Exam Distributed Algorithms

Vrije Universiteit Amsterdam, 1 June 2022, 18:45-21:30

(You may use the textbook Distributed Algorithms: An Intuitive Approach*. Use of slides, solutions to exercises, notes, laptop, calculator is* not *allowed.)*

(The exercises in this exam sum up to 90 points; each student gets 10 points bonus.)

1. Suppose that in a run of the Bracha-Toueg deadlock detection algorithm, some NOTIFY, DONE, GRANT, or ACK message is in transit. Explain why the initiator of this run can then not yet have received a DONE from all neighbors to which it sent a NOTIFY. (14 pts)

   Solution: Suppose a **notify** is in transit from p to q. Then q has not yet sent a **done** to p. So if p is a noninitiator it has not yet sent a **done** to its parent, etc., meaning the initiator is still waiting for one or more **done**'s to arrive.

   Likewise if a **done** is in transit from q to p.

   Suppose a **grant** is in transit from p to q. Then q has not yet sent an **ack** to p. So if p is not the initiator of this **grant**/**ack** subtree, it has not yet sent an **ack** to its parent in this subtree, etc., meaning the initiator from this subtree is still waiting for one or more **ack**'s to arrive before it can send a **done** to its parent (or, in case this is also the initiator of the entire run, before it can check its *requests* counter). Now as before it can be argued that the initiator of the entire run is still waiting for **done**'s to arrive.

   Likewise if an **ack** is in transit from q to p.

2. Adapt the tree election algorithm so that the *initiator* with the largest ID becomes the leader. (9 pts)

   Solution: In the computation of $\max_p$, process p only takes into account its own ID if it is an initiator. In particular, if a leaf is a noninitiator, it sends a bottom element $\perp$ to its parent.

   Alternatively, noninitiators can in tree election replace their own ID by a number smaller than all regular IDs.

3. Suppose that, at some point in the Gallager-Humblet-Spira minimum spanning tree algorithm, a process p receives a message $\langle \textbf{test}, fn, \ell \rangle$ through channel pq, where p's fragment has a different name than fn and at least level ℓ. Explain why p can send an **accept** message to q without fear that p and q are in the same fragment. (13 pts)

   Solution: We make two observations.

   1. If a process is in the state *find*, then it carries the most up-to-date name and level of its fragment.

   Namely, if a process is in the state *find*, then it received a message $\langle \textbf{initiate}, fn, \ell, find \rangle$ with the most up-to-date name fn and level ℓ of its fragment, and the core nodes of its fragment are waiting for messages regarding the lowest-weight outgoing edge of this fragment.

   2. When the name of a fragment changes at a process, always at the same time its level increases.

   Since q sends a **test** message to p, q must be in the state *find*. So by observation 1, q carries the most up-to-date name and level of its fragment F.

   Suppose, toward a contradiction, that p is also in the fragment F. Then by observation 2, the assumption $name_p \neq name_q$ would imply that $level_p < level_q$. This contradicts the assumption that $level_p \geq level_q$.

   Concluding, p and q are not in the same fragment.

4. There is no Las Vegas algorithm for termination detection on anonymous networks.

   Why can the Shavit-Francez termination detection algorithm not be carried over to anonymous networks? That is, where do you run into problems? (12 pts)

   Solution: When a tree disappears, its initiator must start a wave, tagged with its ID, to check whether all trees have disappeared. In an anonymous network, this ID must be chosen at random. If two initiators start such a wave concurrently and happen to select the same random ID, these waves would collide and one (or both) of these waves could complete successfully without having covered the entire network, meaning there may still be active processes in the network. Then termination would be announced prematurely.

5. Explain how a logical clock could be used to make the Walter-Welch-Vaidya mutual exclusion algorithm for MANETs operate correctly if edges are not FIFO. (14 pts)

Solution: The problem with non-FIFO edges is that a node can receive and take into account outdated height information from a neighbor that was overtaken by later height updates. Thi could for instance lead to a cycle in the sink tree.

The vector clock, which can be computed at run-time also in a dynamic network, can be used to spot that height information is outdated as follows. Let each message contain the clock value of its send event.

Nodes keep track for each node p_i from which they received at least one height update, what is the highest ith index of clock values of height updates received from p_i. If a height update from p_i is received with a smaller ith index as clock value, then it is ignored.

When the old root sends the token to the new root, the old root ignores height updates from the new root until it receives a height update with a clock value larger than the one it included in the token.

Note: The last part of this solution would not work properly with Lamport's clock.

6. Consider a distributed transaction with one coordinator and three cohorts. Give two computations of the two-phase commit protocol in which crashed processes must resume their execution before agreement can be reached on whether the transaction commits, one in which all participants vote **yes** and one in which one cohort votes **no**. Also show how these two computations could proceed in the case of the three-phase commit protocol. (14 pts)

Solution: In the first computation, all participants vote **yes**. The coordinator sends **commit** to one of the cohorts, after which the coordinator crashes. This cohort receives the **commit**, commits the changes it made during the transaction, and also crashes. In the second computation, one cohort votes **no** and the other participants vote **yes**. The coordinator sends **abort** to the cohort that voted **no**, after which the coordinator crashes. This cohort receives the **abort**, rolls back the changes it made during the transaction, and also crashes.

In the first computation, in the three-phase commit protocol, the coordinator does not send **commit** but **precommit** to the cohort. So when the coordinator and the cohort crash, the other cohorts can safely abort the transaction, since they did not reply to a **precommit** from the coordinator, so that they can be

certain no participant committed the changes it made during its transaction. Likewise, in the second computation, the cohorts that did not crash can safely abort the transaction.

7. Consider the Winternitz signature scheme with $k = 11$ and $\ell = 3$. Let 10010101100 be the hash of Alice's message to Bob. Explain how Alice signs her message, taking into account the checksum, and how Bob verifies this signature. (14 pts)

   Solution: $k = 11$ and $\ell = 3$, so $n = 4$.

   One 0 is padded at the left of the hash 10010101100 of Alice's message. The 4 binary substrings of length 3 that constitute the resulting string, i.e., 010, 010, 101 and 100, are binary representations of the numbers 2, 2, 5 and 4, respectively.

   Alice computes as checksum $(7-2)+(7-2)+(7-5)+(7-4) = 15$, which has as binary representation 1111. Two 0's are padded at the left to make the length of this string divisible by 3. The 2 binary substrings of length 3 that constitute the resulting string, i.e., 001 and 111, are binary representations of the numbers 1 and 7, respectively.

   Alice generates a private key of 6 random numbers $X_1 \,\|\, X_2 \,\|\, X_3 \,\|\, X_4 \,\|\, X_5 \,\|\, X_6$ and publishes the corresponding public key $h(h^7(X_1) \,\|\, h^7(X_2) \,\|\, h^7(X_3) \,\|\, h^7(X_4) \,\|\, h^7(X_5) \,\|\, h^7(X_6)$. She signs her message with $h^2(X_1) \,\|\, h^2(X_2) \,\|\, h^5(X_3) \,\|\, h^4(X_4) \,\|\, h(X_5) \,\|\, h^7(X_6)$.

   To verify the signature, Bob computes the hash of the message and the checksum, thus determining the sequence of numbers $2, 2, 5, 4, 1, 7$. He applies h^5 to $h^2(X_1)$, h^5 to $h^2(X_2)$, h^2 to $h^5(X_3)$, h^3 to $h^4(X_4)$, h^6 to $h(X_5)$, and nothing to $h^7(X_6)$. Finally, he applies h to the concatenation of the 6 resulting strings and checks that the outcome coincides with Alice's public key.