CHALMERS
EXAMINATION / TENTAMEN

Course code/ kurskod	Course name / kursnamn			
TDA297	Distributed systems Advanced			
Anonymous code Anonym kod		Examination date Tentamensdatum	Number of pages Antal blad	Grade Betyg
TDA297-19		2014-08-22	10	

Solved task Behandlade uppgifter. No / nr		Points per task Poäng på uppgiften.	Observe: Areas with bold contour are to be completed by the teacher. Anmärkning: Rutor inom bred kontur ifylles av lärare.
1	X	8	
2	X	10	
3	X	10	
4	X	10	
5	X	12	
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			
16			
17			
18			
Total examination points Summa poäng på tentamen		50	

TDA297 . 19

a) i) Reliable broadcast - if a correct process broadcasts a message m_1, all correct processes will eventually deliver m_1.

ii) FIFO broadcast - if a process broadcasts m_1 and then m_2, all correct processes will deliver m_1 before m_2. (4)

iii) Causal broadcast - if a process broadcasts m_2 and $m_1 \rightarrow m_2$, then all processes will deliver m_1 before m_2. ✓

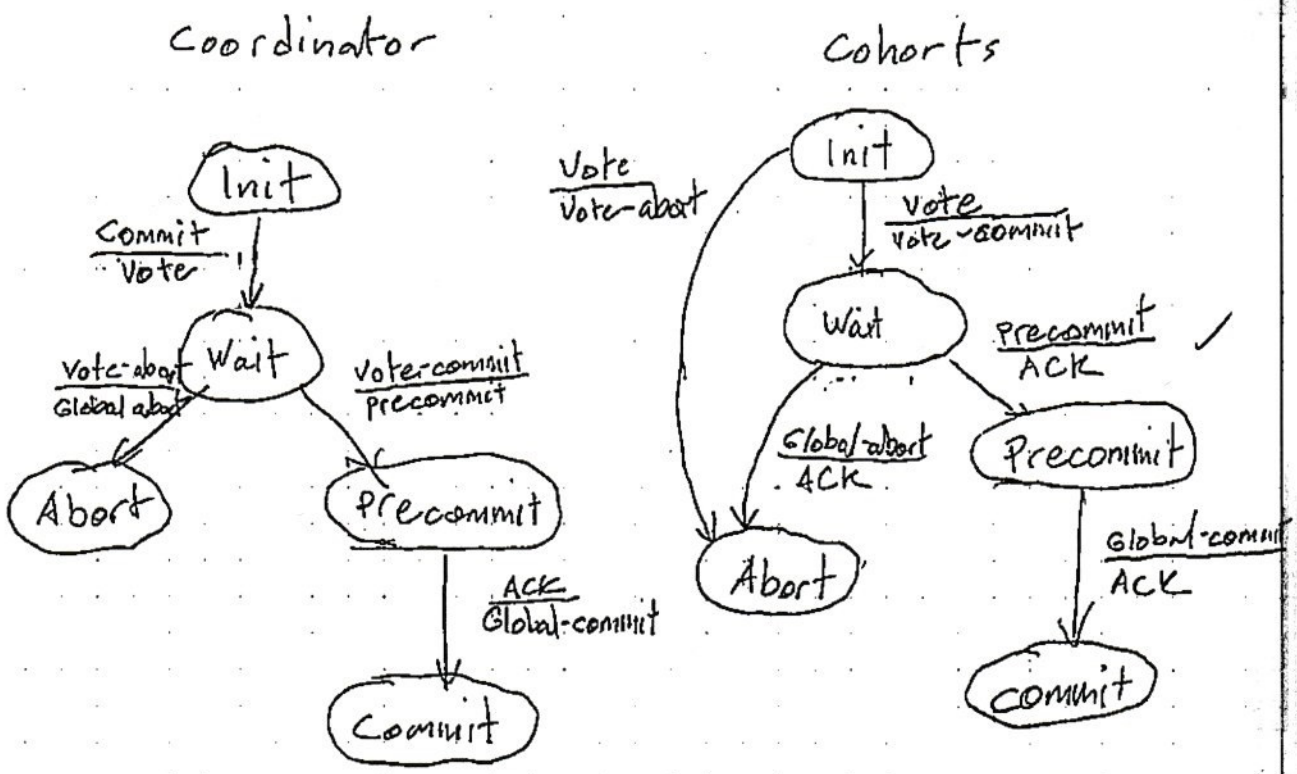
b) FIFO broadcast can be implemented with the use of flooding: (1)
The first time a process receives a message m_1, the process broadcasts the message to all his neighbors including the sender. ✓

c) The message complexity is two times the number of edges, $O(|E|)$, because the initiator starts sending the message over his edges and he gets a reply on each of those edges. (3)

The time complexity is the depth of the graph + 1, $O(D(G))$. because it takes $D(G)$ time for the furthest node to receive the broadcast and 1 time unit for him to broadcast the message.

An initiator begins by broadcasting message m to all neighbors. When a node receives m for the first time, from a process p, the node sets p as its parent. The node then broadcast the message m to all of its neighbors except its parent.
The node replies with ACKnowledgements to all successive messages m it receives.
When a node have received the same number of ACKs as messages it sent, it sends ACK to its parent. ✓
When the initiator receives all ACKs the algorithm is done and all nodes have a parent except the initiator which will be the root node of the minimum spanning tree. (10)

A node which now want to broadcast information can now use this minimum spanning tree instead of broadcasting to the full graph.
The number of messages needed to broadcast information will in the best case be much lesser.



10

The difference between the 3-phase commit protocol (3PC) and the 2-phase commit protocol (2PC) is the extra phase "precommit". If a process in 2PC dies and restarts, or if there is message loss, then the process cannot be sure where in the protocol it should resume. In the worst case, if a cohort dies when it has received a vote-message, then the coordinator waits for a message/a vote it will never receive, and all other cohorts wait for the coordinator to decide what to do. This is called blocking.

In 3PC this is solved by adding the extra phase "precommit".
If a cohort dies while the coordinator is waiting for voting-replies, then the coordinator does a timeout and assumes vote-abort.
If on the other hand the coordinator dies while the cohorts wait for a decision, then the cohorts timeout and assumes abort.
If a precommit-message has been received by the cohorts and the coordinator dies, the cohorts instead commits when the timeout has been reached.

Two generals communicate orders to each other with the use of messengers. How do one general know that the other general have received his order, assuming that the messengers can be captured/killed?

Assume that a general sends an Acknowledgement after receiving a message, how does that general know that the other general receives his Acknowledgement?

(10)

The problem is not solvable. Proof by recursion:

If the generals sends N ACK to an order then the lost message can be lost and we need to prove that N-1 ACKs is enough to know that the other general receives the message. But if only N-1 messages is sent, then we must prove that N-2 messages is all that is needed, and so on. One can never be certain that the other part has received the message in an asyncronous system with message loss.

A) Three-phase commit protocol is not suited for availability because of the all-or-nothing approach. For each commit all servers need to wait until all cohorts have voted and the coordinator have decided.
If the coordinator dies or loses network connection, then all cohorts assume abort, and the whole protocol will need to be restarted.

(5)

A replicated shared object service is said to be linearizable if for any execution there is some interleaving of the sequences of operations issued by all the clients that satisfies the following two criterias:

- the sequences of operations issued by all clients could be written as an interleaving of a single sequence of operations. (That is the sequences are mergable).
- The order of operations is in real time.

The definition of sequential consistency is similar but the second criteria is that the order of operations need not be in real time but in the same order.

5

TDA 297 - 19

The system Eric has is therefore both linearizable and sequential consistent, because linearizable is more strict and the three-phase commit protocol ensures that all replicas have the latest view. ✓

B) Because the updates is lazily propagated the system cannot be sequential consistent nor linearizable. Example below ✓

Client 1	Client 2
setBalance$_B$(x, 1)	getBalance$_A$(x) -> 0
	getBalance$_A$(y) -> 0
setBalance$_A$(y, 1)	

(3)

The updates are not propagated and there exists cases where reads can be of old values.

Eric can perform two operations in the same time because of the lazy propagation.

p(n)?

TDA297-19

5

c) To ensure atomicity for writes, that
he can't write two different values at the
same time he must choose a write quorum
such that $W > \frac{9}{2}$ (Bigger than half of the
servers) ✓
He should also choose read quorums so that
$W + R = 9$ Because then he is sure that at least
one server has the latest value.
Depending on if he has most writes or most reads
he can choose these quorums differently
For example he can choose $R = 4$ and $W = 6$
he then gets the following graph when processing
a write, two reads, then a write (4)

write, two reads write, ✓ p(4) ??

If he have mostly reads, he can choose a
smaller read quorum in order to allow more
reads at the same time. He must then also
choose a higher write quorum.

This system is sequential consistent because
a write is atomic and updates the majority
of servers. The read operation always reads
from a quorum of one server and has the

latest updated state. It is however not linearizable because all servers is not updated in real time.