HW4 Solutions: CS425 FA23

1. (Solution and Grading by: Lilia.)
    a. Cryptocurrency operates on a blockchain that enables secure and transparent peer-to-peer transactions. A blockchain is a distributed, decentralized, and public digital ledger that can be used for cryptocurrency transactions but can also be used for other applications.
    b. (We will accept answers that give the same definition for both terms) “Mining”: Repeatedly compute a hash problem to find an answer below some threshold T. Once such an answer is found, a block is considered “mined” and the miner can become the leader for that block. “proof of work ” (POW): A consensus protocol for validating transactions and generating new blocks. Miners need to solve a puzzle that can only be solved by brute force. Therefore, a valid block is a proof of “hard” work.
    c. The key difference is that permissioned has a set of trusted nodes, while permissionless does not. Anyone can join a permissionless blockchain (and remain anonymous if preferred). There are trusted nodes in permissioned blockchains, and they are typically run by companies. This also means that cheaper consensus algorithms can be run among trusted nodes that don't require all nodes to participate.
    d. No. Paxos/Raft operates in a trusted, centralized environment. Consensus in Bitcoin needs to solve the “Byzantine” model, where nodes can be malicious.

2. (Solution and Grading by: Chaitanya.)

    a. ***Yes, this interleaving is serially equivalent.*** As per the rules of serial equivalence, i.e. *all* pairs of conflicting operations occur in the same order (T1 → T2). The 2 pairs of conflicting operations are:
        i. write(a, caz, T1) & write(a, bar, T2)
        ii. write(c, foo, T1) & write(c, baz, T2)
    b. ***No, all interleavings for T1 & T2 aren't serially equivalent.*** This is one (of many) interleavings that are not serially equivalent:
        i. write(a, caz, T1)
        ii. write(a, bar, T2)
        iii. read(b, T1)
        iv. read(b, T2)
        v. write(c, baz, T2)
        vi. write(c, foo, T1)
    c. ***All 6 interleavings of T1' and T2' are serially equivalent.*** The only pair of conflicting operations is write(a, caz, T1') & write(a, bar, T2'), so it can occur in order in the interleavings. The read operations are on a different object *(c),* so they don't conflict with any of the write operations.

3. (Solution and Grading by: Taksh.)

    a. Can deadlock. Suppose there are 2 objects O1 and O2 being accessed by transactions T1 and T2. Let's say T1 accesses O1 first and then O2 in its operations, while T2 accesses O2 first and then O1. It's possible that T1 has captured an exclusive lock on O1, and is now blocking on O2 in its growth phase. Because T2 has already captured an exclusive O2 lock and is now blocking on O1 in its growth phase. This is a deadlock scenario.
    b. Cannot deadlock
    Given 2 objects (O, O') ordered as such by their "special field". Since their "special field" is globally unique, T1 and T2 should see the same ordering of objects (O, O'). Hence, there will never be a situation where T1 has acquired a lock which T2 is waiting for and vice versa.
    c. Cannot deadlock.
    Proof: Given 2 objects (O, O') ordered as such by their unique ID. Let's say that the transaction access order based on an object's unique ID does result in deadlock. Then a necessary condition is that there is a circular wait in the Wait-For graph between transactions. So that means a transaction T1 must be waiting on a resource that T2 is holding and vice-versa. For this to be true, T1 must have acquired an exclusive lock on object O that T2 is blocking on. While T2 is holding an exclusive lock on object O' that T1 is blocked on. However this would mean T1's lexicographical ordering of the objects is (O, O'). While for T2 it is (O', O). This is impossible because if the objects had a globally unique ordering, T1 and T2 should see the same ordering of objects (O, O').

4. (Solution and Grading by: Aman.)

   It does not satisfy serial equivalence.
   Counterexample:

T1	T2
Write(A) // A's lock is released after this operation	
	Write(A)
	Write(B) // B's lock is released after this operation
Write(B)	

   The transaction order for object A is (T1, T2) whereas the transaction order for object B is (T2, T1). Hence, this system will violate serial equivalence.

5. (Solution and Grading by: Shulin.)
   Cloud has 20 CPUs, 40 GB RAM
   a. 10 tasks for job 1, 5 tasks for job 2
      max(x,y) s.t
      i. x+2y <= 20
      ii. x + 2y <= 40
      iii. x/20 = 2y/20
   b. 5 tasks for job 1, 5 tasks for job 2
      max(x,y) s.t
      i. 2x+2y <= 20
      ii. 4x + 4y <= 40
      iii. x=y
   c. 4 tasks for job 1, 2 tasks for job 2
      max(x,y) s.t
      i. 2x+4y <= 20
      ii. 1x + 8y <= 40
      iii. 2x/20 = 4y/20 (x=2y)
   d. 3 tasks for job 1, 2 tasks for job 2
      max(x,y) s.t
      i. 2x+6y <= 20
      ii. 8x + 2y <= 40
      iii. 8x/40 = 6y/20

6. (Solution and Grading by: Shubhi.)
    a. Highest and lowest IDs of currently active whales: Every node sends to its parent (highest, lowest) value seen so far. If the node is a leaf, then highest=lowest=own value. Each intermediate node calculates highest=max(all highest values from children, own value), and lowest=min(all lowest values from children, own value). This (highest, lowest) value is passed to its parent.
    b. Speed and IDs of top 5 fastest whales: Each node sends (<IDs of fastest 5 whales>, <Speeds of fastest 5 whales>) to its parent. If the node is a leaf then it simply sends its own ID and speed values. If an intermediate node has <= 5 members in its subtree including its own value then it will send the speeds and IDs of all its children and itself to its parent. If the intermediate node has more than 5 members in its subtree including itself then it calculates the IDs and speeds of the 5 fastest whales from among its children and itself and sends this information to its parent.
    c. Count of all whales: Each node sends the <Sum of count value of children in its subtree + 1> to its parent. A leaf node sends value 1. An intermediate node, sums up the count values sent by all its children, adds 1 to it and sends this value to its parent.
    d. Average speed across all whales: Each node sends <The sum of speeds of each node in the subtree including itself, Sum of count value of children in its subtree + 1> to its parent node. A leaf node forwards the pair <its own speed value, 1> to its parent. An intermediate node performs the calculation as follows: <sum the speeds provided by all the children + own speed, sum the count values of all children + 1> and then passes this to its parent. The average is calculated at the root node.
    e. 75th percentile value of speed across all whales: Each node sends <Sorted list of speeds in subtree including itself> to its parent. A leaf node simply sends its own speed value to its parent. An intermediate node sorts (example using merge sort) the lists of speeds it receives from its children. It also incorporates its own speed value in this sorted list and passes it on to its parent. The 75th percentile is calculated at the root.
    Can also send <sorted (in descending order) list of speeds in subtree including itself> of maximum size N/4, where N is the number of whales in the system (in this case 3000) as we want to find the 75th percentile speed across all the whales. This can be solved in a similar way as part b.

7. (Solution and Grading by: Christina.)

   MOESI has an additional “Owned” state that MESI does not have. This additional state allows dirty data to be shared before it is written back to main memory. In order to update data, MESI needs to write it back to main memory first, but MOESI can defer this write-back. In terms of the protocols discussed in class, MESI is similar to the invalidation protocol while MOESI is similar to the update protocol.

8. (Solution and Grading by: Kshitij.)
    a. Incorrect Setup. Only the owner can hold the page in write mode. Also if one process has the page in write mode, no other copies should exist (since there can only be one owner)
    b. Incorrect Setup. Only the owner can hold the page in write mode. Also if one process has the page in write mode, no other copies should exist (since there can only be one owner)
    c. P3 should Use multicast to ask other processes to invalidate their copies of the page. Mark the local copy of the page as “Write”. Write to the copy
    d. Incorrect Setup. Every page must have an owner and only the owner can hold a page in write mode
    e. P3 should use multicast to ask all other processes to invalidate their copies of the page. Fetch all copies and mark the latest one as “Write”. Become the owner of that page and then write to the copy
    f. Incorrect Setup. Two processes cannot be in write mode for the same page.
    g. Assumption is that P1 is the owner. P3 will use multicast to ask all other processes to invalidate their copies of the page. Fetch the copy and mark it as “Write”. Become owner of the page and then write to the copy OR The setup is incorrect since the page has no owner
    h. Assuming P3 is the owner. Write to the copy. OR The setup is incorrect since the page has no owner
    i. P3 should use multicast to ask all other processes to invalidate their copies of the page. Fetch all copies and mark the latest one as “Write”. Become the owner of that page and then write to the copy

9. (Solution and Grading by: Lavanya.)
   Advantages:

1. The server can become stateless again since you do not need to keep track of callbacks at the server side. This also reduces the memory consumption and logic complexity on the server side.
2. Caching the entire file can help in improving performance of exhaustive read and write operations on the file since we do not have to make an extra gRPC call to the server for every unfetched block of the file.
3. Using a smaller freshness interval, FaceFS can have a better consistency rather than using a callback promise when a writer closes the file to invalidate other cached copies of the file.
   (other reasonable answers also accepted)

   Disadvantage:

1. If the file sizes are too huge, it is not viable to cache entire files in memory since memory is going to be limited. In this case, caching blocks of a file is a better approach.
2. In the case of the freshness interval being too small, the client-server network will incur a huge overload since we are fetching the whole file every time the freshness interval passes.
3. If the freshness interval is very big, clients will end up writing to their cached files, leading to inconsistencies across clients caches for the same file.
   (other reasonable answers also accepted)

b. Advantage:
   1. Assuming the file sizes are small and based on locality of reference, if the client is likely to fetch files closer to the current fetched file,i.e., files in the same directory, in that case, it will improve the performance to prefetch the entire directory.
   2. Assuming that the freshness interval is not too small, prefetching can improve consistency, since whenever a client's current cache of a file expires, it will fetch the entire directory. Thus, the client will have a fresher version of the entire directory every time it fetches a file.
   (other reasonable answers also accepted)

   Disadvantage:
   1. If the file sizes are very big, prefetching will result in increased network overhead to fetch the entire directory when fetching a file.
   2. In the case of a small freshness interval, everytime the client wants to refresh its cache for a file it will fetch the entire directory, resulting in a very expensive network operation.
   (other reasonable answers also accepted)

c. This is not a good idea because the client ends up making a gRPC call to the server to check the value of Tmserver every time it wants to access a file. Thus, it defeats the purpose of having a freshness interval, which was to reduce server load by making less calls to the server each time the client wants to access a file.

(**Partially** correct answer): This is a good idea since it will improve the consistency as the clients will check the Tmserver value every time they want to access a file. (good for write heavy workloads)

10. (Solution and Grading by: Maleeha.)

Any names are acceptable. Some names are listed below.
Barbara Liskov invented PBFT (Practical Byzantine Fault Tolerance) and View Stamped Replication.
https://en.wikipedia.org/wiki/Barbara_Liskov
https://pmg.csail.mit.edu/papers/osdi99.pdf
https://amturing.acm.org/award_winners/liskov_1108679.cfm

Other possible names: Deborah Estrin, Sylvia Ratnasamy, Frances Allen, Shafi Goldwasser, Dahlia Malkhi, Kim Keeton, Monica Lam, Dina Katabi, Klara Nahrstedt. All other reasonable answers accepted.

========== END of HOMEWORK 4 SOLUTION ==========