

Midterm B Solutions: CS425 FA 26

Note to Students: Recommended solutions for Midterm B (FA26) are below. For many questions, alternate solutions are possible and reasonable correct solutions will be accepted during grading.

1. (Solution by: Nishant Verified by: Yaqi)

a. Q1: C

- i. With EC2, you are renting virtual machines, and you are in charge of the OS and software deployed to them. This is closest to IaaS.
- ii. For HaaS, you would be renting the HW itself.
- iii. For SaaS, everything up through the software is managed by the provider.
- iv. For PaaS, the runtime environment is managed by the provider.

b. Q2: C

- i. The CAP theorem says that “During a network partition, a system cannot guarantee both consistency and availability”.

c. Q3: C

- i. A is false: queries are forwarded to all neighbors except the one who forwarded it to you.
- ii. B is false: QueryHit messages are routed along the reverse path (peer who forwarded the query to you). Duplicates are dropped, identified by matching DescriptorID.
- iii. C is true: Ping messages are flooded out like Query messages.
- iv. D is false: the Push message itself doesn't transfer file contents. This is handled via HTTP GET after the Push message establishes the parameters.

d. Q4: E

- i. Completeness is unaffected, since we still have a time bound.
- ii. Accuracy can never be 100% in an asynchronous system where you cannot distinguish between silence from node failure & silence from network complications.
- iii. Bandwidth is unaffected, since the heartbeat period is the same.
- iv. False positive rate improves, since a larger grace period prevents false positives caused by network issues.
- v. Failure detection time is worsened, since a crashed process will go undetected for longer.

- e. Q5: B
 - i. Consider the following case w/ two processes which never communicate:
 - 1. e_1 , on P_1 : $L(e_1) = 1$, $V(e_1) = [1, 0]$
 - 2. e_2 , on P_2 : $L(e_2) = 2$, $V(e_2) = [0, 2]$
 - ii. $L(e_1) < L(e_2)$, but $V(e_1)$ and $V(e_2)$ are concurrent.
- f. Q6: E
 - i. Reducers may draw input from every Map task, so they cannot start until all Map tasks are complete.
- g. Q7: C
 - i. Let's say node K joins between nodes A and Z. Node Z owns the key range between A and Z, so when K joins, only Z's key range is affected.
- h. Q8: C
 - i. The heartbeat counter is higher, so this is new info & we incorporate the new heartbeat counter value. However, our clocks are not necessarily synchronized with those of our peers. So we use our own local timestamp.
- i. Q9: B
 - i. More servers w/ same failure rate $\rightarrow$ shorter expected time before a single server in the cluster fails.
- j. Q10: E
 - i. Priority: (1) machine w/ replica of the input data $\rightarrow$ (2) same-rack neighbor for machine w/ replica of input data $\rightarrow$ (3) anywhere.
 - ii. Neither S32, S41, and S43 are free. So (1) is impossible. But S31 is free, so (2) is possible.

2. (Solution by: Weidong Verified by: Nishant)

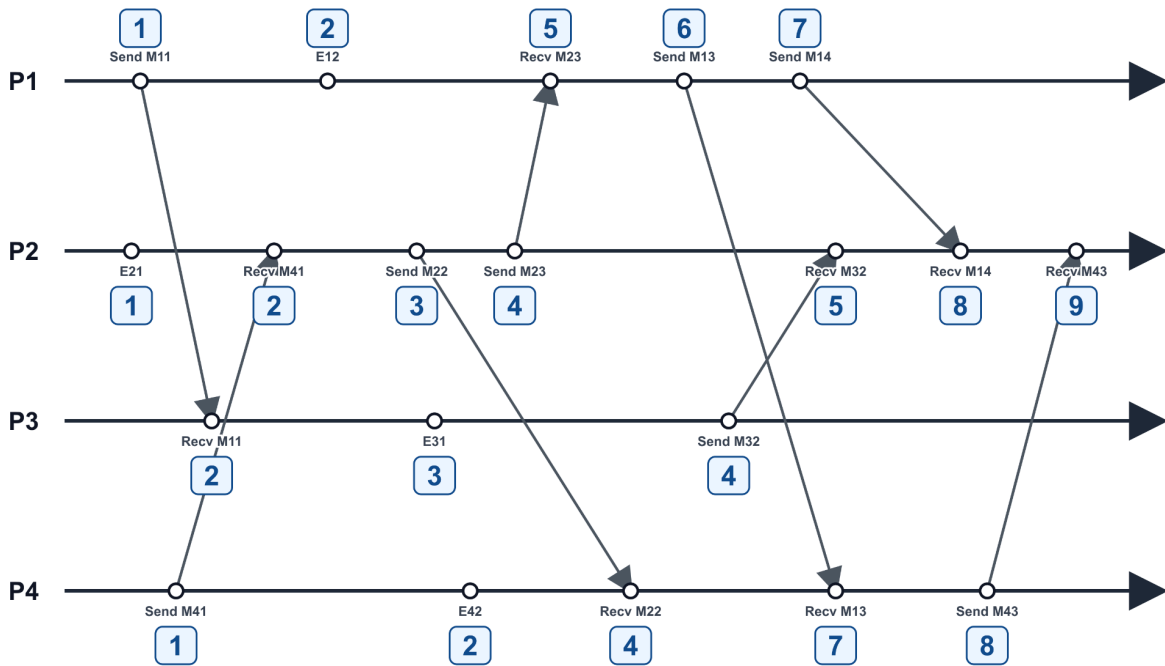
- a. (12') For a local or send event, increment the local Lamport clock by 1. For a receive event, set the clock to $\max(\text{local clock}, \text{received timestamp}) + 1$.

P1: Send(M11) = 1; E12 = 2; Receive(M23) = 5; Send(M13) = 6; Send(M14) = 7.

P2: E21 = 1; Receive(M41) = 2; Send(M22) = 3; Send(M23) = 4; Receive(M32) = 5; Receive(M14) = 8; Receive(M43) = 9.

P3: Receive(M11) = 2; E31 = 3; Send(M32) = 4.

P4: Send(M41) = 1; E42 = 2; Receive(M22) = 4; Receive(M13) = 7; Send(M43) = 8.



- b. (3') One valid pair is E12 and E21, represented as (P1, 2) and (P2, 1). There is no happened-before path in either direction between these two events, so they are concurrent, while their Lamport timestamps are different.

Other valid examples include:

- E12 (P1, 2) and E31 (P3, 3)
- E21 (P2, 1) and E31 (P3, 3)
- Send(M11) (P1, 1) and Receive(M41) (P2, 2)

There are many other correct answers. Any pair should be accepted as long as (1) the two events are concurrent, i.e., neither happens before the other, and (2) their Lamport timestamps are different.

- c. (5') Assuming that vector timestamps are ordered as (P1, P2, P3, P4), the last component of the vector timestamp for Send(M43) is 5. Send(M43) is P4's fifth event: Send(M41), E42, Receive(M22), Receive(M13), and Send(M43). P4 increments its own component once for each of these events.

3. (Solution by: Soham Verified by: Weidong)

(a) Chord

Note that we accepted both intuitive and formal proofs.

The selection of the node that's closest (network distance-wise) to n from the set of peers can essentially be thought of as a random variable. In the worst case, the higher finger picks near its interval's right boundary and overshoots the target, while the preceding finger picks near its left boundary.

Intuitive Proof:

Original Chord guarantees that the distance is halved at each step. But the variant may, by choosing the i -th entry closer to the interval's upper boundary $n+2^{(i+1)}$, potentially overshoot the target. So, we may need to pick the preceding finger, which guarantees only $d/4$ progress (instead of $d/2$). Thus, at each step, the distance between query and peer-with-file becomes at most $3/4$ of its previous value. This means that there is logarithmic reduction of distance to the key at every step, and the routing latency is $O(\log(N))$.

Formal Proof:

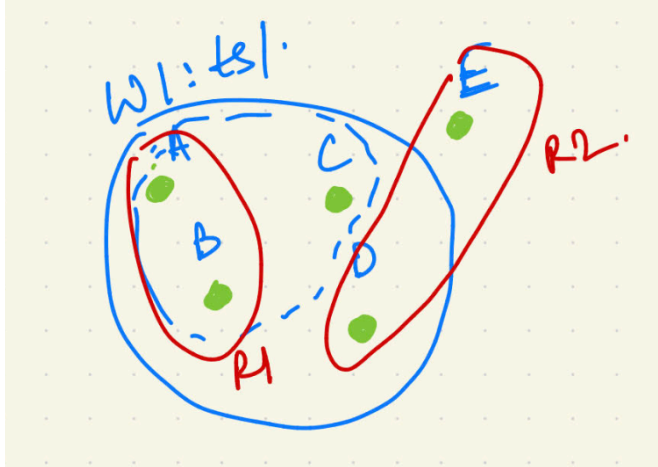
Assume the distance between the current node n and the target key k along the ring is d (in points along the ring). We need to prove that the length of the next hop is at least $1/4 d$.

Proof by contradiction: Let the query be currently at Q . Assume the next hop node H (i.e., farthest hop from Q , but still at or anticlockwise of the key K) is less than $1/4 d$ points distance from Q . Assume the next hop node H is at $ft[i]$ of current node Q 's fingertable. Then we have $Q+2^i \leq H < Q+2^{i+1}$. Since H is less than $1/4 d$ away, we can conclude that $Q+2^{i+2} \leq K$. This means that $ft[i+1]$ will be between H and K , and thus H is not the farthest next-hop from Q that is to the anticlockwise of K (if there is no node in the interval $Q+2^{i+1}$ to $Q+2^{i+2}$, then the fallback selects a node beyond H and at or before K). This is a contradiction to our assumption. Thus, the next hop after Q must be beyond $1/4 d$ distance away from Q towards K . This means the distance becomes at most $3/4$ of the previous value at each hop. Thus, the routing latency is $O(\log_{4/3} N)$ or $O(\log(N))$.

(b) Cassandra

(i) Yes. The last write that completed before the read began, with timestamp t_w , was acknowledged by W replicas. A read gets responses from R replicas. Since $R + W > N$, those two sets must overlap in at least one replica. The read returns the newest timestamp among its R responses, so it returns a version with timestamp at least t_w . Because a single coordinator assigns all the timestamps, they follow the order the writes were issued.

(ii) No. Since multiple reads and writes can be in progress for a key K at the same time, the first read can see a write that is in-progress and has reached fewer than W replicas. The second read might reach a disjoint subset from the first read and be overlapping with replicas that are in the write quorum but haven't yet received the write.



Consider the following example. $N = 5$, $W = 4$, $R = 2$.

Replicas A to E all start with (k_0, v_0) at timestamp 0. Consider the following steps.

1. Write $W1(k_0, v_1)$ at timestamp ts_1 starts and reaches A, B, C: 3 acks, still short of $W = 4$
2. Read $R1$ asks A and B, gets $\{v_1, v_1\}$, returns **v_1** , and completes
3. Read $R2$ starts, asks D and E, gets $\{v_0, v_0\}$, returns **v_0**
4. $W1$ reaches D and completes

Read $R2$ began after $R1$ completed and returned an older version. The figure above should clarify better.

4. (Solution by: Rafid Verified by: Soham)

A.

MR1: Handles D1, finds every pair of people at the same place at the same time

M1(key=null, value=(name, location, start, end)):

output(location, (name, start, end))

R1(key=location, value=V): // V = all visits to this location

for (x, s1, e1) in V:

for (y, s2, e2) in V:

if s1 <= e2 and s2 <= e1: // time periods overlap (e1 == s2 counts)

output(x, ("MET", y)) // x == y is allowed, so positive people are also listed

MR2: Handles D2

M2(key=null, value=name):

output(name, "POSITIVE")

R2(key=name, value=V):

output(name, "POSITIVE")

MR3: Joins outputs of MR1 and MR2

M3(key=x, value=("MET", y)|"POSITIVE"):

output(x, value) // identity mapping

R3(key=x, value=V): // V = everyone x met, plus "POSITIVE" if x tested positive

if "POSITIVE" in V:

for ("MET", y) in V:

output(y, _) // y was at the same place and time as positive person x

MR4: Removes duplicate names

M4(key=y, value=_):

output(y, _) // identity mapping

R4(key=y, value=V):

output(y, _) // each name is output exactly once

We accept solutions that include or exclude self-contact

B.

Advantage: Writing to the local disk is faster and cheaper. In traditional MapReduce, reduce output goes to the distributed file system (HDFS), which keeps 3 replicas on different servers. So the output has to be sent over the network and written three times. A local write skips this, so it uses less network bandwidth and disk space, and the job finishes sooner.

Disadvantage: The output is no longer fault-tolerant. There is only one copy. If that server or its disk fails before the next MapReduce job reads the file, the output is lost, and the reduce task has to be re-run (possibly along with the Map tasks before it). With one copy instead of 3 replicas, the scheduler also has fewer choices for running the next job's Map tasks on a machine or rack that holds their input.

5. (Solution by: Yaqi Verified by: Rafid)

- a. With full-membership gossip, once a process learns fresh heartbeat information, it can forward that information in every round. With a constant fanout b , standard randomized gossip disseminates the information to all processes **in expected $O(\log N)$ rounds[2pt]**. In the modified protocol, however, a particular membership entry is sent by each process **only once every N rounds[2pt]**. Therefore, one effective gossip step for that entry now takes $O(N)$ physical rounds. This slows down dissemination by a factor of N , so the expected dissemination time becomes **$O(N \log N)$ rounds [3pt]**.

Therefore, compared with full-list gossip, fresh heartbeat information propagates much more slowly. T_{fail} must be chosen large enough to account for this $O(N \log N)$ dissemination delay[3pt]. Otherwise, a correct process may be falsely suspected simply because its fresh heartbeat has not reached some processes in time. However, **a larger T_{fail} [1pt]** also means that actual failures take longer to detect, since detection time is at least T_{fail} .

- b. Yes [1pt]

It is no different than gossip-style heartbeating in terms of completeness.

Yes, assuming that correct processes continue running. Once a process fails, it **stops generating newer heartbeat values [2pt]**. Old heartbeat information may continue to circulate for some time, but eventually every correct process will stop receiving fresh heartbeat updates for the failed process. **After T_{fail} expires, the failed process will be suspected by every correct process [2pt]**.

- c. No[2pt]. A correct process may be falsely suspected if its fresh heartbeat information is delayed for longer than T_{fail} [3pt] due to infrequent gossip or network delays.

(or) In an asynchronous distributed system, a failure detector cannot guarantee both completeness and perfect accuracy[3pt].