Problem 1: Snapshots

Use this graph of events and messages from the first Problem Set, and label the messages (bottom to top: 1, 2, 3, 4). Suppose a snapshot is started immediately before event E on S2, and then executes the Chandy-Lamport snapshot protocol. Assume that S1 and S3 only take a snapshot as a consequence of receiving a snapshot marker and that all messages between any two nodes are delivered in order.

a) In a sentence, explain why S3 might receive a snapshot marker from S1 before it receives one from S2.

b) List all possible snapshot states that could be seen. Each snapshot state is the last event seen on each node (this could be a send/receive of messages 1-4), plus any messages that are part of the snapshotted channel state. Note multiple snapshot states are possible; you must list all possible snapshots.

For example, a snapshot taken immediately after event A on S1 but prior to sending message 1, could have these two snapshot states.

(A, nil, nil), channel: nil
(A, nil, G), channel: nil
Problem 2: Write-back cache coherence

Client 1 code
1. put(Y, get(X) + 1)
2. put(Y, get(Y) + 1)

Client 2 code
3. put(X, get(Y) + 1)
4. put(X, get(X) + 1)

Suppose two clients access a simple key-value store using the code above, using an MSI write-back protocol with invalidations. Assume that all keys start at 0, that the statements are executed in the listed order (1, 2, 3, 4), and that each client has its own cache which starts out empty. Show the states of both caches, plus the state of the server, including any MSI state, immediately after each statement is executed.

Problem 3: Paxos acceptor state

Consider a deployment of Paxos (from Paxos Made Simple) with three acceptors. State whether each of these is a valid state at the three acceptors, where a state n : (x, y) means the highest-numbered proposal accepted by acceptor n has number x and value y (and nil means the acceptor hasn’t accepted any proposals). If the state is not valid, explain why in one sentence.

Hint: a state is valid if there is some sequence of message deliveries and message drops and node failures that leads to the state, assuming a correct implementation of proposers and acceptors.

(a) 1:nil, 2:nil, 3:nil
(b) 1:(1, A), 2:nil, 3:nil
(c) 1:(1, A), 2:(2, B), 3:nil
(d) 1:(1, A), 2:(2, B), 3:(3, C)

Problem 4: A dubious Paxos execution

Consider another Paxos deployment with acceptors A, B, and C. A and B are also proposers, and there is a distinguished learner L. According to the Paxos paper, a value is chosen when a majority of acceptors accept a proposal with that value, and only a single value is chosen. How does Paxos ensure that the following sequence of events cannot happen? What actually happens, and which value is ultimately chosen?

(a) A proposes sequence number 1, and gets responses from A, B, and C.
(b) A sends accept (1, "foo") messages to A and C and gets responses from both. Because a majority accepted, A tells L that "foo" has been chosen. However, A crashes before sending an accept to B.
(c) B proposes sequence number 2, and gets responses from B and C.
(d) B sends accept (2, "bar") messages to B and C and gets responses from both, so B tells L that "bar" has been chosen.
Problem 5: Paxos liveness

In the absence of a designated proposer, it is possible for Paxos to fail to make progress even if no messages dropped and no nodes fail. Briefly describe how this can happen in a system with two proposers and three acceptors. Be specific about which messages are sent and in what order they are delivered.