Department Computer Science	**Distributed Systems**
Vrije Universiteit	**19.12.2012**

BE SURE THAT YOUR HANDWRITING IS READABLE

1a Consider a Chord ring based on 7-bit identifiers and keys, with nodes $\{15, 26, 47, 48, 51, 69, 75, 106, 110, 127\}$. Provide the finger tables for nodes 15, 106, 127. *6pt*

Node 15: [26, 26, 26, 26, 47, 47, 106]; Node 106: [110, 110, 110, 127, 127, 15, 47]; Node 127: [15, 15, 15, 15, 15, 47, 69].

1b Show all hops for the following key lookups for the Chord-based P2P system: *4pt*

source	key
15	12
26	3

3@26: [106, 127, 15]; 12@15: [106, 127, 15] (and if 12 knows its predecessor: no hops).

1c Suppose node 15 would (also) have discovered the existence of node 110. If it were asked to look up key 12, is forwarding the request to 110 an option? *5pt*

It depends on your answer to (a), but assuming that $FP_{15}[7] = 106$, the answer would be "yes," provided that the distance between nodes 15 and 110 is less than that between 15 and 106. The underlying assumption is that node 106 will also have a shorter route to the destination, where distance is now measured in network latency instead of only overlap hops (which generally remain the same). In hindsight, the question was confusing: if node 12 knows its predecessor, it would never forward the request at all. The answer to (a) was taken into account when grading.

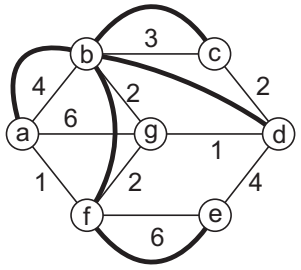
1d Explain briefly how you would implement a DNS-like naming scheme in Chord. *5pt*

Any domain name "x.y.z" would be hashed to a key k and its associated record(s) would be stored at the node responsible for k.

2a Describe what is meant by an overlay network. Be concise! *4pt*

An overlay network comprises a number of processes that maintain "virtual" links to each other: at any moment in time, a process will known about a number of other processes (and no others) with whom it can communicate. Technically, the processes constitute a network, which is referred to as an overlay network.

2b Compute the stretch (also called the relative delay penalty) for the following overlay network (shown in bold) and its underlying network, for all pairs of nodes.



6pt

The complete "stretch" table is as follows:

	a	*b*	*c*	*d*	*e*	*f*
a	-	*4/4*	*7/6*	*7/4*	*14/7*	*8/1*
b		-	*3/3*	*3/3*	*10/7*	*4/4*
c			-	*6/2*	*13/6*	*7/5*
d				-	*13/4*	*7/3*
e					-	*6/6*

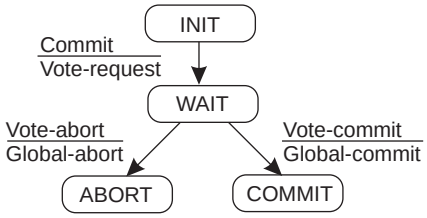
2c Explain what link stress is, and provide examples from the overlay in (b) where the stress is 0, 1, or 2, respectively. *5pt*

Link stress counts how often a packet passes through a specific link in the underlying overlay when being routed in the overlay. Links that are never used for overlay routing have stress 0 (such as $[a,f]$). Link $[a,b]$ is an example where the stress is always 1, whereas $[b,g]$ has links stress 2 for packets routed between node d and f.

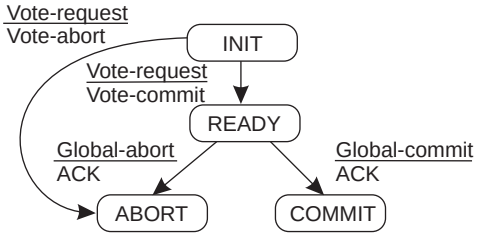
2d Consider the organization of a Chord network like the one used in Q1. How can we attempt to minimize the average stretch? *5pt*

This question is really about optimizing for network proximity. What we can do is suggested in Q1(c): let each node p keep track of other nodes in the interval $[FT_p[i], FT_p[i+1])$ and, if possible, forward a request for key k to any node q in that interval closer to p that would still satisfy the constraint that $q \leq k$. (I'm skipping most of the details here).

3a Explain the working of two-phase commit by providing the finite state machine diagrams for the coordinator and the participants. *6pt*



(a)



(b)

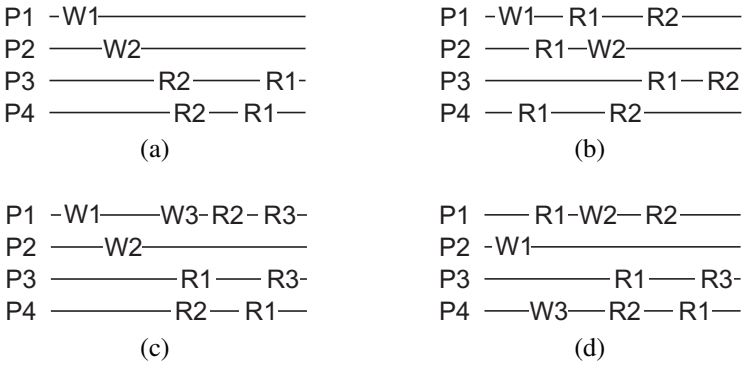
3b Suppose that participant *P* is ready to commit, but that a timeout occurs because it hasn't received an instruction from the coordinator. *P* decides to check the state of the other processes. Explain what it does when it finds participant *Q* in a specific state. *5pt*

If Q is ready as well, P will have to probe another participant. If Q is in the abort or commit state, P will just follow Q. If Q is still in its initial state, P will abort the transaction (knowing that Q will do this most likely as well).

3c Suppose that participant *P* is in the initial state. What should it do when it receives a vote request from the coordinator, notably when it is almost ready to commit. Be sure to explain your answer! *4pt*

It's easiest if P aborts its part of the transaction as it really doesn't know if the transaction can be committed. If it would move to the ready state, then another participant Q who would be in the ready state and probing the other participants, would have to wait until P made up its mind.

4a Consider the following four executions. We write $W1$ as an abbreviation for $W(x)1$, and likewise $R2$ for $R(x)2$, where x the variable shared by the four processes $P_1, \ldots, P_4$. Which of these four are sequentially consistent, and which ones are not? Explain your answer. Hint: think twice in cases (c) and (d).



6pt

(a) and (b) are sequentially consistent. Case (c) is not: after P_1 writes $W3$, yet finds $R2$, it must conclude that it was too late and that the shared variable x has the value 2. Later reading it has 3 doesn't make sense. Likewise, in (d) P_4 finds its write $W3$ didn't make it, and apparently x got value 2. Later, it gets value 1. For this reason, P_3 should not be able to see value 3 pop up after it has read $R1$.

4b Give two straightforward solutions for implementing sequential consistency. *4pt*

A simple one is where each operation is sent to a sequencer that simply timestamps every operation. Operations are subsequently carried out in the order of their timestamp. Another approach is to use totally ordered broadcasting, e.g., implemented using Lamport clocks.

4c Give an example where using only client-centric consistency will lead to a conflict between update operations. *5pt*

Imagine a process doing updates on a shared data item at locations L_1 and L_2. If in the meantime someone else is doing updates at location L_3, then sooner or later, updates are bound to conflict.

5a Consider flooding-based consensus between four processes $P_1, \ldots, P_4$. Assume P_3 has received proposed operations from P_2 and P_4, but also detected that P_1 crashed. Explain why P_3 cannot make a decision on which operation to execute.*P_3 has no clue whether the other processes have seen the operation proposed by P_1, and that they may decide to execute that operation.* *6pt*

5b Flooding-based consensus assumes fail-stop semantics and reliable failure detection. What does this mean? *4pt*

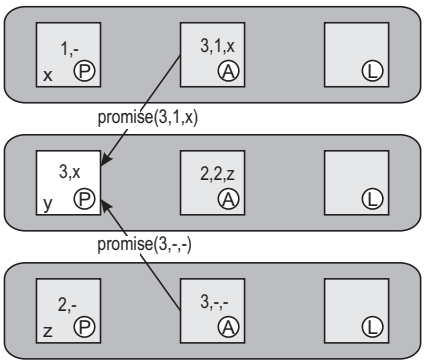
Fail-stop semantics means that when a process crashes, this can be reliably detected. Reliable failure detection means that a process P can indeed reliably detect that Q crashed. The assumption works only for synchronous systems.

5c In Paxos, a proposer maintains two state variables, and an acceptor three. Explain what those variables stand for. *4pt*

Proposer maintains the current proposal number, along with the proposed operation. An acceptor maintains the proposed proposal number, the last proposal number that was accepted, and the associated operation.

5d Consider the following situation in Paxos. What will be the response of the proposer, the subsequent

state changes in each of the acceptors, and their reaction to the state change?



6pt

The proposer will react with a accept$(3,x)$ message, after which each of the acceptors will switch tot state $(3,3,x)$, leading to a multicast of "x" to each of the learners.

Grading: *The final grade is calculated by accumulating the scores per question (maximum: 90 points), and adding 10 bonus points. The maximum total is therefore 100 points.*