Distributed System Paradigms (22)

7. Coordination

Problem behind: How to, mutually exclusive, share the resource "common memory" in a distributed system:
- explicitly exchanging messages, i.e. memory is considered as a local resource of a participant
- sharing common, possibly physically distributed, memory, i.e. memory is considered as a global resource

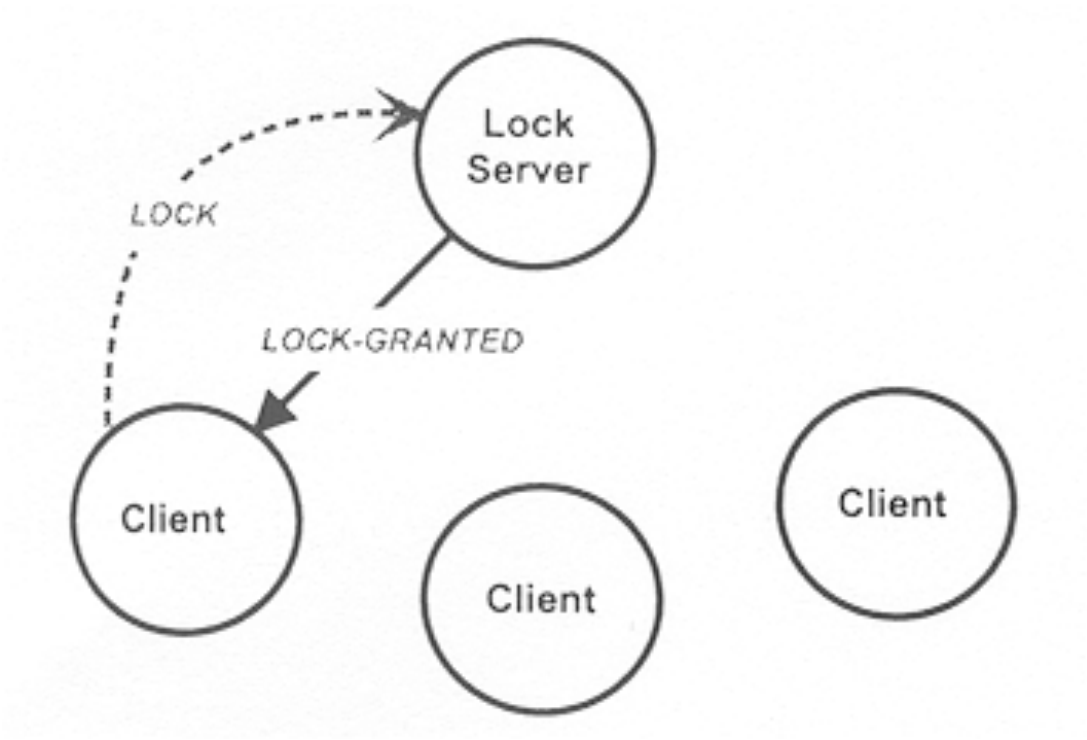
What about the other to main resources in a distributed computer system?
- network: done by the underlying network itself (job of the MAC layer in the layered network architecture)
- processors: act as local autonomous units in distributed systems (in contrast to master/slave systems)

Mutually exclusive sharing memory:
- using the *lock* mechanism, if memory is considered as one indivisible unit (Mutex variable)
- using the *semaphore* mechanism, if memory is considered as a finite number of indivisible units
    - it holds a number of memory units, which is defined when the semaphore is created
    - the *wait (n)* primitive decrements n units from the semaphore
    - it blocks if not enough available ---> requesting process waits until enough available
    - the *signal* primitive increments the number of units available, thus always being non-blocking

Distributed System Paradigms (24)

Distributed mutual exclusion (one-unit semaphore), i.e shared memory is physically distributed

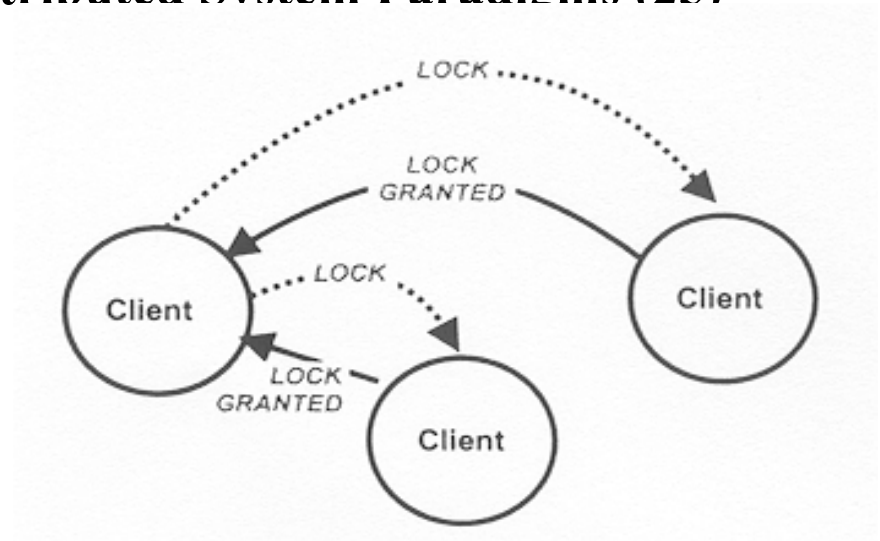
The lock server (centralized):



Con´s:
- single point-of-failure
- failure of the client holding the resource also wrecks the algorithm (no UNLOCK message will be sent)
- performance bottleneck

Distributed System Paradigms (25)

The lock server (distributed):



Problem: *inconsistencies* possible ending in a *deadlock* situation

Solutions:
- Lock request messages to all processes must be delivered totally ordered (see 6.)
  Con: very expensive
- agreeing on temporal leadership based on a distributed *leader election* protocol

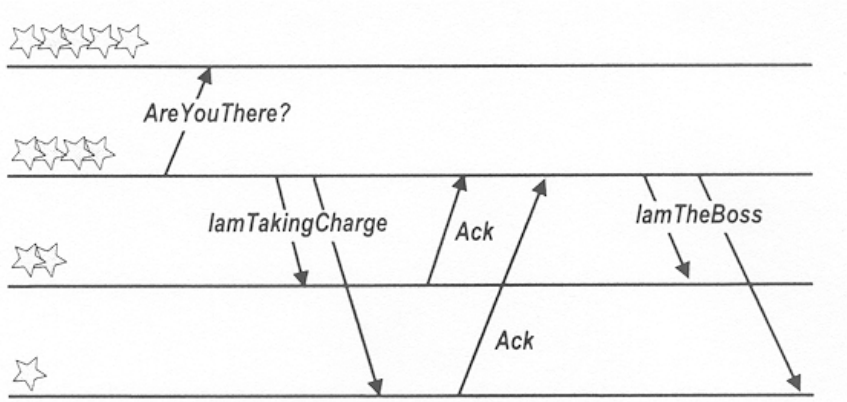
Leader Election

similar to mutual exclusion except but
- can be aborted as soon as a leader is elected (no fairness necessary)
- a process can support election of someone else

Example: The bully algorithm
- each candidate sends a request message timestamped with the logical clock
- the busiest, most active one (the one with the highest timestamp) becomes the leader
- in case of identical timestamps the one with the lowest identifier is elected as a tie breaker

Distributed System Paradigms (26)

The *rank* variant:



deadlock: very important phenomenon in distributed systems preventing system progress(no liveness!)
deadlocks can occur, as long as the following four conditions hold:

- *mutual exclusion*
- *hold-and-wait*
- *no-preemption*
- *circular-wait*

Prevention of deadlocks by rendering impossible ***one*** of these conditions:

- having only sharable resources --> too restrictive
- processes just hold one resource at a time --> too restrictive
  Alternatively: process must hold all needed resources a priori --> inefficient (bad performance)
- allowing preemption --> possibility of process *rollback*
- forcing processes to acquire all resources (which are totally ordered) by the same order
  --> inefficient (bad performance)

Distributed System Paradigms (27)

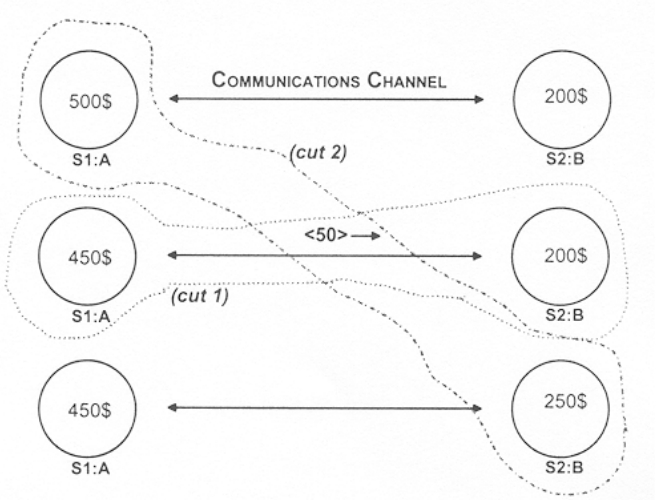
Alternatives
- *deadlock detection* (finding circular-wait conditions) *and resolution* (aborting one or several processes)
- *deadlock avoidance* (making checks before a resource is granted to check that a deadlock cannot occur)

Problem for both: considerable run-time overhead

8. Consistency

determining consistent global states

Taking ad-hoc state snapshots by means of a cut:

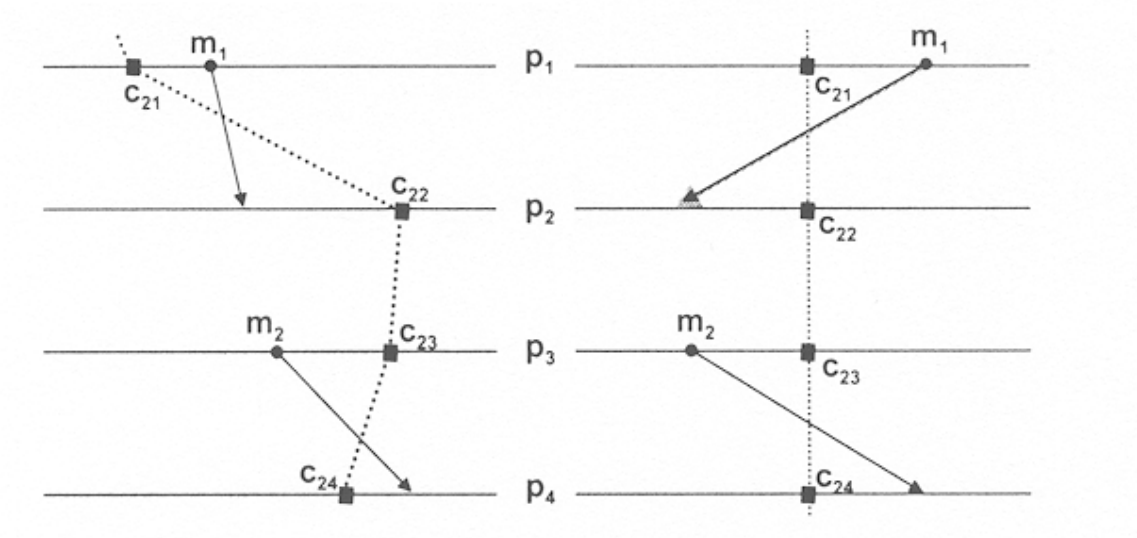


Cut 1: sum of money in the system is not correctly represented --> not *strongly consistent*
considering also messages in transit, states can be correctly updated -->
(weakly) consistent, i.e. consistent with the expectable system evolution

Cut 2: neither of the two views on cut 1 applies --> *inconsistent*

Distributed System Paradigms (28)

Graphical representation of conditions for consistency



cut line: dotted line concatenating the states of a cut

strong consistency --> no cut line intersects a message line

weak consistency --> if cut line intersects message line, then no state of the involved cut line which belongs to the message receiving node (process) is ahead of the time (younger) than the point in time where the message is received (message line crosses cut line from left to right)

inconsistent --> not strong or weak consistent (message line crosses cut line from right to left)

Distributed System Paradigms (29)

- Trying to obtain a global state simply by checking all contributing local states without any additional coordination may lead to inconsistent cuts (views, global state snapshots)

Brute force - solution:

stop the whole system to get a consistent cut (done in a number of commercial settings)

Question:

Can it be done on-line, i.e. while the system is active?

Example: *Snapshot protocol by Chandy/Lamport*

Idea: Assuming FIFO channels to connect processes and using "markers" (special control message) to distinguish between messages sent before and after a snapshot has been taken. The global snapshot includes the local state of each process ***and the state of each channel.*** Each process is responsible for recording the state of all its incoming channels.

- The initiating process saves its state and then sends a marker through all its outgoing FIFO channels
- It continues its normal operation while waiting for receiving a marker from all its incoming channels
- All messages in each channel that are recorded as being received after the process has saved its state and before the marker is received via the channel constitute *the state of the respective channel*.
  All other processes behave similarly:
- When they receive a marker for the first time, they save the state of that channel as empty.
  Then they save their own state and send markers through all outgoing channels while waiting for markers for the remaining incoming channels
- When additional markers are received (local state is already saved), the corresponding channel state is recorded as done by the initiating process.
- The algorithm terminates when all process states are saved and all channel states are recorded