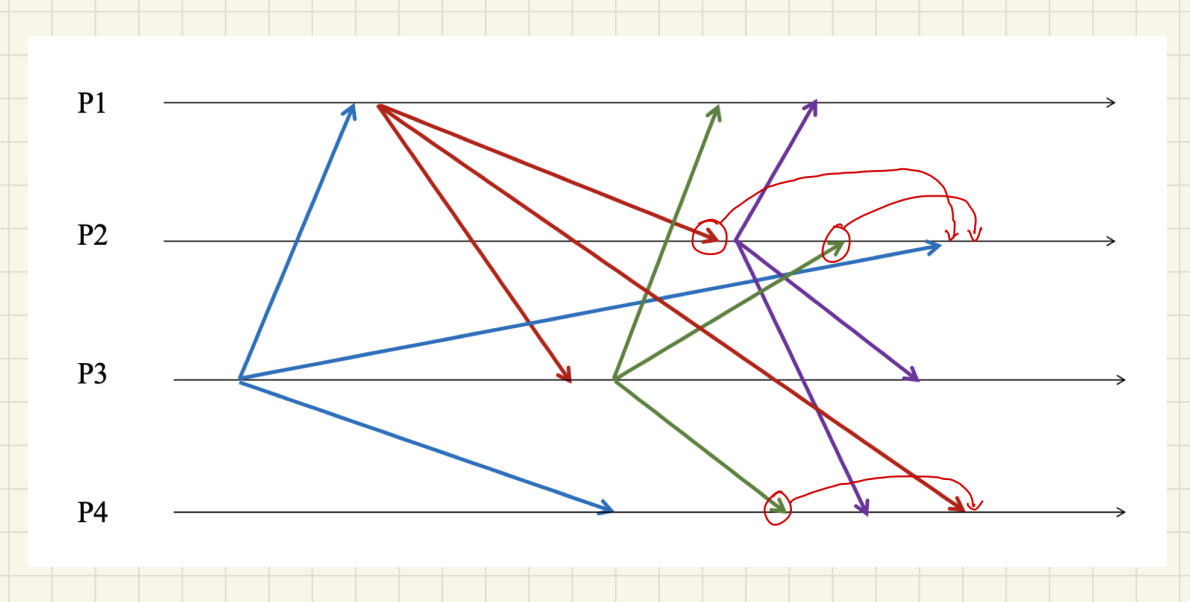
Homework 2

Tags HW

1. Causally Ordered Multicasts



2. True or False:

   a. If processes use `R-multicast`, and each channel follows FIFO order, then causal ordering is satisfied

      This is **true**.

      Suppose that the claim is false, that is, assume that these assumptions do not guarantee causal ordering.

      In other words, suppose there is some pair of messages A and B such that $A \rightarrow B$, but at some process P_i, B is delivered before A.

      $A \rightarrow B$ implies that the process that multicasted B, P_j, delivered A before sending B.

      Since we are using `R-multicasts`, P_j must have retransmitted message A upon receipt of it to P_i.

Then the channel between $P_j \rightarrow P_i$ must contain A before B.

However, we assume that B is delivered before A at P_i, which breaks the assumption that all channels are FIFO-ordered.

Thus, causal ordering is guaranteed under these assumptions.

b. A FIFO and total ordered multicast is also causal ordered, given that processes communicated only through multicast messages.

   This is also **true**.

   Again, suppose that this claim is false, and that these assumptions do not guarantee causal ordering.

   In other words, suppose there is some pair of messages A and B such that $A \rightarrow B$, but at some process P_i, B is delivered before A.

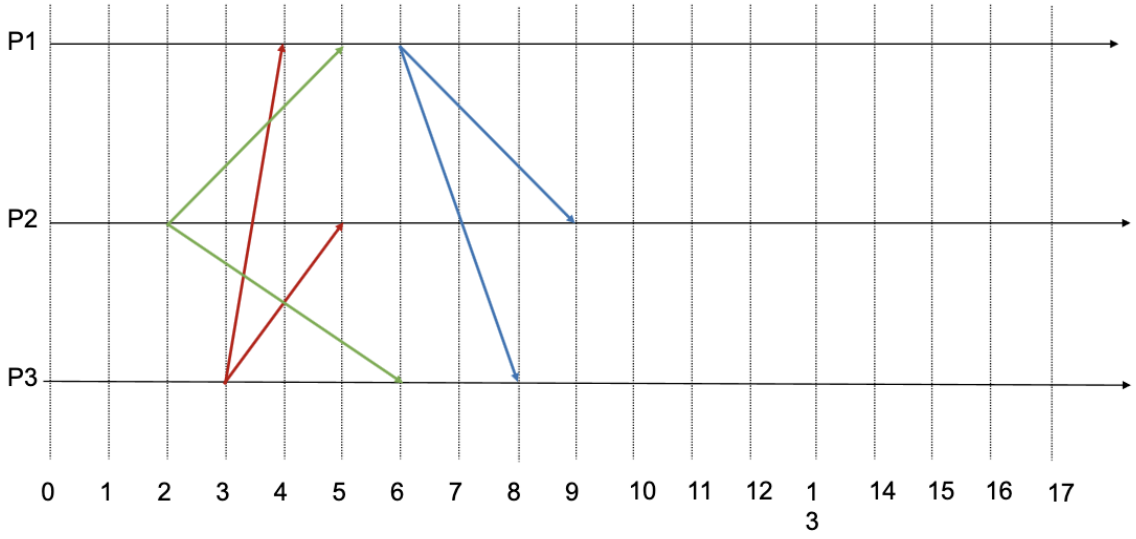
   Since we are in a total-ordered system, B being delivered by A at any single process means that for all k, B is delivered before A at P_k.

   This means that for $A \rightarrow B$ to be true, B must have originated at the same process P_i from which A originated.

   However, since B is delivered *before* A, the agreed-upon sequence number for B must be less than the agreed-upon sequence number for A. This implies that sequence number proposals for B concluded before A (Whether using the sequencer method, or ISIS), which violates the FIFO ordering assumption, as some process must have received a message about B *before* receiving the message about A, even though the message about A was sent *before* the message about B

   Thus, causal ordering is guaranteed.

3. In the following diagram, when will each process enter its critical section?



P2 will enter the critical section first, at time 7.

- Once P2 is finished, at time 10, it sends replies to both P1 and P3's requests.

P3 will enter the critical section second, at time 11.

- Once P3 is finished, at time 14, it will send a reply to P1's request.

P1 will enter the critical section last, at time 15.

4. Consider a run of the Bully algorithm, with 6 processes. Initially all 6 processes are alive and P_6 is the leader. Then P_6 fails and P_3 detects this and initiates the election. Assume no other process fails during the election run, and that there is no processing delay. Also assume that P_3 is the only node that knows that P_6 has failed (and therefore does not send any messages to it). Other nodes are not aware of P_6's failure. One-way transmission time is $T = 10\text{ms}$ (and timeout values set using the knowledge of T).

   a. When will the election finish?

      The election should conclude at time $T = 40\text{ms}$ after P_3 detects the failure and initiates the election.

   b. How many total messages are sent and received by each process in the above scenario?

Since P_1 and P_2 are lower ID-ed proceses and do not participate in the election itself, they only receive one `coordinator` message at the end of the election from P_5.

P_3 sends `election` messages to both P_4 and P_5, receives 2 `disagree` messages from each of the former, and receives a `coordinator` message at the end of the election from P_5

P_4 receives the `election` message from P_3, sends `disagree` to P_3, and sends it's own `election` message to P_5. It will receive a `disagree` followed by a `coordinator` from P_5.

P_5 will receive `election` messages from P_3 and P_4, and reply `disagree` to both. It will send its own `election` messages to P_6, with no reply, and will then conclude the election by sending `coordinator` to P_1, P_2, P_3 and P_4.

In total, if we count multicasts as one message, we have that:

- P_1 receives 1 message
- P_2 receives 1 message
- P_3 sends 1 message and receives 3 messages
- P_4 sends 2 messages and receives 3 messages
- P_5 sends 3 messages and receives 2 messages

And there were a total of 7 messages in the process.

5. Given the proposed consensus algorithm, answer the following questions:

   a. Assume no process fails in the system. When will each process decide on a value and what each of their decided values be?

      Assuming no failures, the R-Multicasts from each process to each other process should be delivered after exactly $T = 20\text{ms}$ has elapsed. Thus, they will all decide on a value at that time, and will choose $y_i = x_1 = 3$ for all i.

   b. Assume P_1 fails right after unicasting x_1 to P_2 and P_3, but just before it could initiate the unicast of x_1 to any of the other processes. When will each of the

alive processes decide on a value, and what will each of their decided values be?

P_2 and P_3 will receive values from all processes at time $T = 20\text{ms}$, like (a), and will decide upon $y_1 = y_2 = 3$ at this time.

They will retransmit the R-Multicast from the failed P_1, which will arrive at P_4 and P_5 at time $T = 40\text{ms}$, which will decide upon $y_4 = y_5 = 3$ at this time.

c. Assume P_1 fails right after unicasting x_1 to P_2 but just before it could initiate the unicast of x_1 to any of the other processes. P_2 fails right after it has relayed x_1 to P_3 but just before it unicasts it to any other process. When will each of the alive processes decide on a value, and what will each of their decided values be?

   P_2 will receive x_1 from P_1 at $T = 20\text{ms}$, and P_3 will receive x_1 from P_2 at $T = 40\text{ms}$. P_3 will retransmit such that P_4 and P_5 receive x_1 at $T = 60\text{ms}$.

   Thus, P_3 will decide on $y_3 = x_1 = 3$ at $T = 40\text{ms}$, but P_4 and P_5 will timeout waiting for x_1, and so will decide upon $y_4 = y_5 = x_5 = 5$ at time $T = 50\text{ms}$.

d. Assume P_5 fails right before it could unicast x_5 to any process. P_1 fails right after unicasting x_1 to P_2 but just before it could initiate the unicast of x_1 to any of the other processes. P_2 fails right after it has relayed x_1 to P_3 but just before it unicasts it to any other process. When will each of the remaining alive processes decide on a value and what will each of their decided values be?

   Identically to part (b), P_2 will receive x_1 from P_1 at $T = 20\text{ms}$, and P_3 will receive x_1 from P_2 at $T = 40\text{ms}$. P_3 will retransmit such that P_4 receives x_1 at $T = 60\text{ms}$.

   Thus, P_3 will decide upon $y_3 = x_1 = 3$ at $T = 50\text{ms}$, and P_4 will decide upon $y_4 = x_3 = 6$ at $T = 50\text{ms}$. Both of these processes timed out waiting for P_5.

e. What is the smallest value that the timeout should be set to for ensuring safety in this system?

   The timeout should be set to the maximum time it could take an R-Multicast to reach one node, if it was transmitted to any alive node.

The worst case occurs when P_1 fails after unicasting to P_2, which fails after unicasting to P_3, which fails after unicasting to P_4. In total, P_5 may only receive x_1 after 80ms, so we should set the timeout to 80ms for this system.

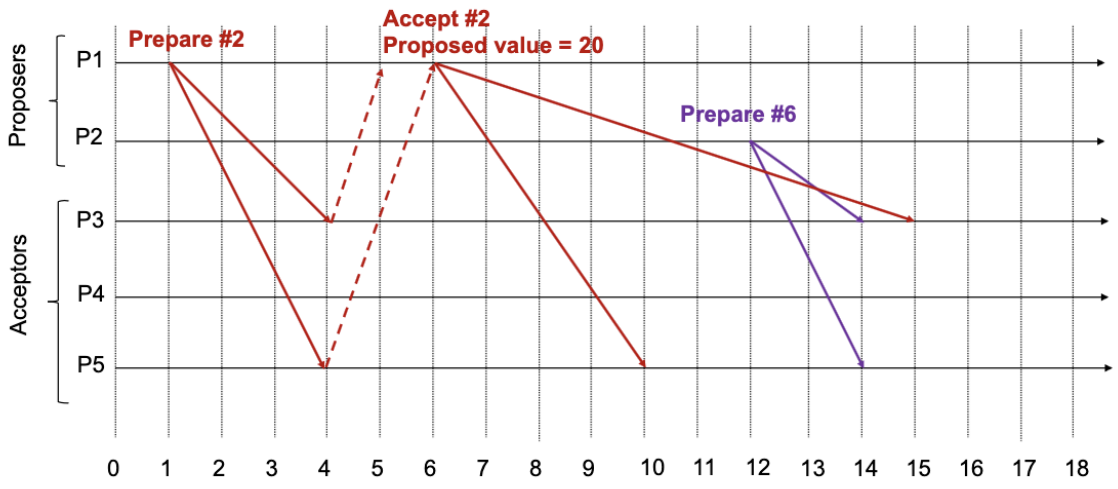
f. Answer (c) assuming the timeout is updated from (e)

Given the updated timeout, we see that P_3 decides upon $y_3 = x_1 = 3$ at $T = 40\text{ms}$ like before, but P_4 and P_5 now wait until receiving x_1 from P_3 at $T = 60\text{ms}$ before deciding upon $y_4 = y_5 = x_1 = 3$ then.

g. Answer (d) assuming the timeout is updated from (e)

Given the updated timeout, we see that P_3 and P_4 both decide upon $y_3 = x_1 = 3$ at $T = 80\text{ms}$. This is because they continue waiting to see if they will receive a multicast from P_5 until the timeout.

6. Consider the Paxos system below:



a. Which processes will accept P_1's proposal?

Only P_5 (and P_1 itself) will accept the proposal.

This is because that by the time P_3 receives `accept`, it has already responded to proposal #6, and cannot accept the lower numbered proposal.

b. Which processes will reply back to P_2's `prepare` message?

   Both P_3 and P_5 will reply to P_2's prepare message, as this is the highest numbered proposal either of them has seen.

c. Will P_2 send out an `accept` message for its proposal #6? If yes, what will be the corresponding proposed value?

   Yes, P_6 will have received a majority of replies so it will send an `accept`. P_5's reply will have accepted value $v = 20$, so the proposed value from P_2's proposal #6 will have $v = 20$

d. Consider the state of the system at time 11 units. Has the proposed value 20 been implicitly decided upon at this point? If yes, explain why? If not, construct a possible scenario that may occur after time 11 units where the system ends up deciding on a different proposed value. The scenario would involve a temporal sequence of events that may differ from the one shown in the figure beyond time t=11 units but must comply with what is shown until t=11 units. An event in such a sequence may include a prepare/accept request sent by a proposer or received by an acceptor, or a prepare reply received by the proposer at some time unit.

   No, the proposed value is not implicitly decided upon at this point, as it has not yet been accepted by a majority of acceptors.

   Consider the situation where P_2 proposes #6 at time 11 to P_3 and P_4. It may receive replies from both of them at time 13, and can send `accept` with some $v' \neq 20$, for example, $v' = 25$. If this `accept` arrives at P_3 before time 15, it will be accepted before $v = 20$, and will become the majority decided value.

e. Suppose that P_1's `accept` request reaches P_3 at time 9 instead of 15. If we now consider the state of the system at time 11 units, has the proposed value of 20 been implicitly decided upon?

   Yes. A majority of acceptors now have accepted 20, so it is not possible for a new proposal to receive enough replies to propose a new value, only to accept with the same value.

f. Suppose that P_1's `accept` reaches P_5 at time 16 and reaches P_3 at the original time 15. Will P_2 send out an accept message for its proposal #6? If yes, what will be the corresponding proposed value?

   Yes, P_2 will send an `accept` because it will receive a reply from both P_3 and P_5. Since neither reply will contain a previously accepted value, P_2 is free to propose any value, and since it intends to propose 25, that will be the corresponding proposed value.