

Homework 2 Solutions: CS425 FA26

Note to Students: Recommended solutions for HW2 (FA26) are below. For many questions, alternate solutions are possible, and reasonable correct solutions will be accepted during grading.

1. (Solution and Grading by: Parham. Verified by: Weidong)
 - a. $2^{(m-1)} - 2$ (c's subtree, removing c itself) + 1 (root) + 1 (c's sibling) + $2^{(m-3)} - 2$ (sibling's subtree, removing c's sibling itself, and note TTL is m-4 when it reaches the sibling) = $5 * 2^{(m-3)} - 2$
 - b. We're looking for the diameter of the tree, which is the distance between the leftmost and rightmost leaves in our case. Their distance is $2 * (m-1)$, where m-1 is the distance of each of them to the root.
 - c. We still need to find the rightmost leaf distance from the leftmost one. Having the new edges, consider the family of routes "climb from the bottom to level k, walk all the way across level k, descend to the rightmost leaf." This way, the number of edges will be $2(m - k) + 2^{(k-1)} - 1$. Trying different k, we can see that k=2 or k = 3 will give us the minimum answer, which is $2m - 3$.

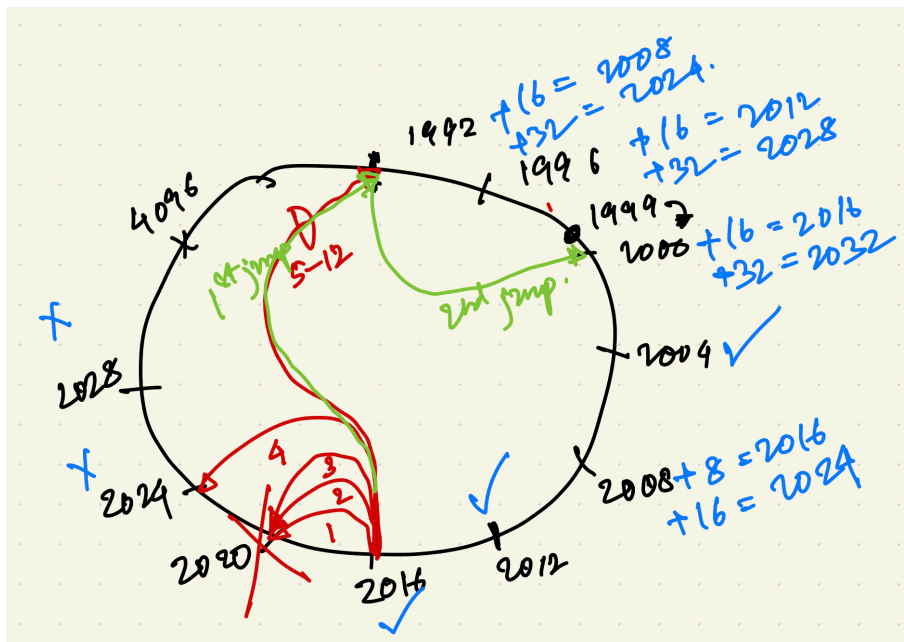
2. (Solution and Grading by: Tianyi. Verified by: Parham)
- a. G will be fetched first, since it is the rarest one.
 - b. We can let 9th planet have only {G}. In this case, the number of G and F will both become 3, and lower letter will be downloaded first, which is F.
 - c. The biggest problem would be that it delays the replication of rare shards. If the peers who have the rarest shards become offline, the complete file could not be downloaded.
 - d. The biggest problem would be that the download speed would be limited by the single peer's upload bandwidth, causing a lower download speed.

3. (Solution and Grading by: Soham. Verified by: Tianyi)

(a) Finger table entries for Node 2012

The i -th finger table entry for node n is given by $n.finger[i].node = \text{successor}((n + 2^i) \bmod 2^{12})$

i	n.finger[i].node
0	2016
1	2016
2	2016
3	2020
4	2028
5	1992
6	1992
7	1992
8	1992
9	1992
10	1992
11	1992



(c) If Node 2020 is dropped, nodes that contain 2020 in their finger table need to update their finger table. The image above shows in blue the i -th and $(i+1)$ -th finger table entry for some nodes. It can be seen that they skip 2020 in their finger table entries. The only nodes that have 2020 in their finger table entries are:

Node **2004**: $\text{successor}(2004 + 2^4) = 2020$

Node **2012**: $\text{successor}(2012 + 2^3) = 2020$

Node **2016**: $\text{successor}(2016 + 2^2) = 2020$

$\text{successor}(2016 + 2^1) = 2020$

$\text{successor}(2016 + 2^0) = 2020$

4. (Solution and Grading by: Will. Verified by: Soham)

a. Part A

- i. k = number of hash functions
- ii. n = size of input
- iii. m = size of Bloom filter in bits
- iv. b = number of Bloom filters
- v. False positive rate: $P_{FP} = (1 - e^{(-kn/m)})^k$
- vi. Bloom Filter B:
 1. $k = 4, m = 1024, b = 1$
 2. $P_{BF} = (1 - e^{(-kn/m)})^k = (1 - e^{(-4n/1024)})^4 = (1 - e^{(-n/256)})^4$
 - a. $N = 1000: (1 - e^{(-1000/256)})^4 = 0.9219 = 92.19\%$
 - b. $N = 10: (1 - e^{(-10/256)})^4 = 2.15 \times 10^{-6} = 0.000215\%$
- vii. Leo Variant A (ALL):
 1. $k = 1, m = 256, b = 4$
 2. $P_{LA} = (1 - e^{(-kn/m)})^{(bk)} = (1 - e^{(-n/256)})^4$
 - a. $N = 1000: (1 - e^{(-1000/256)})^4 = 0.9219 = 92.19\%$
 - b. $N = 10: (1 - e^{(-10/256)})^4 = 2.15 \times 10^{-6} = 0.000215\%$
- viii. Leo Variant B (ANY):
 1. $k = 1, m = 256, b = 4$
 2. For Leo-ANY, the false positive probability is 1 minus the probability that all Bloom filters return 0.
 3. $P(\text{return 0} \mid \text{key not present}) = 1 - FP(\text{one filter})$
 4. $P(\text{all filter returns 0} \mid \text{key not present}) = (1 - FP)^b = (1 - (1 - e^{(-kn/m)})^k)^b = (1 - 1 + e^{(-n/256)})^4 = e^{(-n/64)}$
 5. $P_{LB} = P(\text{at least one filter returns 1} \mid \text{key not present}) = 1 - (1 - FP)^b = 1 - e^{(-n/64)}$
 - a. $N = 1000: 1 - e^{(-1000/64)} = 0.9999998 = 99.99998\%$
 - b. $N = 10: 1 - e^{(-10/64)} = 0.1447 = 14.47\%$
- ix. The Original Bloom filter and Leo-ALL have the same false positive rate. Therefore, for both $N = 10$ and $N = 1000$, the original Bloom filter and Leo-ALL tie for the lowest false positive rate.

b. Part B:

- i. For insertion, all three approaches have about the same performance because they compute 4 hash functions and set 4 bits, so insertion is $O(1)$. For lookup, Original Bloom and Leo-ALL stop lookup at the first 0, whereas Leo-ANY stops at the first 1. Consequently, lookup performance depends on filter occupancy and whether the queried key is present.

5. (Solution and Grading by: Rafid. Verified by: Will)

Solution:

The quorums are chosen arbitrarily, so M must work for every possible choice of Q_1, Q_2, Q_3 , including the worst one. Below, $|S|$ is the number of processes in a set S , and $\lceil x \rceil$ is x rounded up to the nearest integer.

Key fact: For any two sets A and B of processes, $A \cup B$ contains at most N processes. So $|A \cap B| = |A| + |B| - |A \cup B| \geq |A| + |B| - N$. The overlap is smallest when A and B together cover all N processes. The $W + R > N$ condition from lecture rests on the same reasoning.

Warm-up ($k = 2, r = 1$): Two quorums of size M share at least $2M - N$ processes. To share at least one process, we need $2M - N \geq 1$, so $M \geq (N + 1)/2$. This gives $M = \lceil (N + 1)/2 \rceil = \lfloor N/2 \rfloor + 1$, which is a majority. If M were $\lfloor N/2 \rfloor$, we could pick two quorums with no process in common, since $2\lfloor N/2 \rfloor \leq N$. So M cannot be smaller.

(a) Applying the key fact twice:

$$|Q_1 \cap Q_2| \geq 2M - N$$

$$|Q_1 \cap Q_2 \cap Q_3| \geq |Q_1 \cap Q_2| + |Q_3| - N \geq (2M - N) + M - N = 3M - 2N$$

This worst case can happen. Each quorum leaves out $N - M$ processes, and a process is common to all three quorums only if no quorum leaves it out. Suppose we choose the three left-out groups so that they do not overlap. Then exactly $N - 3(N - M) = 3M - 2N$ processes are common. If $3(N - M) > N$, the left-out groups can together cover every process, and then no process is common.

So at least r common processes are guaranteed exactly when $3M - 2N \geq r$, that is, when $M \geq (2N + r)/3$.

Ans: $M = \lceil (2N + r)/3 \rceil$

Example: $N = 10$ and $r = 2$ give $M = \lceil 22/3 \rceil = 8$. With $M = 8$, any three quorums share at least $3(8) - 2(10) = 4 \geq 2$ processes. With $M = 7$, consider these three quorums: all processes except P_1, P_2, P_3 ; all except P_4, P_5, P_6 ; and all except P_7, P_8, P_9 . They share only P_{10} , which is fewer than 2.

(b) Claim: any k quorums of size M share at least $kM - (k - 1)N$ processes. For some choice of quorums, the number shared is exactly $\max(0, kM - (k - 1)N)$.

Proof by induction on k .

Base case ($k = 1$): $|Q_1| = M = 1 \cdot M - 0 \cdot N$.

Inductive step: suppose any j quorums share at least $jM - (j - 1)N$ processes. Then, by the key fact,

$$|Q_1 \cap \dots \cap Q_j \cap Q_{j+1}| \geq (jM - (j - 1)N) + M - N = (j + 1)M - jN.$$

The worst case comes from the same construction as in (a). The k quorums leave out k groups of $N - M$ processes that do not overlap. If $k(N - M) > N$, the groups instead cover all N processes, and no process is common.

So we need $kM - (k - 1)N \geq r$, that is, $M \geq ((k - 1)N + r)/k$.

Ans: $M = \lceil ((k - 1)N + r)/k \rceil$

Check: $k = 2$ and $r = 1$ give $\lceil (N + 1)/2 \rceil = \lfloor N/2 \rfloor + 1$, the majority from the warm-up. Setting $k = 3$ gives the answer to (a).

6. (Solution and Grading by: Yaqi. Verified by: Rafid)

A. Cassandra's full membership and token ownership information allows a coordinator to identify and directly contact the replicas responsible for a key. Replacing this information with Chord finger tables would require multi-hop lookups, typically $O(\log N)$ hops, to locate the responsible node, increasing request latency and network message overhead.

B. (1) SSTables are immutable, so physically deleting records from them would require rewriting affected SSTables, increasing disk I/O and delete latency. (2) Without tombstones, replicas that process a delete retain no timestamped deletion marker. If another replica misses the delete, subsequent repair may copy its stale value back to the replicas that deleted it, resurrecting the deleted record as "zombie data." Tombstones allow repair to propagate the deletion and suppress older values.

C. (1) Bloom filters allow Cassandra to quickly skip SSTables that definitely do not contain a requested key. Without them, Cassandra must examine more candidate SSTables. (2) Additional index lookups and potential disk I/O increase read latency and reduce throughput, especially for absent keys or when many SSTables exist. Reads can remain correct, but become less efficient.

D. (1) It slows down writes, since every write must check if the key already exists, potentially incurring SSTable reads. (2) In-place updates conflict with SSTable immutability. In the original design, SSTables are not modified in place after creation. However, in the proposed design, updating them could require rewriting associated data, indexes, or compressed blocks.

We will accept other reasonable answers, as well as those that focus on the distributed read and write protocols, rather than just the local storage.

7. (Solution and Grading by: Wei. Verified by: Yaqi)

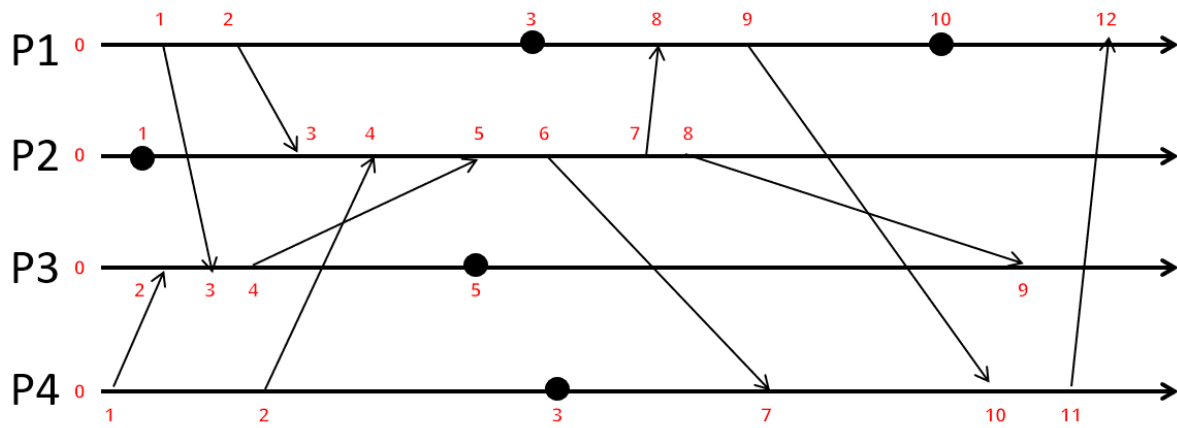
Using: $\text{Error} \leq (\text{RTT} - \text{min1} - \text{min2}) / 2$, where:

- $\text{RTT} = 1.01 \text{ ms} = 1010 \mu\text{s}$
- $\text{min1} = X + 400 \mu\text{s}$: minimum client-to-server delay
- $\text{min2} = 15.7 + 100 = 115.7 \mu\text{s}$: minimum server-to-client delay

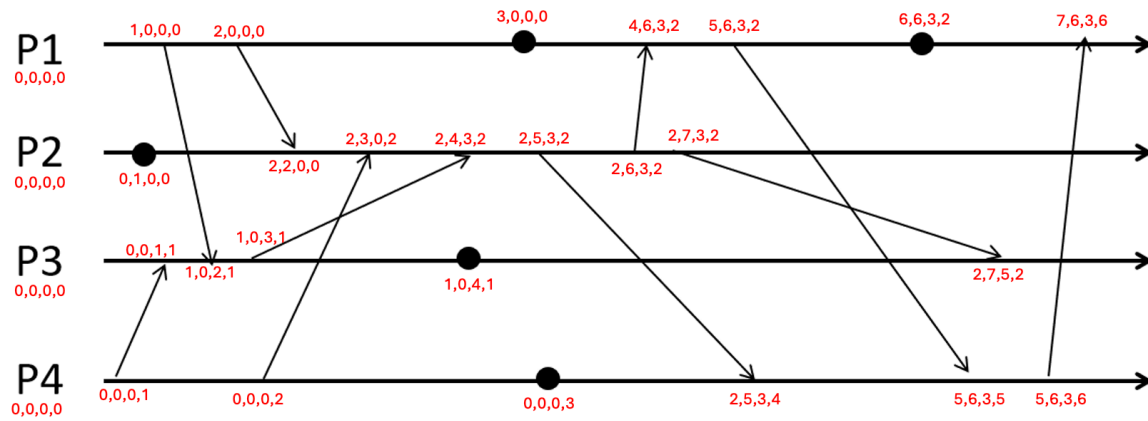
Therefore, $\text{Error} \leq [1010 - (X + 400) - 115.7] / 2 = (494.3 - X) / 2 \mu\text{s}$.

We also accept solutions that use 0 as the conservative lower bound for X since X is unknown.

8. (Solution and Grading by: Nishant. Verified by: Wei)



9. (Solution and Grading by: Xinying. Verified by: Nishant)



10. (Solution and Grading by: Weidong. Verified by: Xinying)

a. **Yes. The modified Lamport timestamps still preserve causality.**

For consecutive events E1 and E2 on the same process, the local clock is incremented, so $L(E1) < L(E2)$. If E1 is a send event and E2 is the corresponding receive event, the receiver sets its clock to: $\max(\text{local clock}, \text{message timestamp}) + 1$.

Therefore, $L(E1) < L(E2)$ also holds for message delivery. By transitivity, $E1 \rightarrow E2$ implies $L(E1) < L(E2)$. The random positive initial values only add different initial offsets and do not violate the Lamport clock condition.

B. Under the lecture algorithm, the answer is No.

In the lecture version, when process P_i receives a message, it takes the component-wise maximum only for components $j \neq i$ and then increments its own component.

For example, suppose two processes are initialized as follows:

P1: [1, 10]

P2: [1, 1]

When P1 sends message M, it increments its own component:

Timestamp of Send(M): [2, 10]

When P2 receives M, it updates the first component using the message timestamp but does not copy the message's second component. P2 then increments its own component:

Timestamp of Receive(M): [2, 2]

Although $\text{Send}(M) \rightarrow \text{Receive}(M)$, we do not have $[2, 10] < [2, 2]$. Therefore, random initialization does not necessarily preserve forward causality under the lecture algorithm.

We also accepted yes, provided students were referring to the textbook version of the algorithm.

The textbook version takes the component-wise maximum over all components before incrementing the receiver's own component. In the same example, P2 computes:

$\max([1, 1], [2, 10]) = [2, 10]$

After incrementing its own component, P2 obtains:

Timestamp of Receive(M): [2, 11]

Therefore, the receive timestamp is strictly greater than the send timestamp. Local-event ordering is also preserved because each process increments its own component for every event. By transitivity:

$$E1 \rightarrow E2 \text{ implies } V(E1) < V(E2)$$

Because the lecture and textbook use different receive rules, either answer was accepted, **but only** when supported by the corresponding correct explanation or counterexample.

c. **No.** The modified algorithm does not preserve all the usual properties of vector timestamps.

Standard vector timestamps guarantee both directions:

$$E1 \rightarrow E2 \text{ if and only if } V(E1) < V(E2).$$

With random initial vectors, the reverse direction may fail. For example, suppose two processes that never communicate start with:

P1: [1, 1]

P2: [5, 5]

Their first local events receive:

E1 at P1: [2, 1]

E2 at P2: [5, 6]

Although $[2, 1] < [5, 6]$, E1 and E2 are concurrent. Therefore, vector comparison may incorrectly suggest causality. The modified algorithm can no longer reliably distinguish causally related events from concurrent events.