Department Computer Science	**Distributed Systems**
Vrije Universiteit	**06.02.2013**

BE SURE THAT YOUR HANDWRITING IS READABLE

1a Explain how iterative name resolution in DNS works and why it may incur high latency costs. Also give an example where recursive and iterative resolution is effectively combined. *6pt*

In essence, you should provide and explain Figures 5-15. The combination may happen at the lower layers, which are generally controlled by an ISP. In that case, we're essentially letting the ISP control the flow of DNS requests, which may be sensible for internal redirection and caching policies.

1b When resolving name www.distributed-systems.net, DNS will return an address, but not when trying to resolve distributed-systems.net. How can this be? *6pt*

DNS maintains domain names; not every domain name is necessarily associated with an IP address. In fact, many names are really just like directories. Obviously, each domain name does have an associated name server, but that's something else.

1c User `maarten` in domain `van-steen.net` has a forwarding mail address to `steen@cs.vu.nl`. Show which messages are exchanged between a user mail agent sending a message to `maarten@van-steen.net`, DNS, mail servers, and the recipient user agent. *10pt*

1. SND requests MX record for van-steen.net. 2. DNS returns domain name. 3. SND requests IP address of mail server of van-steen.net; DNS returns IP address (4 is often returned along with 2). 5. SND sends mail to mail server MS1 at van-steen.net. 6. MS1 looks up mail server for cs.vu.nl (as before). 7. MS1 sends mail to mail server MS2 at cs.vu.nl. 8. MS2 forwards mail to user's mail box/user agent.

2a Explain how Lamport's logical clocks work. *6pt*

Every process P_i keeps a local counter C_i, which is initially set to 0. When a message m is to be sent, P_i increments C_i and adds it as a timestamp $T(m)$ to m. When process P_j receives a message m, it sets its own value C_j to $\max\{C_j, T(m)\}$, and subsequently increments C_j.

2b Explain how totally ordered multicasting can be implemented with Lamport's logical clocks. *8pt*

The essence is as follows. When process P_i wants to update the replicas, it broadcasts the update m_i to itself and all other processes. Message m_i is timestamped with P_i's current value of C_i. When a process P_j receives m_i it puts it in a local queue, ordered by $T(m_i)$, and broadcasts an acknowledgement. Only when m_i is at the head of the queue, and for every process there is a message queued with a higher timestamp, will m_i be passed to the application. Lamport assumes that messages are sent FIFO-wise, and that no messages are lost.

2c Provide an alternative implementation that realizes totally ordered multicasting. *6pt*

A simple solution is to use a sequencer: when P wants to send m, it contacts the sequencer to get a unique sequence number s (the sequencer will then increment its local counter). Message m is then timestamped with s and subsequently sent to all processes. When receiving a message m, the recipient will pass the message to the application if and only if it is the next message in line.

3a In continuous consistency, consistency is measured in three different dimensions. What are those three dimensions, and for each one, give a compelling example illustrating its use. *6pt*

The three dimensions are numerical values, time (staleness), and order of operations. They are used to describe to what extent replicas may differ before they are considered to be inconsistent. Value-based continuous consistency can be used in the case of stocks (how far are two stocks allowed to differ). Time-based continuous consistency is typically used for caching Web pages. Order-based consistency can be used to express how many transactions one replica of a database may lag behinf another replica.

3b For continuous consistency and N replica servers $\{S_1, \ldots, S_N\}$, we use the notation $TW_k[i,j]$ to express the view of which operations originating at server S_i that server S_k believes have been propagated to server S_j. Explain what each of $TW_i[i,j]$, $TW_j[i,j]$, and $\sum_{i=1}^{N} TW_i[i,i]$ stand for. Also explain if $TW_i[i,j] = TW_j[i,j]$ *8pt*

$TW_i[i,j]$ are exactly the operations that have been submitted to S_i, that S_i propagated to S_j. $TW_j[i,j]$ are the operations that S_j has received from S_i and that had been submitted to S_i. The two are always the same if and only if we can assume that communicaton is reliable. $\sum_{i=1}^{N} TW_i[i,i]$ is nothing but the sum of all submitted operations.

3c Give a formal expression that describes the fact that a value x is numerically bounded by δ units, in terms of TW as used in (b). Also explain how that bound can be enforced. *6pt*

We need to ensure that $|\sum_i TW_i[i,i] - \sum_i TW_i[i,j]| < \delta$ for each server S_j. This can be achieved by letting server S_i propagate its log to server S_j as soon as $|TW_i[i,i] - TW_i[i,j]|$ exceeds $\delta/(N-1)$.

4a How large must a k-fault tolerant group be? *6pt*

It all depends on your assumptions about failure semantics. With anything as simple as crash-failure semantics, you need $k+1$ processes. As soon as we're dealing with arbitrary failures, we need $2k+1$ members. Finally, if the group is assumed to reach consensus under arbitrary failures, we will need $3k+1$ members.

4b Consider a primary-backup replication scheme with one primary and two backup processes. For which k is this a k-fault tolerant group, and what are the underlying assumptions? Explain your answer! *6pt*

If we assume that a failing primary can be detected and that a new primary can be elected, then depending on our failure semantics, $k = 2$ or $k = 1$. In this case, we also need to assume that a client can contact any of the three processes, and that it may even have to contact all three of them. If the primary is fixed and irreplaceable, $k = 0$.

4c Consider a k-fault tolerant group, with $k > 1$. Assume that one process fails. Do we still have a k-fault tolerant group? Explain your answer! *4pt*

By definition, a k-fault tolerant group is a group that can withstand the failure of k processes. If $k > 1$ and 1 process fails, then by definition we still have a k-fault tolerant group. The trick, of course, is how you measure the group: if it still includes the failing process, nothing changes. If the failed process is permanently removed from the group, then the remaining group is not *k-fault tolerant.*

4d Consider a process group with N members running Paxos. Assuming crash-failure semantics, how many failed processes can this group tolerate to continue meeting its specifications? *4pt*

Paxos requires $2k+1$ processes to survive a single crashed process. This means that it can tolerate $\lfloor (N-1)/2 \rfloor$ failing processes. So, for $N = 5$, $k = 2$.

4f Paxos works in two phases. Explain the first phase by providing the flow of messages between a *proposer* and *acceptors*, along with the respective states and state transitions. *8pt*

Assume a proposer has state $\langle r,o \rangle$, meaning that its current proposal number is r, and that it's proposing operation o. It sends prepare(r,o) *to a a majority of acceptors. An acceptor reacts as follows. Assume its local state is $\langle r_p, r_a, o_a \rangle$ with r_p being the highest promised proposal number; r_a the highest accepted-proposal number, and o_a the associated accepted operation. If $r_p > r$, the acceptor does nothing. Otherwise, it returns the message* promise(r, r_a, o_a) *(note that r_p may be equal to "-", and likewise for o_a).*

***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.*