Online Resit Exam Distributed Algorithms

Vrije Universiteit Amsterdam, 30 June 2021, 18:45-21:30

I declare to understand that taking an online exam during this corona crisis is an emergency measure to prevent study delays as much as possible. I know that fraud control will be tightened and realize that a special appeal is being made to trust my integrity. With this statement, I promise to make this exam completely on my own, only consult those sources that are allowed explicitly, not share my solutions with other students, and make myself available for any oral clarifications regarding this exam.

You can write your solutions with pen and paper. You are allowed to open the pdf's of the textbook and slides (only) at the following links. You are advised to open them in different tabs in your browser.

- `https://canvas.vu.nl/courses/53186/files/3745199`
- `https://canvas.vu.nl/courses/53186/files/3745089`

(The 7 exercises in this exam sum up to 90 points; each student gets 10 points bonus.)

1. Explain how the values of the vector clock can be computed at run-time in case of *synchronous* message passing communication. (14 pts)

   Solution:

   * Let a be an internal event at a process p_i, with $(k_0, \ldots, k_{N-1})$ the clock value of the previous event at p_i (take $(0, \ldots, 0)$ if there is no such previous event). Then $VC(a) = (k_0, \ldots, k_i + 1, \ldots, k_{N-1})$.
   * Let a be a send event at p_i and b is the corresponding receive event at p_j, with $(k_0, \ldots, k_{N-1})$ the clock value of the previous event at p_i with $(\ell_0, \ldots, \ell_{N-1})$ the clock value of the previous event at p_j. Then $VC(a) = VC(b) =$

   $$(\max\{k_0, \ell_0\}, \ldots, k_i + 1, \ldots, \ell_j + 1, \ldots, \max\{k_{N-1}, \ell_{N-1}\}).$$

   In an implementation of synchronous communication, the sending party must wait with sending a message until the receiving party is ready to receive it. So the sending party needs to be informed on the state of the receiving party. Thus we can assume in the second case that it knows the value of $(\ell_0, \ldots, \ell_{N-1})$.

2. Consider the requirement that processes that never crash are never suspected. Argue that this requirement is weaker than strong accuracy and stronger that eventual strong accuracy. (14 pts)

Solution: Strong accuracy clearly implies the requirement, because if only crashed processes are ever suspected, then processes that never crash are never suspected.

To show that the requirement does not imply strong accuracy, give a computation in which a process p in $\mathit{Crash}(F)$ is suspected before it has crashed. To be more precise, p is in $H(q,\tau)$ for some other process q and some time τ, p isn't in $F(\tau)$, and p is in $F(\tau')$ for some $\tau' > \tau$.

The requirement clearly implies eventual strong accuracy. For each execution, from some point on, no processes will crash anymore. The requirement means that from that point on there will be no more false suspicions.

To show that eventual strong accuracy does not imply the requirement, give a computation in which some process that never crashes is suspected by another process at some moment in time (and eventually there are no false suspicions anymore).

3. Propose an adaptation of the weight-throwing termination detection algorithm that works for decentralized basic algorithms. (12 pts)

   Solution 1: Each initiator of the basic algorithm initially holds a certain amount of weight. Weight is distributed through the system in the same way as in the weight-throwing termination detection algorithm for centralized basic algorithms, but weights originating from different initiators are kept separately. Some weight from some (or more) initiator(s) must be attached to each basic message. When a process becomes passive, it returns the weights it holds to the initiators from which they originate.

   When an initiator has becomes passive and reclaimed all weight it held at the start, it starts a wave. Only passive noninitiators and passive initiators that have reclaimed all weight they held at the start take part in this wave. If the wave completes at an initiator, it calls *Announce*.

   Solution 2: There is still a single initiator of the weight-throwing termination detection algorithm, which starts the computation by distributing weights to all initiators of the basic algorithm. Weight is distributed through the system in the same way as in the weight-throwing termination detection algorithm for centralized basic algorithms. Passive processes return their weight to the initiator of the control algorithm. When the initiator of the control algorithm has become passive and reclaimed all weight in the system, it calls *Announce*.

   Note: Underflow can be dealt with in the same ways as for the weight-throwing termination detection algorithm for centralized basic algorithms.

4. Consider the Agrawal-El Abbadi algorithm with $N = 2^k - 1$ processes. Let fewer than

k processes have crashed. Argue, by induction on the depth of the complete binary tree, that the remaining network still contains a quorum. (14 pts)

Solution: The base case, where the binary tree is a single node, is trivial, because then $N = k = 1$, so there is only one process, which has not crashed and forms a quorum.

In the inductive case we distinguish two cases.

Case 1: Let the process at the root of the binary tree have crashed. Since fewer than $k - 1$ of the other processes have crashed, by induction both subtrees below the root, which have depth $k - 1$, contain a quorum of live processes. The union of two such quorums, one in each of the subtrees, forms a quorum for the entire network.

Case 2: Let the process at the root of the binary tree not have crashed. Since fewer than k of the other processes have crashed, in at least one of the two subtrees of the root fewer than $k - 1$ processes have crashed. By induction that subtree, which has depth $k - 1$, contains a quorum of live processes. The union of that quorum with the root forms a quorum for the entire network.

5. In the two-phase commit protocol, let only the coordinator crash, right between the voting and the completion phases. Why can the cohorts safely abort the transaction? You may assume there is a known upper bound on network latency. (14 pts)

   Solution: In case the coordinator decided to commit the transaction, it would send the **commit** messages before making visible its writes during the transaction. The cohorts can, after the last **yes** vote was sent to the coordinator, wait for twice the upper bound on network latency whether a **commit** message arrives. After this waiting period, they can agree together that no **commit** messages were sent to them, and so no writes of the transaction were made visible by any of the participants in the transaction. They can roll back to the old values before the transaction.

6. Consider the Chord ring depicted in Example 18.1 of the textbook. Suppose a peer joins the ring at ID 45. Explain in detail how this peer computes its initial finger table. (12 pts)

   Solution: This peer q copies the finger table of its successor s at ID 48: $\mathit{finger}_s[1] = \mathit{finger}_s[2] = 51$, $\mathit{finger}_s[3] = \mathit{finger}_s[4] = 56$, $\mathit{finger}_s[5] = 1$, and $\mathit{finger}_s[6] = 14$.

   q concludes that $\mathit{finger}_q[1] = \mathit{finger}_q[2] = 48$ and $\mathit{finger}_q[3] = 51$. Furthermore, q looks up its three remaining *finger* values using the finger table of s. To determine $\mathit{finger}_q[4]$, q searches for the ID $45 + 2^3 = 53$. It contacts the peer at ID 51, who locates ID 45 at the peer at ID 56. So $\mathit{finger}_q[4] = 56$. To determine $\mathit{finger}_q[5]$, q searches for the ID

$45 + 2^4 = 61$. It contacts the peer at ID 56, who locates ID 61 at the peer at ID 1. So $\mathit{finger}_q[5] = 1$. To determine $\mathit{finger}_q[6]$, q searches for the ID $45 + 2^5 = 77 = 13 \bmod 64$. It contacts the peer at ID 1. That peer in turn contacts the peer at its 3rd finger value ID 8, who locates ID 13 at the peer at ID 14. So $\mathit{finger}_q[6] = 14$.

7. Let Alice and Bob build a private key using the Diffie-Hellman protocol, with $p = 13$ and d the smallest positive integer that is a primitive root modulo 13. Moreover, let $a = 5$ and $b = 4$. Explain how Alice and Bob construct their private key. (10 pts)

   Solution: The smallest primitive root modulo 13 is 2, because its respective powers modulo 13 are: $2, 4, 8, 3, 6, 12, 11, 9, 5, 10, 7, 1$.

   Alice computes $2^5 = 32 = 6 \bmod 13$ and sends 6 to Bob. Bob computes $2^4 = 16 = 3 \bmod 13$ and sends 3 to Alice.

   Alice computes $3^5 = 9 \cdot 27 = 9 \bmod 13$ and Bob computes $6^4 = 36^2 = 10^2 \bmod 13 = 9 \bmod 13$. So their private key is 9.