parent
 - forwards ELECTION message to all its neighbours
- Subsequent ELECTION messages (not from parent) are acknowledged only
- Nodes wait a set time for acknowledgements from neighbours, before sending own acknowledgement to parent

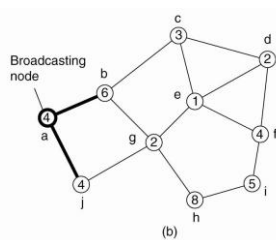
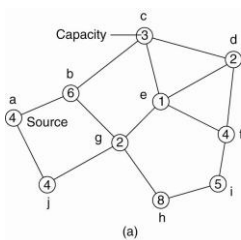
43

Election Algorithms - for Wireless Environments (3)

- Acknowledgements contain information on the resource capacities of the node's best neighbour (e.g., battery power)
- The source node uses the acknowledgement information to select the coordinator

44

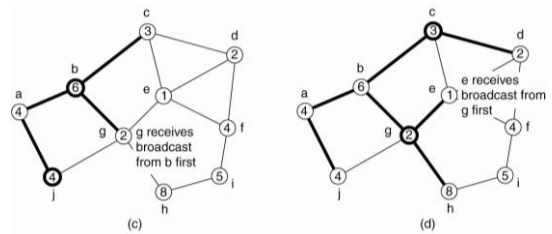
Election Algorithms - for Wireless Environments (4)



Election algorithm in a wireless environment with node a as the source. (a) is initial state. (b) – (e) tree building phase

45

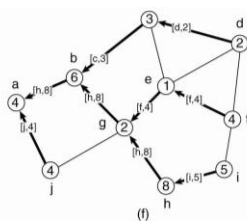
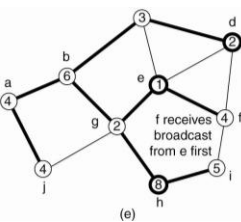
Election Algorithms - for Wireless Environments (5)



(b) – (e) The tree building phase.

46

Election Algorithms - for Wireless Environments (6)



(e) The build-tree phase.
(f) Reporting of best node to source.

47

Election Algorithms - for Large-Scale Systems (1)

- Previous algorithms select one node only
- Large-scale systems may require many local coordinators (e.g., for peer-to-peer networks superpeers keep index of content on neighbours to speed searches)
- Superpeers should:
 - offer regular nodes low-latency access
 - be evenly distributed throughout network
 - exist in predefined proportion to regular nodes
 - serve no more than a set number of regular nodes

48

Election Algorithms - for Large-Scale Systems (2)

- Two approaches for selecting superpeers:
 - using a Distributed Hash Table (DHT) identifier
 - using repulsion forces
- Distributed Hash Table identifier:
 - fraction of DHT identifier space is reserved for superpeers
 - lookup keys and-ed with superpeer bitmask to find superpeer responsible for file

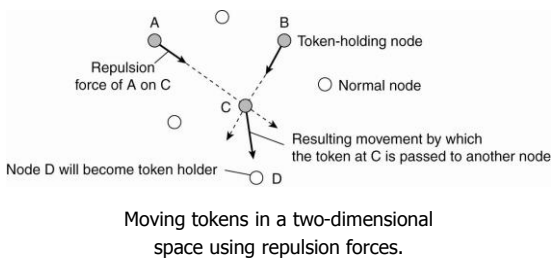
49

Election Algorithms - for Large-Scale Systems (3)

- Repulsion forces approach:
 - n tokens spread across peer-to-peer overlay
 - each node holding a token learns about other token-holders
 - each token is "repulsed" by nearby tokens (token holder sends token to another peer if too many tokens nearby)
 - tokens passed around network until tokens spread evenly across network
 - token must be held by a node for a set time period before node can become superpeer

50

Election Algorithms - for Large-Scale Systems (4)



51

Summary

- In DS there is no globally shared clock
- Processes often need to work together in DS, which requires a common notion of time for those processes
- Many time synchronisation algorithms exist for DS, which are related to:
 - Physical clock synchronisation
 - Logical clocks
 - Mutual exclusion
 - Election algorithms
- Reading: Chapter 6

52