Online Exam Distributed Algorithms

Vrije Universiteit Amsterdam, 26 May 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 6 exercises in this exam sum up to 90 points; each student gets 10 points bonus.)

1. Propose an adaptation of the Lai-Yang snapshot algorithm in which basic messages may be buffered at the receiving processes and the channel states of the snapshot are always empty. (14 pts)

   Solution: Processes that want to initiate a snapshot send a control message into each outgoing channel, informing the process at the other side how many presnapshot basic messages were sent into it. Furthermore, they start appending *true* to the basic messages they send.

   When a noninitiator receives a control message or a basic message with *true* for the first time, it also sends a control message into each outgoing channel, informing the process at the other side how many presnapshot basic messages were sent into it, and starts appending *true* to the basic messages it sends.

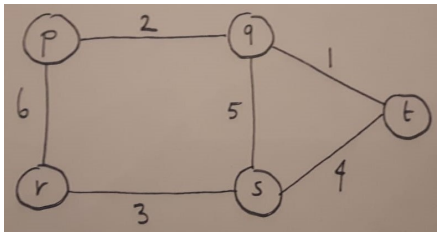
   Postsnaphot basic messages, with *true* attached, that are received through an incoming channel are buffered until the receiver has taken its local snapshot.

   When a process has received all presnapshot basic messages through all its incoming channels, it takes a local snapshot.

2. Let Rana's algorithm be applied to an always terminating basic algorithm. Suppose a process at some point sends a basic message. Argue that there is a computation in which only this process calls *Announce*. (14 pts)

Solution: Let process p send a basic message. Suppose the acknowledgment that is sent in reply is extremely slow, so that by the time it finally reaches p, the basic algorithm has terminated (so in particular p is passive) and all waves started by other processes that became quiet have died out. Since p refuses to participate in any of these waves, as p was not quiet until the acknowledgment arrived, none of these waves ended with a decide event, so no process yet called *Announce*. The last concurrent waves initiated by processes that became quiet must have visited p, as it is the only process refusing to participate in these waves. They pushed p's logical clock beyond the last time any other process became quiet. So when finally the acknowledgment arrives at p, making it quiet, the wave initiated by p will complete and p will call *Announce*.

3. Give one possible computation of the Gallager-Humblet-Spira algorithm on the undirected weighted network below to determine a minimum spanning tree.



During the computation, the handling of the second **test** message from r to p should be delayed at p before it is rejected. (14 pts)

Solution: q and t change channel qt from basic to branch and send $\langle \textbf{connect}, 0\rangle$ to each other. Moreover, p changes channel pq from basic to branch and sends $\langle \textbf{connect}, 0\rangle$ to q. Moreover, r and s change channel rs from basic to branch and send $\langle \textbf{connect}, 0\rangle$ to each other.

q and t receive each other's **connect** messages and merge into one fragment with name qt and level 1 by sending $\langle \textbf{initiate}, qt, 1, \textit{find}\rangle$ to each other. Next, q receives p's **connect** message and sends $\langle \textbf{initiate}, qt, 1, \textit{find}\rangle$ to p. Moreover, r and s receive each other's **connect** messages and merge into one fragment with name rs and level 1 by sending $\langle \textbf{initiate}, rs, 1, \textit{find}\rangle$ to each other.

q and t receive each other's **initiate** messages, make each other their parent, update their fragment name and level, and both send $\langle \textbf{test}, qt, 1\rangle$ to s. Moreover, r and s receive each other's **initiate** messages, make each other their parent, update their fragment name and level, and send $\langle \textbf{test}, rs, 1\rangle$ to p and t, respectively. Moreover, p receives q's **initiate** message, makes q its parent, updates its fragment name and level, and sends $\langle \textbf{test}, qt, 1\rangle$ to r.

p, r, s (two times) and t receive the **test** messages from r, p, q, t and s, respectively, and reply with **accept**.

r and s receive the **accept** from p and t, respectively, and send $\langle \textbf{report}, 6\rangle$ and $\langle \textbf{report}, 4\rangle$ to each other. As a result, s sends $\langle \textbf{connect}, 1\rangle$ to t, which postpones replying to this message because both the **connect** message and t are at level 1. Moreover, p, q and t receive the **accept** from r, s and s, respectively. Next, p sends $\langle \textbf{report}, 6\rangle$ to q, which sends $\langle \textbf{report}, 5\rangle$ to t, while t sends $\langle \textbf{report}, 4\rangle$ to q. As a result, t sends $\langle \textbf{connect}, 1\rangle$ to s.

Since s and t have sent $\langle \textbf{connect}, 1\rangle$ to each other, now they send $\langle \textbf{initiate}, st, 2, \mathit{find}\rangle$ to each other; from s this message travels to r, and from t it travels via q to p. Parent value and fragment name and level at these processes are changed accordingly.

r and s send $\langle \textbf{test}, st, 2\rangle$ to p and q. Responses to these messages are delayed at p and q until they have received the **initiate** message. Then they send a **reject** message in reply. Finally, $\langle \textbf{report}, \infty\rangle$ messages flow to the core edge st of the fragment and the computation terminates.

4. Consider the Afek-Kutten-Yung self-stabilizing algorithm for computing a spanning tree in an undirected network.

   (a) Suppose that at the start of the algorithm, all processes declare themselves root. Explain how this allows to simplify the algorithm. (8 pts)

   Solution: Then all false roots are removed from the network straight away. Therefore join requests are no longer needed, and the algorithm is reduced to: (1) a nonroot that detects an inconsistency declares itself root, and (2) a root that has a neighbor with a larger root value makes that neighbor its parent.

   (b) Motivate why the resulting algorithm is not really self-stabilizing, in the sense that it should be able to cope with e.g. arbitrary bit flips at the hardware level. (8 pts)

   Solution: A random error could at some point cause the introduction of a false root at multiple processes in the network concurrently. Then the scenarios that show the need for join requests apply, as without join requests the algorithm may not stabilize with a spanning tree. But the processes would be unaware of this and the false root could survive, because only at the start of an execution the processes declare themselves root.

5. Consider the time stamp approach to distributed transactions. Two transactions T_1 and T_2 run concurrently, with time stamps t_1 and t_2, respectively. Let variable x

initially contain the value €10. T_1 writes the value €30 to x and then reads x, while T_2 reads x and then increases its value by €10.

(a) Explain for all possible interleavings of the events of T_1 and T_2 whether T_1 and T_2 commit or abort. Distinguish two possible cases: $t_1 < t_2$ and $t_2 < t_1$. (8 pts)

Solution: Let $t_1 < t_2$. If the write of T_1 happens after the read of T_2, then T_1 aborts at its write, because T_2 read x and has a larger time stamp than T_1. If the write of T_1 happens before the read of T_2, then the read of T_2 is delayed until T_1 has committed, because T_1 wrote to x and has a smaller time stamp than T_2. In all cases, T_2 commits.

Let $t_2 < t_1$. If the write of T_2 happens after the read of T_1, then T_2 aborts at its write, because T_1 read x and has a larger time stamp than T_2. (If the write of T_2 happens before the read of T_1, the read of T_1 is not delayed, because T_1 wrote to x previously.) In all cases, T_1 commits.

(b) Explain why in the case $t_2 < t_1$, aborts by T_2 are spurious. Propose an optimization of the time stamp approach that avoids these aborts. (8 pts)

Solution: Since T_1 writes to x before reading it, T_1 is oblivious to the write to x performed by T_2.

Aborts of T_2 can be avoided by labeling read operations by a process that wrote to the same variable earlier. Such a read operation does not lead to an abort of a transaction with a smaller time stamp that wants to write to the same variable after this read operation.

6. (a) Consider the Winternitz signature scheme with $k = 10$ and $\ell = 3$. Let 1011000001 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. (8 pts)

Solution: Two 0's are padded at the left of the hash of the message, to make its length divisible by 3. The binary representations of b_1, b_2, b_3 and b_4 are 001, 011, 000 and 001, respectively, so $b_1 = 1$, $b_2 = 3$, $b_3 = 0$ and $b_4 = 1$.

The checksum is $(7-1)+(7-3)+(7-0)+(7-1) = 23$. Its binary representation is 10111. One 0 is padded at the left to make its length divisible by 3. This means the binary representations of b_5 and b_6 are 010 and 111, so $b_5 = 2$ and $b_6 = 7$.

Alice generates large random numbers $X_1, X_2, X_3, X_4, X_5, X_6$.

Her public key is $h(h^7(X_1) \,\|\, h^7(X_2) \,\|\, h^7(X_3) \,\|\, h^7(X_4) \,\|\, h^7(X_5) \,\|\, h^7(X_6))$.

Her signature is $h(X_1) \,\|\, h^3(X_2) \,\|\, X_3 \,\|\, h(X_4) \,\|\, h^2(X_5) \,\|\, h^7(X_6)$.

Bob uses the hash of the message to compute $b_1, \ldots, b_6$.

With regard to Alice's signature, he applies h^6 to $h(X_1)$, h^4 to $h^3(X_2)$, h^7 to X_3, h^6 to $h(X_4)$ and h^5 to $h^2(X_2)$ and leaves $h^7(X_6)$ unchanged. Finally, he applies h to the concatenation of these strings, compares the result to the public key, and accepts Alice's signature.

(b) Suppose the Winternitz signature from (a) is placed in the third leaf of a binary Merkle tree of depth 4 and used by Alice in a Merkle signature of a message to Bob. Explain what the signature looks like and how this signature is employed by Bob to verify whether the public key is genuine. (8 pts)

Solution: Alice's signature takes the form $sig_3 \,\|\, Y_3 \,\|\, h(Y_4) \,\|\, H_{31} \,\|\, H_{22} \,\|\, H_{12}$ with sig_3 the signature and Y_3 the public key from (a). Each H-values is the h-value of the concatenation of the strings in the two children of the corresponding node in the Merkle tree.

Bob applies h to Y_3 to determine the value in the third leaf. Then he consecutively computes $H_{32} = h(h(Y_3) \,\|\, h(Y_4))$, $H_{21} = h(H_{31} \,\|\, H_{32})$, $H_{11} = h(H_{21} \,\|\, H_{22})$, and $H_{01} = h(H_{11} \,\|\, H_{12})$. He compares the computed value of H_{01} with the Merkle root.

After completing the exam, show your solutions to the camera before closing Proctorio.

After closing Proctorio, upload your solutions on Canvas, within 15 minutes.