Why do we need clock synchronization? Explain NTP and the Berkeley algorithm. What is the challenge when synchronizing physical clocks?

Book p. 231-242; Slides p. 3, 9-10, 14

Why (slides p. 3)

When each machine has its own clock, an event that occurred after another event may nevertheless be assigned an earlier time. So we need synchronization for assigning a natural order in which operations are processed in distributed systems. It is a special topic in distributed systems, because in a centralized system each process asks the same clock, so there cannot be a negative gap between the times received from a system call to the kernel. But in distributed systems there are several clocks which have to be synchronized to get the same stable effect.

NTP (slides p. 9-10)

...is a protocol for synchronization with an external time server which can provide an accurate time. The problem are the message delays over the network and the solution is to find a good estimation of these delays.

In this case, A will send a request to B, timestamped with value T1 B, in turn, record the time of receipt T2 (taken from its own local clock), and returns a timestamped with value T3, and piggybacking the previously recorded T2. Finally, A records the time of the response's arrival, T4. If we assume that the

propagation delays from B to A and from A to B are the same, means T1-T2 = T3-T4 = dTreq = dTres, then A can estimate its offset relative to B as

$$\theta = T_3 - \frac{(T_2 - T_1) + (T_4 - T_3)}{2} = \frac{(T_2 - T_1) + (T_3 - T_4)}{2}$$

After this estimation A **adjusts its local clock (means it slows it down or accelerates it)** and so both are synchronized. If A's clock ran too fast, so it should be adjusted **backwards, no adjustment** is done, because setting a clock back will cause serious problems. The delay estimation process is **repeated eight times** and the best (shortest) delay is taken for adjustment. If the the delay is above a specific threshold, the time source is not reliable.

Problem:
One of the main problems is that the delay in one direction can be different than the other one because of e.g. security advisories. If this happens the algorithm interprets the delay in a wrong way.

Berkeley (slides p. 14)

The main differences of the Berkeley algorithm is, that it is a “push”-algorithm, i.e. the time server (time daemon) asks the other machines instead of the other way round.

a) The time daemon sends its local time to all the other machines (including itself).
b) The machines answer with the time of their local clock.
c) The time daemon calculates the mean difference time and sends adjustment for the single machines (again including itself) to them, so they can adjust their local times.

Problems (slides p. 10)

- The clock is slowly adjusted forwards but never backwards, because timestamps must always go straight forward.
- The network delays can be different (against NTP algorithm assumption)

- There are different levels of accuracy in WAN (ms), LAN (µs) or with special hardware (ns).
- There can be security threats because of security vulnerabilities.

Question 2 Slide 6: p. 1-24

Why do we use logical clocks and what are the differences to physical clocks? What is the "happened-before" relation and how do "Lamport-Timestamps" work?

A logical clock is a mechanism for capturing chronological and causal relationships in a distributed system.
Logical clock algorithms of note are:

- Lamport timestamps, which are monotonically increasing software counters.
- Vector clocks, that allow for total ordering of events in a distributed system.

For many applications:

- it is sufficient that all machines agree on the same time.
- it is not essential that this time also agree with the real time

E.g. make example - it is adequate that all machines agree that it is 10:00 even if it is really 10:02.
Meaning: it is the internal consistency of the clocks that matters, not whether they are particularly close to the real time.
For these algorithms it is conventional to speak of the clocks as **logical clocks.**

Counterexample
when the additional constraint is present that the clocks

- must not only be the same,
- but also must not deviate from the real time by more than a certain amount,

the clocks are called **physical clocks.**

Happened-before

- relation between the result of two events, such that if one event should happen before another event, the result must reflect that. Even if those events are in reality executed out of order. This involves ordering events based on the potential causal relationship of pairs of events in a concurrent system, especially asynchronous distributed systems In Java specifically, a happens-before relationship is a guarantee that memory written to by statement A is visible to statement B, that is, that statement A completes its write before statement B starts its real.

The happened-before relation is formally defined as the least strict partial order on events such that:
If events a and b occur on the same process, $a \rightarrow b$ if the occurrence of event a preceded the occurrence of event b.
If event a is the sending of a message and event b is the reception of the message sent in event a, $a \rightarrow b$.
the happened-before relation is transitive, irreflexive and antisymmetric,

Lamport-timestamp

simple mechanism by which the happened-before ordering can be captured numerically.
It follows some simple rules:

- A process increments its counter before each event in that process;
- When a process sends a message, it includes its counter value with the message;
- On receiving a message, the receiver process sets its counter to be greater than the maximum of its own value and the received value before it considers the message received.

Exercise 03: Lamport timespamps (slide 06, p. 32-39)

Question: What are the disadvantages of Lamport timestamps and how can they be improved with vector timestamps?

Properties of Lamport timestamps:

- Two sequenced events a and b in the same process: L(a) < L(b)
- Reception of a message always has a higher counter than sending of messages
- The relation is transitively coped

Disadvantages of Lamport (slide p. 32):

- No causal dependencies; Just because L(a) < L(b), doesn't mean, that a happened before b!
  → L(a) < L(b) ≠> a →b
  It is possible that the times derive from two or more different, independent processes.
  Rather: L(a) < L(b) ≠> (a →b) or (a||b)
- Concurrent events can't be identified

All in all → too restrictive! → better use Vector timestamps (see next chapter)!

Vector Timespamps (slide p. 33 -39, see picture below)

- Every process has a timestamp in the form of (1,0,0) → so every process has a view on itself and the other processes (vector with counter of own und other processes)
- The following can be assumed: VC(a) is assigned to event a, VC(b) is assigned to event b. If VC(a) < VC(b), so a is executed causally before b.
- A vector is smaller than another vector, if <u>all</u> members are smaller or equal and 1 member is smaller than the specified members
- Every one of the n processes has a vector of length n (initialized with Null-Vector). After every event in process i, the element i is incremented. Afterwards the specified vector is attached to the sent message. When process j receives the message, the delivery to the application is <u>delayed</u> until the following condition is accomplished:
  - m[i] = Vj[i] + 1 // m is the message, m[i] describes the counter of process i in the vector of the messagej
  - m[k] <= Vj[k] for all k, except of i
- Potential causal dependent messages can be found by comparing element to element of vectors. It is a matter of partial order of events. Now: <u>V(a)<V(b) <--> (a→b)</u>

<u>Disadvanages:</u>

- More storage and message payload
- Optimizations exist

Question4: How does distributed mutual explusion work? Compare different algorithms (centralized, distributed, token-ring) with respect to scalability and fault tolerance?

Distributed Mutual Exclusion: (slide 32 PDF)

The aim of distributed mutual exclusion is to allow exclusive access to resources. That is, two processes shouldn't access the one resource at the same time.

Distributed mutex conditions:

- Safety: At most one process may execute in the critial section at a time
- Liveness: no deadlock, no starvation: requests to enter and exit the critical section eventually succeed
- Ordering: Happened-before (→ Fairness)

There are different ways mutual exclusion can be achieved:

Centralized: (slide 33 PDF)

One process acts as coordinator. Other processes request access to resources and the coordinator ensures that every resource is accessed by only one process at any given time. A process tells the coordinator if he doesn't need a resource anymore. There are queues for processes waiting for resources.

Disadvantages
If the coordinator crashed, the system doesn't work anymore

Distributed: (slide 34 PDF)

There is no coordinator! A process requests access by sending a message to all other processes, attaching his timestamp (lamport clock). A recipient of such a request has three options:

If he is using the resource: DENIED

If he is not using the resource: OK

If he plans on using the resource, OK or DENIED (depending on timestamp of request)

Disadvantages
If each process waits for an OK for every request, there are multiple points of failure. As soon as one of the processes crashed, the principle doesn't work anymore. Also, bottlenecks are likely because each process gets asked for access every time a process requests access to a resource. There is no sharing of the total work load.

Token ring: (slide 35 PDF)

A token gets passed along between processes. If a process holds the appropriate token for a resource, he is allowed to access it, i.e. he is allowed to access the critical region.

Disadvantages
If the process holding a token crashes, the token gets lost and it is difficult to determine when that happened. If no one wants to access a resource, the token stills needs to be passed around among the processes. The Safety condition is only met if there is only one token

<u>Decentralised probabilistic algorithm</u> (slide 36 PDF) NOT IN QUESTION???

Each resource is replicated n times, every replica has its own coordinator. When requesting access to a resource, only a majority vote is needed.

- ➔ If more than half of the coordinators answer with "OK", access is granted

The idea is that when a coordinator crashed, it recovers quickly but will have forgotten the votes it gave before it crashed. Also, there is no single point of failure in this approach.

Disadvantages
If k=2m-n coordinators fail, the voting correctness is violated, but probability is extremely low Low efficiency

<u>Comparison</u> (slide 37 PDF)

Algorithm	Messages per entry/exit	Delay before entry (in message times)	Problems
Centralized	3	2	Coordinator crash
Decentralized	3mk, k = 1,2,...	2 m	Starvation, low efficiency
Distributed	2 (n – 1)	2 (n – 1)	Crash of any process
Token ring	1 to ∞	0 to n – 1	Lost token, process crash

Question 6 Slide 6: p. 66-75

What are the problems for recording the global state of a distributed system and how can they be solved? Explain the algorithm.

Global State

- Time is very much related to the notion of global state.
- If we cannot agree on a time how should we agree on a global state

Global state is important:

- garbage collection
- dead-lock detection
- termination
- debugging

Problem: inconsistent cut

made by the system. (e.g.Receiving a message without sending this message)

a) A consistent cut
b) An inconsistent cut (effect without cause)

- No one has a global view of the system.
- There is no common time for recording ("date")

Snapshot – Algorithm by Chandy and Lamport

is an algorithm used in distributed systems for recording a consistent global state of an asynchronous system.

- The algorithm can be initiated by any process by executing the "Marker Sending Rule" by which it records its local state and sends a marker on each outgoing channel.
- A process executes the "Marker Receiving Rule" on receiving a marker. If the process has not yet recorded its local state, it records the state of the channel on which the marker is received as empty and executes the "Marker Sending Rule" to record its local state.
- The algorithm terminates after each process has received a marker on all of its incoming channels.
- All the local snapshots get disseminated to all other processes and all the processes can determine the global state.