The algorithm would fail with a given condition, and if each process sends their value only to a subset before failing. Consider the following Counterexample for $N = 6$ processes,

Consider that 5 rounds occur (there being 4 crashes), each process sends the value $v_i$ where $v_1$ being the minimum:
Each process sent their minimum value received from all other processes.

**At the end of Round 1:**
P1 send $v_1$ (apart from all the other values it receives) to P2, and fails immediately

**At the end of Round 2:**
P2 send $v_1$ to P3, and fails immediately

**At the end of Round 3:**
P3 sends no messages as discussed in the question.

**At the end of Round 4:**
P4 sends no messages as discussed in the question, and then fails immediately.

**At the end of Round 5:**
P5 receives no messages, sends it appropriate value $v_5$ and fails.

However, since only P3 has $v_1$ and it can't send any more messages, thus the remaining process P6 will have a different value (i.e. $v_5$) and hence the system has not reached consensus.

Note: We also accepted other reasonable solutions, as long as a concrete counter-example was given by the student.
2. (Solved and Graded by: Xiaojuan.)

D: Deliver; B: Buffer. A message without D or B is a sending event.
3. (Solved and Graded by: Kshitij.)

D(Mk) - Message Mk
Delivered

B(Mk) - Message Mk
Buffered
4. (Solved and Graded by: Fangqi Han.)
This question is graded based on the answer’s justification. No credit will be given to a simple “yes” or “no” answer without sufficient explanation.

a. (4 points)

Liveness requires the algorithm to terminate. It does NOT require the algorithm to come to consensus. Answers that confuse liveness with safety will lose points.

For all variants i, ii, and iii, the answer is no. Changing quorum to size L does not impact liveness guarantee of Paxos, which only promises eventual liveness since it may never converge.

Alternatively, “yes” is also acceptable as long as the answer specifies “eventual liveness”.

Note that points will be deducted for any mention of “not eventually live” unless the answer states “under majority failure”.

b.(3 points)

To receive credit for this part where the variant is not safe, the answer needs to explicitly explain how a split decision may be made. Note that “having two possible leaders” and “no longer requiring a majority to pass some decision at phase x” are not sufficient conditions for proving safety violations. You are expected to further explain how these conditions will break safety.

For variant i, both following answers are accepted:

(1) Variant i is not safe as the value accepted by a majority during the Bill phase in a previous round may not be sent to a new round’s leader during its election, which results in inconsistent values being decided in different rounds. (This is what the simplified version of Paxos covered in the lecture will do.)

(2) Variant i is still safe. Despite having the risk of electing two leaders, one of the leaders’ proposed values will not go through the Bill phase. (Which is possible with extra checks during the Bill phase.)

Variants ii and iii are not safe. In variant ii, values decided in different rounds may differ, which can violate consensus given that rounds are asynchronous. To receive credit, the answer must explain why there may be multiple proposals despite the election phase requiring a quorum. In variant iii, in addition to the pitfall of variant ii, safety can be further broken by two leaders deciding on two different values in the same round.

c.(3 points)
Variants i and iii may run faster due to requiring fewer replies or slower due to a second leader having to wait for its timeout and start a new round. Both answers are acceptable. For variant ii, the second scenario does not exist and it will typically be faster due to requiring fewer replies in the Bill phase (note, however, this is not safe).
5. (Solved and Graded by: Samarth.)

**Process States**
- NHX: S(f)
- Earth: S(a)
- Moon: S(e)

**Channel States**
- NHX->Earth: c
- NHX->Moon: f
- Earth->NHX: NULL
- Earth->Moon: NULL
- Moon->NHX: e
- Moon->Earth: b
I. (4 points) A simplified version of an algorithm will be as follows: route to machine 0 and then route to predecessor to find the machine with the highest DHT ID, let’s call this machine N. Then have the first predecessor of N be elected as the leader. Spread this information to everyone.

II. (2 points) Safety: At most 2 processes are neighbors of N, and they get to decide who is the closest. At most one is elected.
Liveness: Since there are no failures and messages are eventually delivered, all Chord messages are eventually routed, one leader is elected, and everyone knows about it.

III. (2 points) Completion: O(N), messages: O(N). (Explanation: O(log N) Chord routing messages to elect a leader, O(N) Chord routing or regular messages to inform everyone of the new leader.)

IV. (2 points) If there are failures, and finger tables are inconsistent, Chord routing may not be correct, and the two nodes deciding may not be the actual successor and predecessor of N. Safety might be violated. As messages may loop around forever, Liveness may be violated.
7. (Solved and Graded by: Adit Bhagat.)
   We will first make the following assumptions:
   1. Synchronous system
   2. At least k non-faulty nodes in the cluster
   3. All nodes know each others' IDs

**K-Leader Election Bully Algorithm:**

1. If a node detects a failure among any of the current k leaders, it will send out an ELECTION message to all other nodes.
2. Upon receiving an ELECTION message, a node will send an OK message to all nodes with a higher ID than it, and then wait a period of time to receive any OK messages from other nodes.
3. All nodes that received at least k OK messages will begin waiting for k COORDINATOR messages to be received.
4. Any node that received less than k OK messages will elect itself as a leader, and send COORDINATOR messages to all other nodes.
5. Any node that does not receive k COORDINATOR messages will initiate a new election run after a timeout period (start over from step 1).

**Safety:** Since the OK messages are only sent to nodes with higher IDs, we guarantee that there will only be k nodes that receive less than k OK messages. In other words, there will always be exactly k nodes that are chosen at the end of an election run, which therefore satisfies safety.

**Liveness:** Based on the assumptions made, we know that messages in the system will be received within a known time bound. If we design the timeout periods for receiving OK and COORDINATOR messages around this time bound, we can guarantee that the k leaders will eventually be elected. Failures during the election process are essentially handled by step 5.

Note: The above algorithm essentially always elects k leaders, even if only one of the previous leaders failed. It’s possible to modify this algorithm such that single or few failures of the k leaders can be handled, instead of always re-electing all k leaders (i.e. replace the failed leaders rather than re-elect most of the same leaders again). This would likely require additional message types and/or piggybacked information, however.
8. (Solved and Graded by: Pengyu Lu.)

a. (4 points) Safety: Safety is violated. Considering two events e1 (the first event on process P1) and e2 (the first event on process P2) that do not have a causal relationship, both with Lamport timestamps of 1. The request sent by both processes would be <1, 1> (<Lamport time, Local sequence number>). According to the algorithm, a process that sets its state as wanted and receives a request that has a later timestamp would queue the request received. Otherwise the request is granted by the process. In the scenario above, both processes would reply to each other's request without queueing. Therefore both P1 and P2 would be able to enter the critical sections at the same time. Safety is violated.

b. (3 points) Liveness: Liveness is not violated. For a request with request number pair <T, S> from process P to not receive access to the critical section, another process P' must have queued <T, S> and reply to a request <T', S'> first where either (1) T' < T or (2) T' = T and S < S'. However, based on the modified assumption of the algorithm, the request number pair received by each process will only monotonically increase and therefore, P' will eventually receive <T'', S''> so that <T'', S''> > <T, S> based on the modified timestamp. Therefore, request <T, S> will eventually collect replies from all processes and receive access to the critical section and liveness will not be violated.

c. (4 points) Causal ordering: Causal ordering is not violated. For events e1, e2 that have causal ordering of e1 -> e2, request timestamp from e1, T1 is always less than request timestamp e2, T2 (T1 < T2), based on the definition of Lamport timestamp. The request number pair <T1, S1>, <T2, S2> must satisfy <T1, S1> < <T2, S2> based on definition of the RA algorithm. Therefore, a request based on e1 is always satisfied before a request based on e2 has been granted access by all processes. Therefore causal ordering is preserved.
9. (Solved and Graded by: Matt Hokinson.)
   b. (1.5 points) Correct. Every non-faulty process received the same set of multicasts and everything in the view stayed in the view.
   c. (1.5 points) Incorrect. Again, M32 is sent in V11 and delivered by some processes (p1). Then, p2 delivers M32 in V12. Correction: both p1 and p2 deliver M32 in V11 when it is sent, then deliver V12.
   d. (1.5 points) Incorrect: Since p1 and p2 both fail to deliver their own multicasts, not all non-faulty processes are receiving the same multicasts. Correction: both p1 and p2 deliver their own multicasts in V11.
   e. (1.5 points) Incorrect: The view V12 does not reflect all non-faulty processes, and p3 should not deliver a view that does not contain itself. Correction: p3 discards the view V12.
   f. (1.5 points) Correct: All the remaining processes have received the same multicasts, and no process presents a view which doesn't contain itself.
   g. (1.5 points) Incorrect: M32 is delivered in V12, not V11. Correction: processes 1-3 should deliver multicast M32 in V11. Then, p4 joins and all processes deliver V12 (with all four processes).
10. (Solution and Graded by: Tomoko Sakurayama.)
   a. (5 points) An incoming channel state should be recorded as soon as it receives a
      marker (instead of waiting for the last marker). The current algorithm waits too
      long to capture the channel state, causing possible capture of unnecessary
      messages, which might create an inconsistent cut.

   b. (5 points) Two bugs needs to be fixed:
      i. In the last line of the “if(this is the first marker Pi is seeing)” conditional:
         Pi:Cji (for j = 1 to N except i) replace it by => Pi:Cji (for j = 1 to N except i
         and the *incoming* channel)
      ii. In the else block: remove existing lines and add: Mark the state of the
         *incoming channel* as all the messages that have arrived on it since
         recording was turned on for that *incoming channel*.
   c. (0 points) “One small step for a human, a giant leap for…Doraemon and Einstein,
      come back here right now!” (Any quote is acceptable)

======= END OF HW3 SOLUTION =======