Online Exam Distributed Algorithms

Vrije Universiteit Amsterdam, 27 May 2020, 8:30-12:00

By participating in this exam, 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,
- not share my solutions with other students, and
- make myself available for any oral explanation of my answers.

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

1. Compute the vector clock values of the send and receive events and the decide event in an execution of the echo algorithm on an undirected ring of three processes, in which a spanning tree of depth 2 is constructed. (12 pts)

   Solution: Consider a ring with processes p, q, r, where p is the initiator. The computation of the echo algorithm is started by p, which sends messages to q and r, with vector times $(1, 0, 0)$ and $(2, 0, 0)$, respectively. The first of these messages is received by q, with vector time $(1, 1, 0)$, and q makes p its parent. Next, q sends a message to r, with vector time $(1, 2, 0)$. To construct a spanning tree of depth 2, we let the message from q arrive at r first (i.e., before the message from p), with vector time $(1, 2, 1)$, so that r makes q its parent. Next, r sends a message to p, with vector value $(1, 2, 2)$. This message is received by p, with vector time $(3, 2, 2)$. Finally, the message from p arrives at r, with vector time $(2, 2, 3)$. As a result, r sends a message to its parent q, with vector time $(2, 2, 4)$. This message is received by q, with vector time $(2, 3, 4)$. As a result, q sends a message to its parent p, with vector time $(2, 4, 4)$. This message is received by p, with vector time $(4, 4, 4)$. The computation is concluded by a decide event by p, with vector time $(5, 4, 4)$.

2. Consider the Merlin-Segall algorithm with topology changes. Suppose a new channel pq becomes operational. How can processes p and q together determine whether this channel is part of a shortest path toward the initiator, and how could they act if this is the case? (10 pts)

   Solution: Processes p and q can send their distance values to each other through the new channel. If $dist_q + weight(pq) < dist_p$ or $dist_p + weight(pq) < dist_q$, then p or q concludes that it now has a shorter distance to the initiator through the new channel.

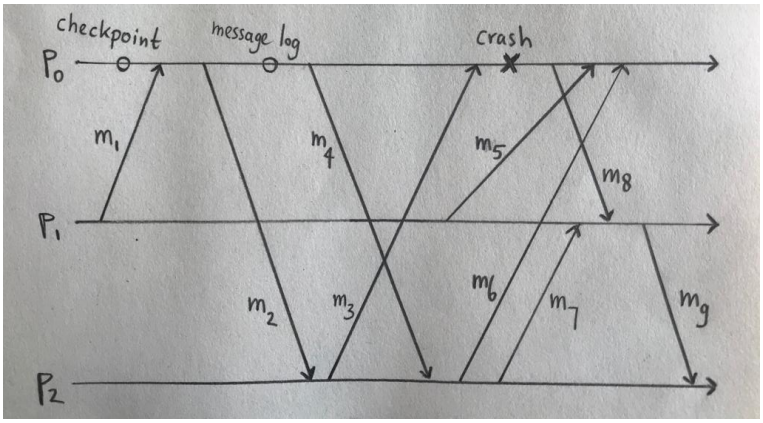
If this the case, p or q can inform the initiator through the sink tree that it should start a new run of the Merlin-Segall algorithm. Alternatively, if e.g. p finds an improved route toward the initiator through the new channel, it can change its distance value to $dist_q + weight(pq)$ and its parent to q, after which it informs its neighbors except q of its improved distance value, which may then lead to a cascade of improved distance values at other processes a la the Chandy-Misra algorithm.

3. Explain where the proof of Theorem 12.1 in the textbook, that there is no (always correctly terminating) algorithm for 1-crash consensus, breaks down in the presence of an eventually weakly accurate failure detector. (16 pts)

Solution: Consider the second case in the proof of theorem 12.1, where the transitions $\gamma \to \gamma_0$ and $\gamma \to \gamma_1$ correspond to events at the same process p. If p is from some point in time on never suspected by the other processes, then the other processes need not be 0-potent or 1-potent to achieve consensus.

For example, consider the Chandra-Toueg consensus algorithm. When the coordinator of a round is never going to be suspected, all processes are forced to wait for the coordinator's message in that round. So the processes without the coordinator are from that point on not 0- or 1-potent.

4. The picture below shows the time line of events at three processes p_0, p_1, and p_2 with regard to some basic computation, where real time progresses from left to right.



Explain in detail how the three processes roll back to a consistent configuration in the past using the Peterson-Kearns algorithm. (12 pts)

Solution: Let the sequence number initially be 0; all basic messages carry this number. After p_0 has crashed, it restarts from its last checkpoint with sequence number 1,

replays the receipt of m_1 from its message log, and reconstructs the send of m_2 (without actually sending it). Let the message log have vector time (k_0, k_1, k_2). Then p_0 sends control messages to p_1 and p_2, containing the vector time (k_0, k_1, k_2), its index 0, and sequence number 1. Upon receipt of this message, p_1 and p_2 start a run of the rollback procedure with sequence number 1; they store k_0 paired with index 0 and sequence number 1. No event needs to be rolled back at p_1, but m_5 needs to be resent (with sequence number 0) because the vector time of its send event is not smaller than (k_0, k_1, k_2). Since the original instance of m_5 is received by p_0 after it resumed its execution, it will receive this message twice; the second received instance of m_5 is discarded. Because of the receipt of m_4, the vector time at p_2 carries a value greater than k_0 at index 0. So p_2 restarts at its last checkpoint (not shown in the picture) and reconstructs events. This replay halts right before the receipt of m_4. Message m_3 is resent by p_2 because the vector time of its send event is not smaller than (k_0, k_1, k_2). Since the original receive event of m_3 was lost in the crash, p_0 will include the receipt of this second instance of m_3 during its resumed execution. When m_6 and m_7 reach p_0 and p_1, respectively, they are discarded as orphan messages, because their sequence number is 0, while the vector time of their send event carries a value greater than k_0 at index 0. Note that m_8 is sent by p_0 after it resumed execution, so the receive of m_8 at p_1 and also the send and receive of m_9 at p_1 and p_2, which carry sequence number 1, are not rolled back.

5. In the time stamp ordering approach for transactions, suppose transaction T_1 wants to perform a write on a variable, but finds that another ongoing transaction T_2 that comes later in the serialization order read this same variable. Explain why it would be a bad idea, instead of aborting T_1 immediately, to let T_1 wait to see whether T_2 will maybe abort, in which case T_1 could still perform the write. (12 pts)

   Solution: The transaction T_2 may want to read another variable on which transaction T_1 has already performed a write. Then T_2 is waiting for T_1 to commit or abort. Moreover, T_2 can only commit when T_1 has completed. For both these reasons, letting T_1 wait for T_2 to commit or abort would create a deadlock situation.

6. Sketch how the AODV protocol can be adopted to allow a peer to look for multiple minimum-hop paths to different destinations with the broadcast of a single RREQ message. (16 pts)

   Solution: Let an RREQ arrive at a peer r with hop count h, where r doesn't have an active route to p with a more recent sequence number than sn_p, or with the sequence number sn_p and a distance value $d \leq h$. If r doesn't have an active route to one or more destinations in the RREQ, then it broadcasts the currently received RREQ, with

the hop count increased by 1, including only those destinations in the RREQ to which it doesn't have an active route. If on the other hand r does know an active route to one or more destinations $q_{i_1}, \ldots, q_{i_k}$ in the RREQ, then it answers with an RREP, which contains the IDs of p and $q_{i_1}, \ldots, q_{i_k}$, the overall distance of each route to the k destinations, and the sequence number of each route, originating from $q_{i_1}, \ldots, q_{i_k}$. In particular, if $r = q_i$ for some i, then the RREP is provided with the sequence number of r, which is then increased by 1. Suppose a peer s receives such an RREP. For each destination q_i that in the message carries a more recent sequence number than s's current route to q_i, or the same sequence number and yielding a shorter route to q_i, s updates its routing information to q_i. If s isn't p and updated its routing information by the received RREP, then s forwards this RREP toward p, preserving only those (one or more) destinations q_i's, and the corresponding information (overall distance of the route, sequence number), for which s updated its routing information.

7. Consider the Merkle signature scheme.

   (a) Suppose an attacker manages to store the entire Merkle tree in memory. Explain why this does not seriously jeopardize the corresponding one-time signatures. (7 pts)

   Solution: To build a forged one-time public key that can replace a genuine public key Y_i, the attacker would have to find a value Z that is in line not only with the corresponding one-time signature, but also with the Merkle root, meaning that replacing $h(Y_i)$ by $h(Z)$ in the Merkle tree should give rise to the same Merkle root as in the original Merkle tree. By preimage resistance of the cryptographic hash function h, this is extremely difficult.

   (b) Why is it still not a good idea to publish the entire Merkle tree, relieving Alice from the duty to provide authentication values to Bob in her corresponding signatures? Give two reasons. (5 pts)

   Solution: First and foremost, this would put a huge strain on memory resources, as the size of the Merkle tree grows exponentially with its depth.

   Second, attackers should be provided with as little information as possible that can aid them in forging signatures. So although attackers will have a very hard time abusing information in the Merkle tree, it is still better to publish the values in its nodes piecemeal when needed.