Midterm B Solutions: CS425 FA25

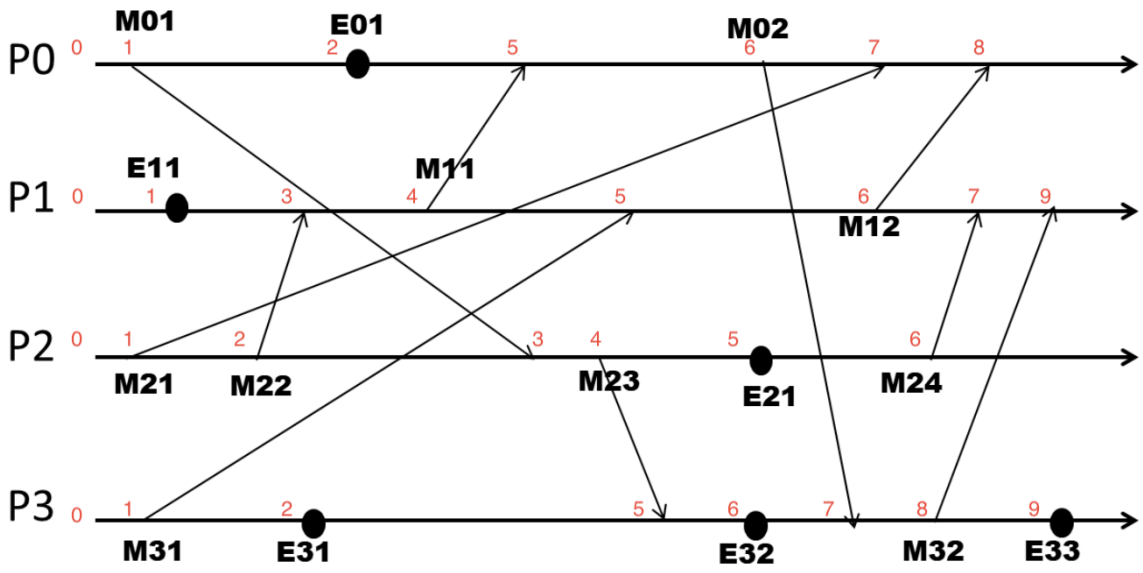
Note to Students: Recommended solutions for Midterm B (FA25) are below. For many questions, alternate solutions are possible and reasonable correct solutions will be accepted during grading. So please refrain from asking questions about solutions until you receive your exam grades back.

1. (Solution and Grading by: Ashish Kashinath.)

1. (c)
2. (c)
3. (c) or (d) [we now accept option (d) as one of the correct choices for MCQ3]
4. (b)
   Reason:
   - False Positive Rate decreases as it is less likely that two heartbeats are lost with the heartbeat interval is changed 10 → 5s,
   - Bandwidth increases since the heartbeat bytes per second is doubled when the internal is changed 10 → 5s),
   - Worst-case failure detection time is still unchanged at 20 secs, as a node still waits 20 secs before declaring failure,

5. (b) (reason: even if there is Lamport timestamp ordering L(e1) < L(e2), V(e1) and V(e2) can be concurrent)
6. Ceil((3N+1)/4) (reason : If k is size of quorum then 4(N-k) < N will ensure at least one process belongs to each and every quorum)
7. Concurrent
8. 3 (reason: only available on the seed, rarest block first strategy of Bittorrent)
9. (q, 38, 90) (reason: unchanged, since the received heartbeat is older)
10. S31 (reason: it is sharing rack with replica on machine S32)

2. (Solution and Grading by: Kartik).
   A.



   B.
   Concurrent and have different Lamport timestamps. There might be a lot of answers for this:
   E01 (P0, 2) and E11 (P1, 1)
   M01 (P0, 1) and M11 (P1, 4)
   M01 (P0, 1) and M22(P2, 2)
   M01 (P0, 1) and E31 (P3, 2)
   E01 (P0, 2) and M31 (P3, 1)

   C.
   E33 would include all the events that happened on P3. The answer is 7.

3. (Solution and Grading by: Lilia.)

a)

2048 points on ring → 2^m = 2048 → m = 11
(n + 2^i) mod 2^m, n = 34
PEER 34 FINGER TABLE

i	ft[i]
0	89
1	89
2	89
3	89
4	89
5	89
6	169
7	169
8	433
9	610
10	1325

-1 for each incorrect finger table entry
1 point for attempting the question

b)

We are looking for K200, which is stored at Peer 233 (first peer with id >= key)
SOLUTION: 34 → 169 → 194 → 233

Per the question, we start at Peer 34
First, look at the finger table for Peer 34 and hop to Peer 169 (largest entry <= 200):
PEER 169 FINGER TABLE for i=0..5

i	ft[i]
0	194
1	194
2	194

3	194
4	194
5	233

Hop to Peer 194 (largest entry <= 200)
PEER 194 FINGER TABLE for i=0

i	ft[i]
0	233

Want to send a query to the largest finger entry <= 200. None exists, so send to the successor, which is Peer 233. Found 200.

-2 for each incorrect hop without correct explanation
-1 instead if the incorrect hop has careless error/cascading from part a but correct explanation
-2 for each extra hop
-1 including 200 as a node
1 point for attempting the question
No additional points off for cascading errors from part b

c) 433, 610, 985, 1325

-1 for each missing node
-1 for each extra nodes
1 for having at least one correct node

4. (Solution and Grading by: Talha and Tianchen.)

   Q4 MapReduce: For the 3rd condition, this solution uses: "Total number of posts made by X on Mondays > 100".
   [for the older condition based on average: goto Original MR condition 3 not used in current midtermB]

   MR1: Handles D1
   M1(key=null, value=(a, b)):
   output(a, b)
   output(b, a)
   R1(key=user, value=friend_array):
   if @Boss in friend_array and |friend_array| >= 1M:
   output(user, "CANDIDATE")

   MR2: Handles D2
   M2(key=null, value=(a, p, wd:hh:mm:ss)):
   if wd == "Mon":
   output(a, 1)
   R2(key=user, value=[1, …, 1]):
   output(user, sum(value))

   MR3: Aggregate outputs from MR1 and MR2 to find X
   M3(key=user, value="CANDIDATE"|post_count):
   output(user, value) // identity mapping
   R3(key=user, value=[*"CANDIDATE"*, post_count]): // "CANDIDATE" may not exist
   if "CANDIDATE" in value and post_count > 100:
   output(_, user)

5. (Solution by: Nishant Sheikh.)

   a. If nobody selects you randomly, then you are only monitored by your k successors and k predecessors.
      Then, if you + your k successors + your k predecessors go down simultaneously, then no one will detect that you failed.
      Thus, completeness is violated for M = 2k + 1.
   b. No. To a heartbeating protocol running on an asynchronous system, a dropped heartbeat (e.g. due to lossy network) and a real machine failure both lead to a timeout. As such, it is always possible for message drops to lead to false positives, and accuracy cannot be 100%.
   c. Three cases: worst, best, and average.

      Worst case:
      Everyone has you as a target (as successor, predecessor, or random).
      N-1 processes monitor you.
      Load: N-1 heartbeats/period

      Best case:
      Smallest monitoring set - nobody randomly selected you, so only your k successors and k predecessors monitor you.
      k + k = 2k processes monitor you.
      Load: 2k heartbeats/period

      Average case:
      (elegant approach)
      The average number of heartbeats sent is, by definition, the average number of heartbeats received. By the problem statement (k predecessors + k successors + k distant nodes), this is 3k heartbeats/period.

      (probability approach)
      First, your k successors and k predecessors monitor you no matter what, since you're in their predecessor/successor range.
      That leaves D = N - 1 - 2k distant nodes that can randomly select you.
      Each one of these distant nodes can pick k nodes out of their random selection pool, size R = N - 1 - 2k; N - (self) - (k successors + k predecessors).
      P[selected] = 1 - P[not selected].
      P[not selected] = P[not selected in round 1] * ... * P[not selected in round k]
      P[not selected] = (R-1)/R * (R-2)/(R-1) * (R-3)/(R-2) * ... * (R-k)/(R-k+1)
      Cancelling terms, we get P[not selected] = (R-k)/R.
      Plugging back in, we get P[selected] = 1 - (R-k)/R = k/R.
      So for a given distant node, there is a k/R chance you will be in its random pool.

There are D distant nodes, so the expected number of distant nodes that monitor you is D*(k/R) = (N - 1 - 2k) * (k / (N - 1 - 2k)) = k.
Putting this all together, you can expect 3k nodes to monitor you on average: k successors, k predecessors, and k distant nodes.
Load: 3k heartbeats/period

Q4 (MapReduce): Appendix

Original MR condition 3 not used in current midtermB.

These solutions use: "•X makes more posts than an average user on Mondays (i.e., number of posts made by X > average of the number of posts made by all users on Mondays)."

Solution by Aishwarya.

```
MR1: Handles D1
M1(key=null, value=(a, b)):
   output(a, b)
   output(b, a)
R1(key=user, value=friend_array):
   if @Boss in friend_array and |friend_array| >= 1M:
      output(user, "CANDIDATE1") // candidates who satisfy cond1 and cond2

MR2: Handles D2
M2(key=null, value=(a, p, wd:hh:mm:ss)):
   if wd == "Mon":
      output(a, 1)
R2(key=user, value=[1, …, 1]):
   output(user, sum(value))

MR3: Find users who post than the average user on Mondays
//read output of R2.
M3(key=user, value=post_count):
   output("ALL", (user, post_count))

// pass all user, post-count pairs to one reducer to find average and output users who satisfy cond3
R3(key="ALL", value=[(user,post_count), …]):
    counts = list of all post_counts from value array, consisting (user, post_count) tuples
    average_post_on_monday = sum(counts)/len(counts)
    for (user, post_count) in value:
        If post_count > average_post_on_monday:
            output(user, "CANDIDATE2") // candidates who satisfy cond3

MR4: Join outputs from MR1 and MR3 to find candidates who satisfy all three

M4(key=user, value="CANDIDATE1"|"CANDIDATE2"):
   output(user, value) // identity mapping
R4(key=user, value=["CANDIDATE1", CANDIDATE2"]): //  may be empty
```

```
if “CANDIDATE1” in value and “CANDIDATE2” in value:
    output(_, user)
```