1. (Graded by: Tomoko Sakurayama, Indy, and several TAs)
   Sequence of answers: f b b d a c d b b e

1.1. **Answer: f**
   SRM uses NAKs
   RMTP uses ACKs
   Gossip does not use ACK/NAKs to avoid the overhead.

1.2. **Answer: b**
   The new heartbeat’s counter (89) is older than the current counter (95), hence the entry is not updated.

1.3. **Answer: b**
   Mapreduce attempts to schedule a map task on 1. a machine that contains a replica of corresponding input data, or failing that, 2. on the same rack as a machine containing the input, or failing that, 3. Anywhere

   The only machine on the same rack as a replica is S21.

1.4. **Answer: d**
   A failure of a process must be detected by a non-faulty process. A faulty process detecting a failure, especially when the detector fails right afterwards, is incomplete. (A common question was: can failed processes recover? You should know from the lecture, where we repeated many times, that the default failure model in this class is fail-stop.)

1.5. **Answer: a**
   Let L(e) be the lamport timestamp of e. Let V(e) be the vector timestamp of e. Additionally, let V(e1) = V(e2) mean V(e1) is equal or incomparable to V(e1). We can now prove each of the statements to be correct or incorrect.
   a. We know L(e1) < L(e2). This means either e1 → e2 or e || e2. So, either V(e1) < V(e2) or V(e1) = V(e2). So, the statement is false.
   b. We know V(e1) < V(e2). So, e1 → e2. So, L(e1) < L(e2). As desired.
   c. We know V(e1) is equal to V(e2). WLOG, suppose e1 is on p1 and e2 is on p2. From the vector timestamp algorithm, V(e1)[p1] is equivalent to L(e1), and V(e2)[p2] is equivalent to L(e2). Since V(e1)[p1] is equal to V(e2)[p2], L(e1) = L(e2). As desired.
   d. We know L(e1) = L(e2). So e1 || e2. So, V(e1) = V(e2). As desired.

1.6. **Answer: c**
   Remember the rules of comparing two Vector timestamps. VectorTS(e2) < VectorTS(e1).

1.7. **Answer: d**
Because C100 is using Consistency Level ONE for reads, it may not receive the latest written value by itself or any other client written with CL QUORUM.

1.8. **Answer: b**
MTTF = mean time to failure. The more machines, the more frequent the failures.

1.9. **Answer: b**
Increasing gossip frequency (i.e., lowering gossip period) increases bandwidth, and reduces false positive rate (latter is because it increases reliability of gossips due to increased gossip “fanout”). Detection time does not change. Worst case detection time occurs when the process fails right after sending out its heartbeat. This detection time depends only on timeouts (Tfail, Tcleanup) and not the gossip frequency.

1.10. **Answer: e**
a. Only outputs within each Map task are sorted. They are not sorted across the Map tasks, so it is necessary to sort again to manage the inputs to Reduce.
b. Each Reduce task’s input is sorted via mergesort.
c. Reduce cannot happen until all Maps are finished. The unfinished Maps may have outputs to use in the prematurely started Reduce task. This is again due to the need to sort the outputs across the Map tasks as stated in choice a.
2. (Graded by: Kshitij Phulare)
   a) The Last element of the Vector Timestamp assigned to M44 would be 8, because there are 8 events at P4, and vector timestamps rules say that the last entry (its own entry) at P4 is incremented by +1, whenever any event occurs at P4 (regardless of whether the event is a send, receive, or instruction).
3. (Graded by: Fangqi Han)

a. (-1 for each incorrect/missing entry)

In a Chord P2P system with 512 points, we can compute \( m=9 \). \((2^9=512)\)

The finger table entries of machine with peer ID 21 should therefore include the first peer whose id is greater than or equal to \((21+2^i)\), where \( i \) ranges from 0 to 8:

<table>
<thead>
<tr>
<th>( i )</th>
<th>ID Requirement</th>
<th>Finger Table Entry</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>( \geq 21+2^0=22 )</td>
<td>34</td>
</tr>
<tr>
<td>1</td>
<td>( \geq 21+2^1=23 )</td>
<td>34</td>
</tr>
<tr>
<td>2</td>
<td>( \geq 21+2^2=25 )</td>
<td>34</td>
</tr>
<tr>
<td>3</td>
<td>( \geq 21+2^3=29 )</td>
<td>34</td>
</tr>
<tr>
<td>4</td>
<td>( \geq 21+2^4=37 )</td>
<td>55</td>
</tr>
<tr>
<td>5</td>
<td>( \geq 21+2^5=53 )</td>
<td>55</td>
</tr>
<tr>
<td>6</td>
<td>( \geq 21+2^6=85 )</td>
<td>89</td>
</tr>
<tr>
<td>7</td>
<td>( \geq 21+2^7=149 )</td>
<td>233</td>
</tr>
<tr>
<td>8</td>
<td>( \geq 21+2^8=277 )</td>
<td>377</td>
</tr>
</tbody>
</table>

b. (+2 for each correct hop; -1 for each case where the student copied a node number wrong)

In Chord, files are stored at first peer with ID greater than or equal to their keys, so file 15 is stored at node 21. Starting from node 233, we send query to its largest successor/finger entry \( \leq 15 \), which in this case is node 0. Node 0 then finds peer ID 8 in its own finger table and sends query to it. (Note that node 13 is not in node 0’s finger
Node 8 then hops to node 13, and node 13 sends the query to its successor node 21 due to all its successor and finger table entries being larger than 15. At this point node 21 can locate the key in its own storage. The hops are 233->0->8->13->21.

c. (+1 for each correct node; extra +1 for all 3 correct nodes; -1 for each incorrect node)

The nodes needing to change finger table entries must at least have some finger table entry that is larger than 377. They must therefore at least satisfy node_id > 121 (node_id + 2^8 > 377).

Nodes that meet this requirement are 144, 233, and 377. You can verify that they indeed need to change some of their finger table entries. (e.g. i=8 for 144 and 233, i=0 for 377.)
4. (Graded by: Taksh Soni and Adit Bhagat)

Key idea: MR1 below checks criteria (i-iii) in the question. MR2 calculates the most popular user (first half of calculation for criterion (iv)). MR3 checks if the MR1-outputted users satisfy condition (iv) (second half of calculation for criterion (iv)).

Other correct solutions were also accepted.
A common error was students re-using data across multiple Reduce tasks within the same stage -- you should know from lecture (where we repeated this often) that Reduce tasks are independent and there is no inter-Reduce communication or “global variables” among Reduce tasks!

General Rubric:
-2 for each condition (i-iv) that is not satisfied
-5 for any major errors (such as the errors described above)

M1 (a, b): //a follows b
- Emit (key=a, value=(b , empty_set))
- Emit (key=b, value=(empty_set, a))

//value[0] indicates a “following” relationship between the key and value[0]
//value[1] indicates a “follower” relationship between the key and value[1]

R1 (key, value):

followingSet = empty
followerSet = empty

For each v = (a, b) in value:
- followingSet = followingSet.union({a})
- followerSet = followerSet.union({b})

if length(followingSet) < 10
    AND length(followerSet) > 10 million
    AND all items in followingSet are also present in followerSet:
    Emit (key, "valid")

Emit ("NumberOfFollowers", [key, length(followerSet), followerSet])
    // We assume no user exists named "NumberOfFollowers"
M2 (key, value):
// Key = user, value = "valid"
// or key="NumberOfFollowers", value = [user, FollowerCount, followerSet]

Emit (key, value)

R2 (key, value):
// Key = user, value="valid"
// or key = "NumberOfFollowers", value = ([user1, FollowerCount1, followerSet1],
[ user2, FollowerCount2, followerSet2], .....

if(key== "NumberOfFollowers"): // 1 Reduce task to calculate the popular user
    maxFollowerCnt = 0
    maxUser = empty
    followerSet = empty
    for each [a,b, c] in value: //Loop and find the user with maximum followers
        if(b > maxFollower):
            maxFollowerCnt = b
            maxUser = a
            followerSet = followerSet

    Emit(maxUser, ("max", followerSet))
    // We assume no user exists named "max"

else if(value contains "valid"): // Other Reduce tasks are identity
    Emit (key, value)

M3 (key, value):
//key = user, value = ("max", followerSet)/"valid"
if(value contains "max"): 
    for each follower in followerSet:
        Emit(follower, "invalid")

Emit (key, value)

R3 (key, value):
//key=user, value= ("valid", "invalid" -->(if invalid))
if (value does not contain "invalid"):
    Emit(_, key)
5. (Graded by: Xiaojuan Ma)

(a) All of the processes in an intersecting row/column can detect one another. Since there are M+K-1 unique processes in this ‘cross’, if all but one of these processes were to die, the remaining process would be able to detect the failure. Thus, L=M+K-1, and L-1=M+K-2 is the maximum number of failures allowed for completeness.

(b) M=4. K+M-2>=10. This gives us K+2>=10, K>=8

(c) (Beware! This is a trick question). It is impossible to guarantee accuracy in an asynchronous system with a SWIM/ping based protocol (or a heartbeating protocol), because it is impossible to distinguish between a failed process and a process that is dropping incoming/outgoing messages.

Rubrics for Q5:
sub-problem(a)
-1: if justification is correct but the value has minor issues: say, L should be M+K-1, but they have no -1, or -2.
-2: value is correct, has some justification but may not be enough.
-5: value is correct, (nearly) no justification.
-6: value is incorrect, i.e., M*K, has justification and it is wrong.
-8: value is incorrect, but has M or K or N, i.e., M*K, (nearly) no justification.
-10: value is incorrect and irrelevant, i.e., just a number or some meaningless expression. (nearly) No justification.

sub-problem (b)
-1: if they miss some correct answer. The answer is >=8, but they have =8, or >8
-2: wrong values but partially correct method, i.e, they have <= 8, <=7
-3: more mistakes, small part makes sense
-4: not really makes sense, but tried a lot
-5: nearly no trying and totally wrong

sub-problem(c)
-5: if gives an incorrect number/expression with or without a wrong reason
-3: if some part of the argument makes sense