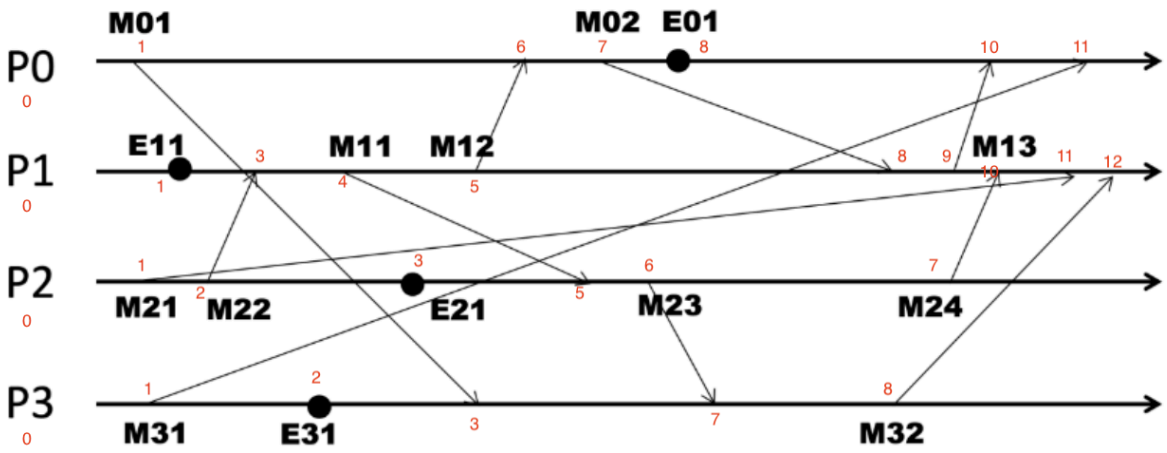
# Midterm A Solutions: CS425 FA25

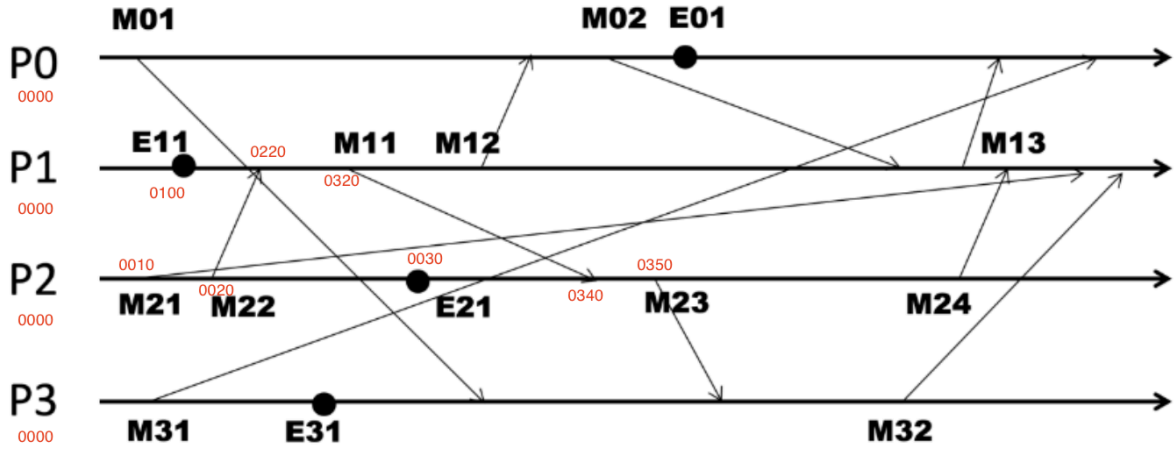
*Note to Students: Recommended solutions for Midterm A (FA25) are below. For many questions, alternate solutions are possible and reasonable correct solutions will be accepted during grading. So please refrain from asking questions about solutions until you receive your exam grades back.*

1. (Solution and Grading by: Gabriella Xue.)
    1) C
        - PUE = Total Facility Energy / IT Equipment Energy = 9000 / (9000-3000) = 1.5
    2) E
        - BitTorrent downloads Local Rarest First. For availability counts, block 4 is the least replicated among neighbors.
    3) C
        - A strictly higher Lamport timestamp doesn't imply causality. Events can be concurrent with different Lamport values. In that case, vector clocks are incomparable.
    4) B
        - Gossip period refers to how often each process sends gossip messages.
        - Gossip period does not affect the detection time, since the failure detection timeout is still the same.
        - But it affects the number of gossip messages per second, which affects bandwidth usage and false positive rate.
    5) D
        - Both a and b claims "any client C", but for C100, it uses a consistency level of ONE for read, therefore, C100 can return stale data if it hits a lagging replica.
    6) Concurrent.
        - For vector clocks, e1->e2 if and only if every component of V(e1) $\leq$ V(e2) and at least one is <.
    7) 1 minute.
        - MTTF = total operating hours / number of servers
        - new_MTTF = total operating hours / (2 * number of servers) = $\frac{1}{2}$MTTF = 1
    8) O(log(n))
        - TTL $\geq$longest shortest path in the network. Chord's max shortest path length is O(log(n)).
    9) (q, 95, 99)
        - Incoming heartbeat count is smaller than the current one, so no update.
    10) S21
        - Rack-local to replicas on S22 and S23.

2. (Solution and Grading by: Madhav Jivrajani.)




Part b: third element of vector clock of M23: 5 (includes all events that occurred on P2), visualizing it below (you don't need to calculate the vector timestamps):



3. (Solution and Grading by: Hanbo Guo.)
   a.

| i | ft[i] |
|---|---|
| 0 | 1597 |
| 1 | 1597 |
| 2 | 1597 |
| 3 | 1597 |
| 4 | 1597 |
| 5 | 1597 |
| 6 | 1597 |
| 7 | 1597 |
| 8 | 1597 |
| 9 | 1597 |
| 10 | 1 |

   b. File 15 is stored at the machine with ID 21.
   Finger table entry for node 1597:

| i | ft[i] |
|---|---|
| 0 | 1 |
| 1 | 1 |
| 2 | 1 |
| 3 | 1 |
| 4 | 1 |
| 5 | 1 |
| 6 | 1 |
| 7 | 1 |
| 8 | 1 |
| 9 | 89 |
| 10 | (not important) |

Finger table entry for node 1:

| i | ft[i] |
|---|---|
| 0 | 3 |
| 1 | 3 |
| 2 | 5 |
| 3 | 13 |
| 4 | 21 |
| 5 | (not important) |

So the hops from machine 89 to 15 are:

i. 89 -> 1597, reason: 1597 is 89's largest finger table entry <= 15.
ii. 1597 -> 1, reason: 1 is 1597's largest finger table entry <= 15.
iii. 1 -> 13, reason: 13 is 1's largest finger table entry <= 15.
iv. 13 -> 21, reason: 13 has no finger table entry <= 15, so send to the successor 21.
v. 21 has the file 15! Routing completed.

c. 987 only.

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

```
M1(key=null, value=(a, b)): // output link in both directions
        output(a, b)
        output(b, a)

R1(key=a, value=friend_array): // aggregate friend list of a
        for b in friend_array:
                output((a, b), friend_array)

M2(key=(a, b), value=friend_array):
        // map friend_array of both a and b to the same reducer
        output(lex_sorted(a, b), (a, friend_array))

R2(key=lex_sorted(a, b), value=V): // V = [(a, a_friend_array), (b, b_friend_array)]
        if @Boss not in a_friend_array or @Boss not in b_friend_array:
                return // output nothing
        If @Company not in a_friend_array and @Company not in b_friend_array:
                return // output nothing
        If @Company in a_friend_array and @Company in b_friend_array:
                return // output nothing
        if a in {@Boss, @Company} or b in {@Boss, @Company}:
                return // output nothing
        output(_, (a, b))
```

5. (Solution and Grading 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