COMPUTER SCIENCE & ENGINEERING Göteborg SAA	EXAMINATION 2010-04-07	HT-09 CTH-TDA596 GU-DIT240

DISTRIBUTED SYSTEMS examination

DAY: 7/4 - 10 TIME: 14 - 18 ROOMS: J

Responsible: Sven-Arne Andreasson 1043

Results ready: see course homepage for information

Grades: GU: G 24p, VG 42p
CTH: 3:a 24p, 4:a 36p, 5:a 48p
of maximum 60 points.

Allowed aids: Nothing except paper, pencil and English - xx dictionary.

NOTE:

- All questions **MUST** be answered in English only!
- Write clearly and use the pages in a clever way so it is easy to read.
- Each task should be started on a new sheet. Use only one side of each paper.
- When describing an algorithm (protocol) use numbered paragraphs in order to make it easier to read (and get right).
- All answers should be motivated!

Question 1) Two processes *A* and *B* should execute a task together. The processes are cooperating by sending messages through an unsafe communication network where messages can be corrupted or disappear.

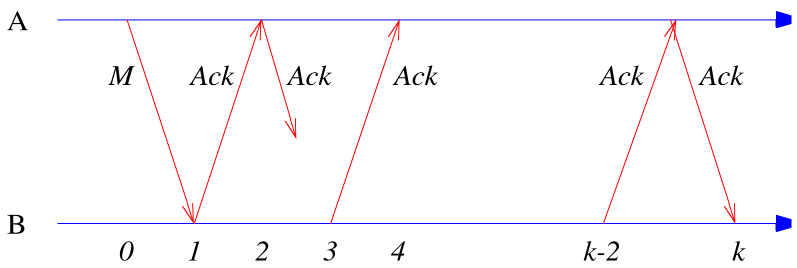
a) Show that they can not be synchronized in such a way that they will change state simultaneously.

The Coordinated Attack Problem:

Presumption: Two processes communicate using messages on an unreliable medium where messages can be corrupted or disappear.

Statement: It is impossible for the two processes to agree on a specific point of time when both should change their states simultaneously.

Proof: here we will show that it is not possible even in a simplified case when the messages have a constant network transmission time (if the message arrives).



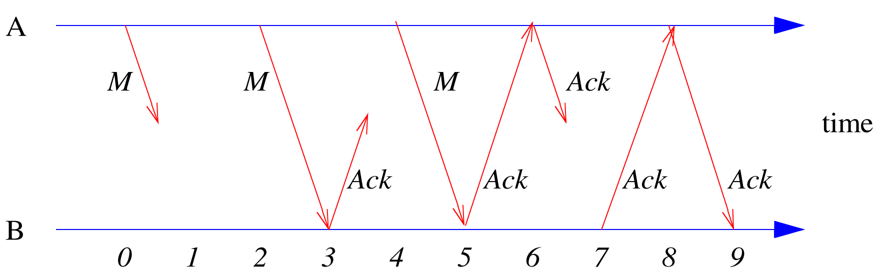
If we can agree at time *k* then the last message can not be part of the decision since process A does not know if it has arrived or not. Then the decision could as well be done at time *k-1*. But then process B does not know if its last message has arrived or not so it can not be part of the decision. Then the decision could as well be done at time *k-2*., and so forth. We will find the no message can be part of the decision and then no decision can be made.

6 points

b) How can they still cooperate?

It is possible for two processes to decide that they change their states **some time** in the future but **not** at the same time.

State: S_1 S_2



State: S_1 S_2

The system **“converges”** to a consistent state.

2 points

c) Give examples of applications where this statement influences the implementation and in what way.

ATM, commuication procols, synchronization of distributed processes.

2 points

(10 points)

Question 2) Describe an algorithm that uses *Logical Clocks* to allocate resources in a distributed system.

a) What are the prerequisists?

- The algorithm uses
  - Distributed Request Queue (empty at start)
  - Logical Clock
- all messages between two processes is delivered in the same order as they were sent and no message will disappear (FIFO).
  This will be the case when using a communication protocol such as TCP/IP.

2 points

b) Describe the algorithm using a state-event model (same as in the lecture notes).

The algorithm events

- P_i *REQUEST*:
  - $<T_i;P_i;REQUEST>$ is sent to all other processes, T_i is the timestamp taken from C_i
  - $<T_i;P_i;REQUEST>$ is also put in P_i's local copy of the Request Queue sorted according to "$\rightarrow_T$"
  - P_i increments its local Logical Clock value
- P_j receives *REQUEST* $<T_i;P_i;REQUEST>$:
  - P_j adjusts its local copy of the Logical Clock according to the Logical Clock definition
  - $<T_i;P_i;REQUEST>$ is put in P_j's local copy of the Request Queue sorted according to "$\rightarrow_T$"
  - P_j updates its TS-table
  - P_j increments its local Logical Clock value
  - P_j sends an acknowledgement $<T_j;P_j;ACK>$ to P_i
  - P_j increments its local Logical Clock value
- P_i receives an acknowledgement $<T_j;P_j;ACK>$ from P_j:
  - P_i adjusts its local copy of the Logical Clock according to the Logical Clock definition.
  - P_i updates its TS-table with T_j from P_j
  - P_i increments its local Logical Clock value
- P_i is allowed access to the resource when:
  - $<T_i;P_i;REQUEST>$ is number one in the (local) Request Queue
  - $T_i \rightarrow_T T_j$ for all T_j in the (local) TS-table
- P_i want to *RELEASE*:
  - $<T_i';P_i;RELEASE>$ is sent to all other processes, T_i' is the timestamp taken from actual C_i
  - $<T_i;P_i;REQUEST>$ is erased from the (local) Request Queue
  - P_i increments its local Logical Clock value
- P_j receives $<T_i';P_i;RELEASE>$:
  - P_j adjusts its local copy of the Logical Clock according to the Logical Clock definition.
  - $<T_i;P_i;REQUEST>$ is erased from the (local) Request Queue
  - P_j updates its TS-table with T_i' from P_i
  - P_j increments its local Logical Clock value

7 points

c) Give a short example.

1 points

(10 points)

Question 3) Consider the Snapshot algorithm.

a) What are the prerequisites for the algorithm?

Algorithm conditions:

- the nodes are part of a directed graph, real or virtual
- the graph is strongly connected, i.e. there is a path from any node to any other node
- the algorithm messages are only sent according to the directed graph
- FIFO secure transmissions on the communication links (e.g. TCP/IP)
- the process can continue working and change their states while the snapshot takes place

2 points

b) What is it that the algorithm computes and why is it defined this way?

Calculates an estimated global state of the system consistent with the real state.

- this might never have existed
- but this state is a possible state that doesn't conflict with the real states and might be used for system control.

1 points

c) Describe the algorithm.

uses *Marker Messages* (*MM*)

Local Calculated State (*LS*)

A node initiates the algorithm

- records its local state, *LS*
- puts itself in recording state
- sends an *MM*-message on each out-going link

When a node gets its first *MM*-message on an in-going link

- records its local state, *LS*
- puts itself in recording state

- marks the in-going link as ready
- sends an *MM*-message on each out-going link

When a node in recording state gets an *MM*-message on an in-going link

- marks the in-going link as ready

When a node in recording state gets another message on an in-going link

- if the in-going link is not marked as ready the recorded state, *LS,* will be recalculated according to the message
- if the in-going link is marked as ready the message will not affect the calculated state

When a node in recording state has got an *MM*-message on all in-going links

- its part of the global calculated state, *LS,* is ready.
- its recorded state is
  - sent to one node for assembling the global calculated state
  - or sent to all other nodes so all can calculate the global state
- it leaves recording state

A calculating node that has got the local calculated states, *LS,* from all other nodes can put them together to the Calculated Global State

7 points

(10 points)

Question 4) Atomic Transactions.

a) What is meant by that a transaction is *atomic.*

A transaction must obey the following properties:

- Atomicity—all or nothing
- Consistency—take the system from one consistent state to another consistent state,
  e.g. in a banking system it might be that the sum of all accounts is constant.
- Isolation—no illegitimate influence among different transactions.
- Durability—the write operations of a committed transaction must hold for the future.

4 points

b) Describe an algorithm for making transactions atomic in a distributed system.
(10 points)

two-phase commit

phase 1:

- Lock transaction data.

- perform reads and writes in normal order **but**
- write on an intention list instead of changing the corresponding memory addresses.
- The intention list should be stored on stable storage,
  i.e. memory that survives a processor crash, e.g. a hard disk.

commit - the intention list is made valid

phase 2:

- write to the memory addresses according what has been stored on the intention list.
- Then the locks can be released.

If after a crash the intention list

- is marked as valid the transaction was committed before the crash occurred.
  - The write operations on the intention list are performed
- then the locks are released.
- is not marked as valid the transaction was not committed before the crash occurred.
  - The locks are released.

The transaction might be restarted.

6 points

Question 5) Give an algorithm for atomic broadcast in a general network with synchronized clocks. The algoritm should allow *omission failures*.

a) What are the prerequisites?.

G is a network with n nodes and m links.

- Each node has a physical clock.
- Node p's clock is denoted by C_p
- $C_p(t)$ denotes the clock value at the real time t.

Assumptions:

- The nodes have a unique names that are totally ordered.
- F is the set of nodes with links that got failures during the atomic broadcast.

- G-F is the sub net of G where links and nodes work correctly.
  Assume that G-F is connected,
  i.e. the protocol does not allow a network partition.

- Each clock is going forward.
  Two read operations on the same clock should give different values.
  i.e. a clock is not allowed to stop or to be to slow.

- Also the clocks should be synchronized in such a way that there is a small value ε such that for every real point of time t it should hold:
  $\forall$ p,q$\in$G-F: $| C_p(t) - C_q(t) | < \varepsilon$.

- There is a maximal time for sending and treating the messages that the protocol uses, δ.

3 points

b) Describe the algorithm.

Protocol that survives "omission failure"

The protocol uses flooding.

- The flooding is interrupted at each node when a message arrives a second time. (the first technique we described for flooding termination)

The message format is: `(T,s,σ)`

- T is a timestamp that gives the initiating time,
- s is the senders unique name,
- σ is the information that should be delivered (the broadcast message).

The messages will get a unique identity: (T,s)

The received messages are placed in a log H (queue of broadcast messages waiting to be delivered)

- When there is no faults in the network a message will be received within the time dδ after it was sent from the originator
  d is the network diameter
  dδ will then be the maximal message jump times the maximal handling and transmission time.

- The messages are time stamped with the sending time T.
  When the clock in the receiver is $T + d\delta + \varepsilon$ there can not arrive a message from any nod with a lower timestamp than T.
  ε is the maximal error among the local clocks.

- For *omission failure* the time limit has to be extended to $T + \pi\delta + d\delta + \varepsilon$.
  Failures may increase the network diameter to $\pi + d$.

$\Delta \equiv \pi\delta + d\delta + \varepsilon$ is the protocol terminating time

6 points

c) Give a small example.

1 points

(10 points)

Question 6) Explain what is meant by the concept *transparency* in distributed systems.
Why are they desirable and what might be their disadvantages?
Give examples of some (>3) different types of transparency.

Programs developed for a single computer can be used without modifications.

Simplified program development.

- .Simplified system model for the developer.

Simplifies system reconfiguration.

- Programs don't have to be rewritten or even re parametrized.

Higher potential for reliability.

Simplified system model for users.

- System can be used without awareness of its configuration.

6 points

Network Transparency

Name Transparency

Location Transparency

Semantic Consistency

Access Transparency

Execution Transparency

Replication Transparency

Performance Transparency

Configuration Transparency

...

4 points

(10 points)