CS 425: Distributed Systems
Homework 3

Net ID: ps71

9/8/2024

Question 1: (For this question you can use the Web. However, please write answers in your own words! Don't cut and paste from websites.) You put "Consensus" on your resume, and immediately every single cryptocurrency company on this planet wants to interview you. So you pick the one with the highest market value – you're going to be a gazillionaire! But before you go to the interview, you need to educate yourself, especially as your Professor Indy seemed to not think too highly of blockchains. Answer each of the following (Please limit your answer for each part to less than 50 words. Be concise!)

Part (a): What is "mining" and "proof of work" in blockchain?

Mining is the process of validating and adding new transactions to a blockchain by solving complex cryptographic puzzles. Proof of Work (PoW) is a consensus mechanism where miners compete to solve these puzzles, ensuring network security and preventing fraud.

Part (b): How does a blockchain use consensus?

A blockchain achieves consensus through protocols like PoW or Proof of Stake (PoS), ensuring all nodes agree on a single version of the ledger. This prevents double-spending and maintains integrity without a central authority.

Part (c): What is the key difference between "permissioned" and "permissionless" blockchains?

Permissioned blockchains restrict participation to authorized entities, ensuring controlled access. Permissionless blockchains, like Bitcoin, allow anyone to join, validate transactions, and maintain the ledger without requiring approval.

Part (d): Does "proof of stake" use less energy than "proof of work", or vice versa? Why?

Proof of Stake (PoS) uses significantly less energy than PoW because it doesn't require miners to solve computationally expensive puzzles. Instead, validators are chosen based on their staked cryptocurrency, reducing electricity consumption.

Question 2: During an interview at IBM Research Almaden, they tell you that the relational database model was invented by E. F. Codd, so they love transactions. They give you a log of two transactions executed concurrently by two clients – T1 and T2 (a, b, c are objects at the server):

T1: read(a, T1); write(b, caz, T1); read(a, T1); write(c, foo, T1);
T2: read(b, T2); write(b, bar, T2); write(a, baz, T2); read(c, T2);
The transaction management system seems to think each of the following interleavings is serially equivalent. For each of the following interleavings, say if (and why/why not) it is serially equivalent:
First, we will list all the pairs of conflicting operations between T1 and T2:
1.read(a, T1)— write(a, baz, T2)
2.[2nd] read(a, T1) — write(a, baz, T2)
3.write(b, caz, T1) — read(b, T2)
4.write(b, caz, T1) — write(b, bar, T2)
5.write(c, foo, T1) — read(c, T2)

Part (a): Alternating T1's and T2's statements, starting with T1's first statement, then T2's first statement, then T1's second statement, and so on.

For this interleaving, these are the orders in which the above pairs are executed:
1.(T1, T2)2. (T1, T2)3. (T2, T1)4. (T1, T2)5. (T1, T2) Since the 3rd pair is executed in a different order than all the other pairs,this interleaving is NOT serially equivalent.

Part (b): Alternating T2's and T1's statements, starting with T2's first statement, then T1's first statement, then T2's second statement, and so on.

Following the same logic:
1.(T1, T2)2. (T2, T1)3. (T2, T1)4. (T2, T1)5. (T2, T1) Since the 1st pair is executed in a different order than all the other pairs,this interleaving is NOT serially equivalent.

Part (c): read(b, T2); read(a, T1); write(b, caz, T1); read(a, T1); write(b, bar, T2); write(a, baz, T2); write(c, foo, T1); read(c, T2);

1. (T1, T2)2. (T1, T2)3. (T2, T1)4. (T1, T2)5. (T1, T2) Since the 3rd pair is executed in a different order than all the other pairs,this interleaving is NOT serially equivalent.

Part (d): read(a, T1); write(b, caz, T1); read(b, T2); write(b, bar, T2); read(a, T1); write(a, baz, T2); read(c, T2); write(c, foo, T1);

1. (T1, T2)2. (T1, T2)3. (T1, T2)4. (T1, T2)5. (T2, T1) Since the 5th pair is executed in a different order than all the other pairs,this interleaving is NOT serially equivalent.

Part (e): Your interviewer claims that if T1 had its 2 write operations removed (and so had only read operations), while T2 was left unchanged, then any interleaving of T1 and T2 would be serially equivalent. Are they correct? If yes, justify why. If no, give a counter-example.

No, the interviewer is incorrect.Take the following counterexample: read(a, T1); read(b, T2); write(b, bar, T2); write(a, baz, T2); read(a, T1); read(c, T2);
This follow T1-T2 for first read(a, T1); write(a, baz, T2), while T2-T1 for second write(a, baz, T2); read(a, T1);

Question 3: When interviewing at Oracle, they tell you that they use a transaction management system that assigns unique words as ids to to objects, where the words are dictionary words (Oxford English dictionary only). The system uses exclusive locking of objects for concurrency control. The system restricts transactions to acquire locks only in lexicographically decreasing order of object id (i.e., if a word appears later in the dictionary it has a higher object id, and it will be locked first). They tell you that this system will not deadlock only if all locks are acquired at the start of the transaction. You think otherwise, and say that even if you acquire locks during the transaction (obeying the decreasing lexicographic rule), there will be no deadlocks. Who is right? Give an example or present a proof, as applicable. You can assume 2 phase locking for this problem.

Analysis

The Oracle engineers claim that deadlocks can occur if locks are acquired incrementally, even when following a lexicographically decreasing order. We analyze whether this claim is not correct.
Our solution that the system will not deadlock even if we try to acquire locks in the middle of transaction is right. Since we are only acquiring locks based on the lexicographical order of the words in the dictionary and since the order of words in the dictionary does not change, there will not be a situation where a word which is lower in the order requests for a lock on a word which is higher in the lexicographical order.
zebra -> Monkey -> Elephant -> Dog -> Cat
Take for example a transaction requires the locks on these particular transaction ids. Since it is mentioned that the system restricts transactions to acquire locks only in lexicographically decreasing order of object id, it does not matter when the locks are acquired as the order of acquiring locks does not change and even if we tried to acquire locks in the middle of a transaction, the system would not be a deadlock. Assuming that it is a 2-phase locking system, meaning that once a transaction has started acquiring locks, it cannot start releasing these locks until all required locks are acquired and vice-versa.

Question 4: In traditional two phase locking, as you've seen in lecture, a transaction cannot acquire (or promote) any locks after it has started releasing locks. Thus transactions have a growing phase and a shrinking phase. During one of your interviews you meet Gordon Gekko, who tells you that approach is too safe, and that greed is better. But Gordon has been tempered by his time in prison. He tells you that he has a new algorithm called limited-greedy two phase locking. The idea is the following: there is still a growing phase where locks can only be acquired. However, once the shrinking phase starts, the transaction has an opportunity to acquire at most two new locks (objects are never locked more than once by one transaction).

Part (a): Does this limited-greedy two phase locking still satisfy serial equivalence? Give a proof/counterexample (as appropriate).

Limited-Greedy two-phase locking does not satisfy serial equivalence. This is demonstrated in the example below.

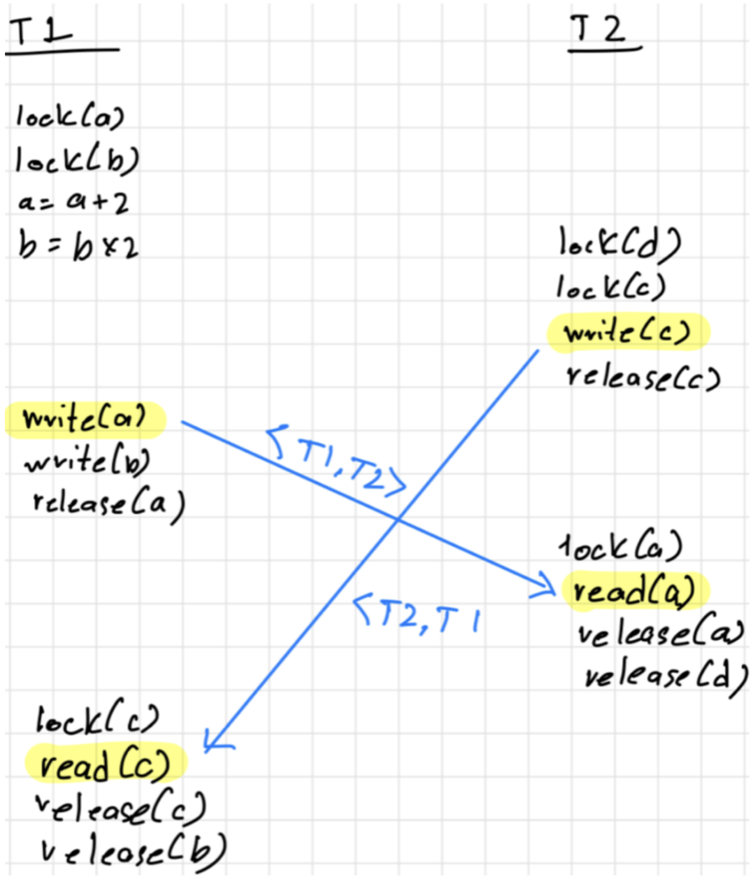

Figure 1: sol. 4a

Part (b): Can this limited-greedy two phase locking deadlock? Give a proof/counterexample (as appropriate).

As shown in the above diagram, T1 acquires lock on object a and b, releases b and then tries to acquire lock on object c, whereas T2 acquires lock on objects c and d, releases d and then tries to acquire lock on object a which T1 has not released yet. This leads to a deadlock in the system as both transactions cannot proceed without one of them releases the required resource.

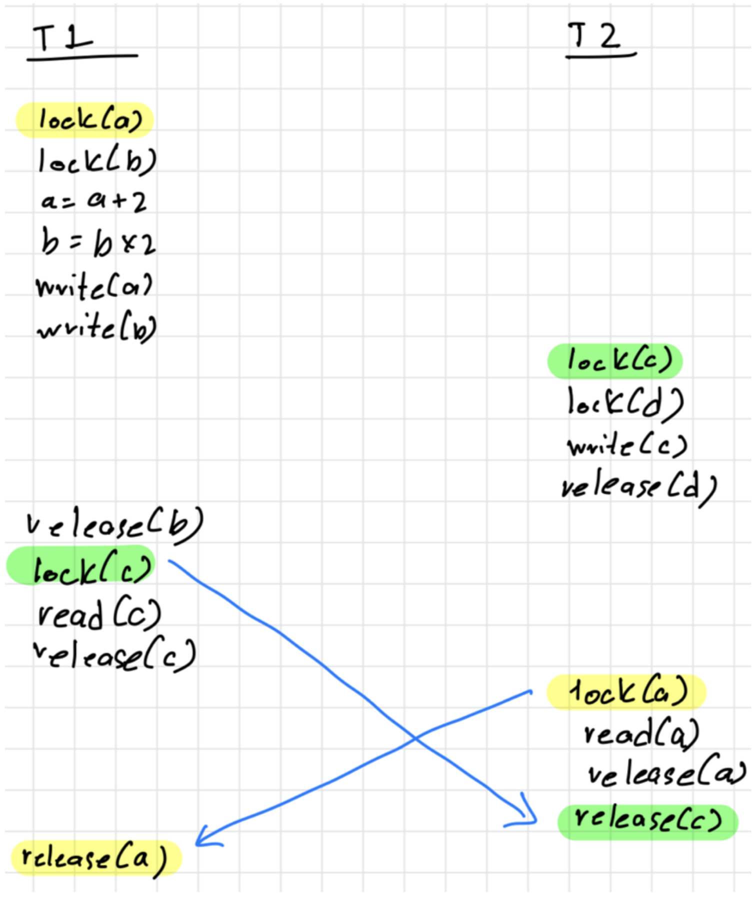

Figure 2: sol. 4b

Question 5: Your next interview is with Montgomery Burns, who thinks he can one-up Gekko Gordon and so he comes up with an "Excellent!" scheme. Mr. Burns presents you a transaction system where he has each transaction acquiring a lock for an object just in time, right before the transaction's first access to that object, and then the transaction releases the lock object right after the last access of that transaction to that object.

Part (a): Will this system satisfy serial equivalence? Give a proof/counterexample (as appropriate).

This system does not satisfy serial equivalence. As mentioned in the question, we do not release the lock until the last access of that object in a transaction. Meaning if any other transaction tries to acquire lock while the first transaction is still using the object, it must wait till the first transaction releases that lock. But in case that there are multiple objects that need to be used in the transaction, this system will fail serial equivalence, as shown in the example below. Consider this example:

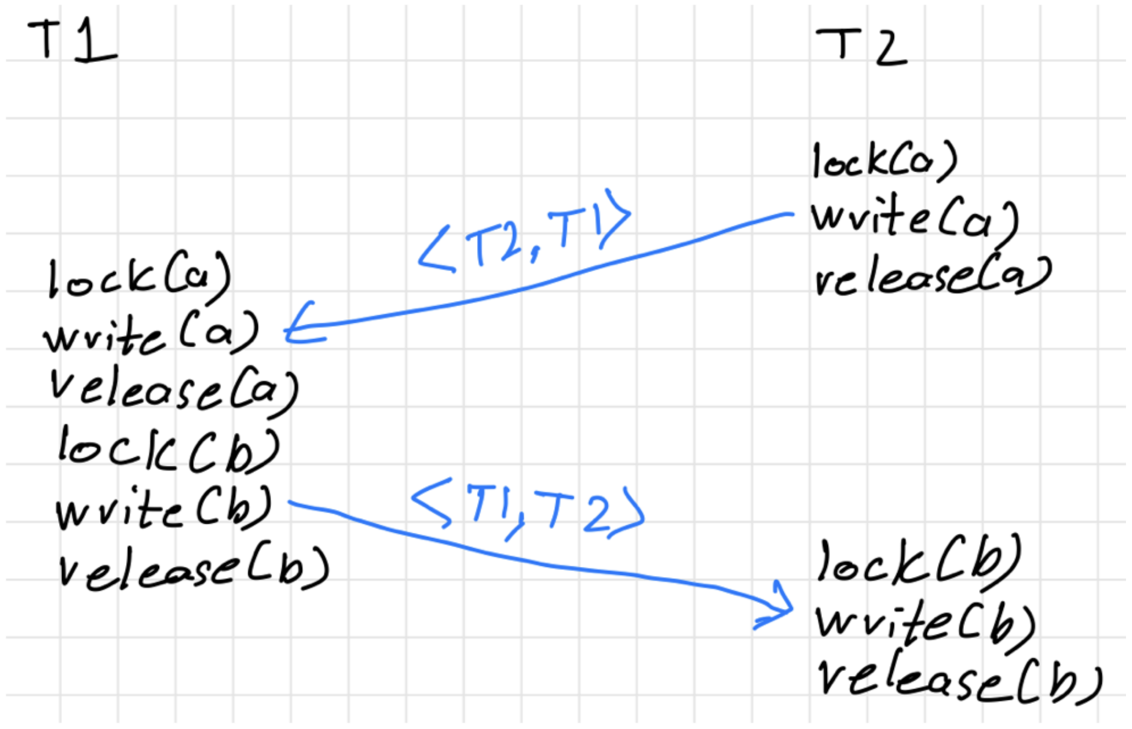

Figure 3: sol 5a

Here, transaction T1 acquires the lock after T2 is done with object a, hence the ordering will be <T2, T1>, whereas in case of object b, transaction T1 acquires lock before T2 and hence T2 acquires lock after T1 is done with object b, leading to the order <T1, T2>. Hence proved, this system does not satisfy serial equivalence.

Part (b): Can this scheme deadlock? Give a proof/counterexample (as appropriate).

This system will have deadlocks, explained in the diagram below.

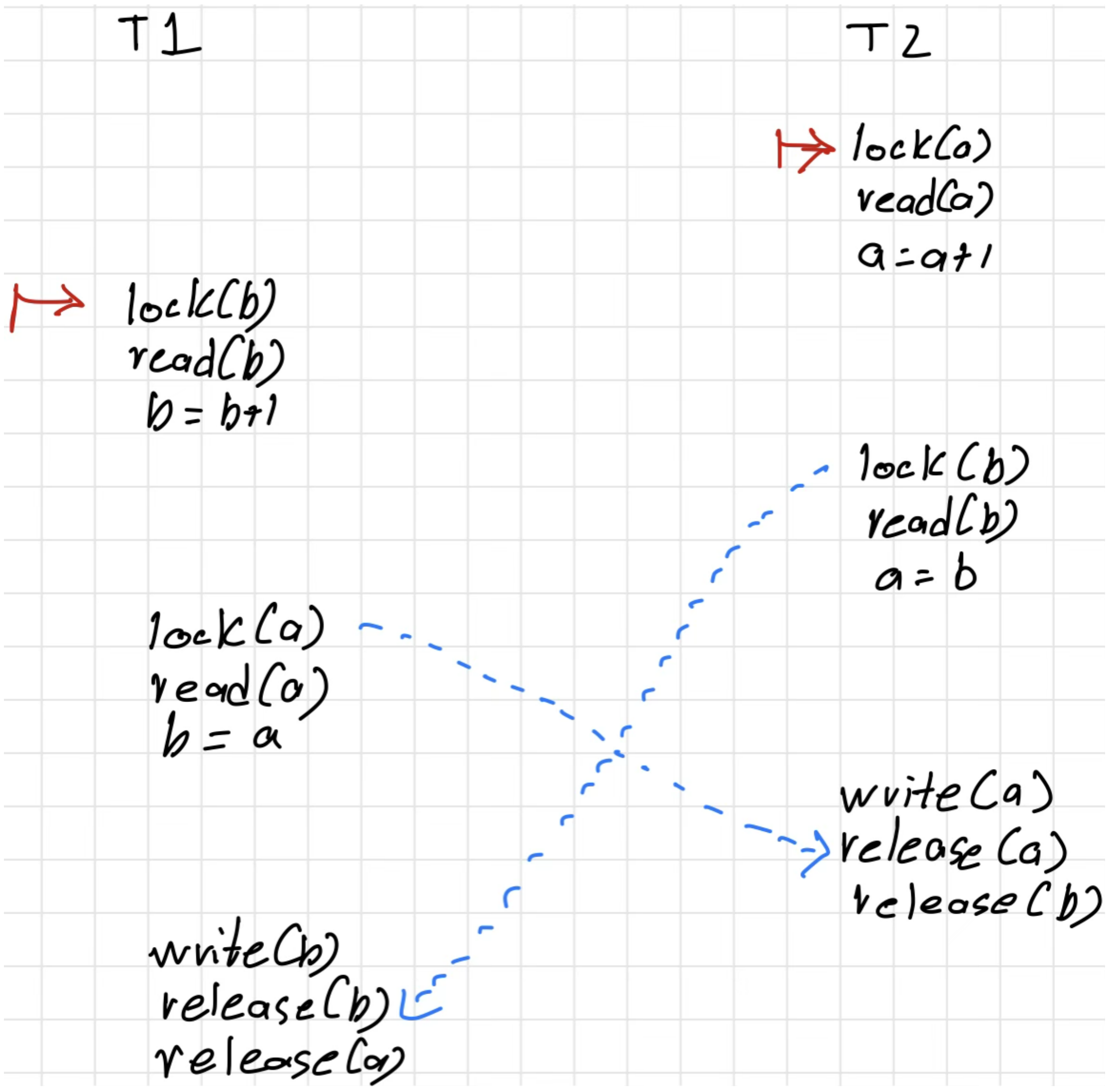

Figure 4: sol 5b

Transaction T2 acquires lock on object a, reads a and does a+1. Transaction T1 acquires lock on object b, reads from it and does b+1. Now Transaction T2 is waiting to acquire a lock on object b, whereas Transaction T1 is waiting to acquire a lock on object a. Since both these transactions were waiting to get access just in time to read/write to these objects. This leads to a deadlock and hence the system can lead to a deadlock.

Question 6: While you're interviewing at the top-secret Cyberdyne Corporation, they face a catastrophic attack from a group of hackers known as "The Terminators". The attack partitions their datacenter into two partitions – the NYC partition and the NJ (New jersey) partition. Calmly, you tell your interviewers that this is called a partitioned system, you learnt this in CS425, and you've got this. You've got a few choices for how to handle this partition – these choices are listed below. For each of these choices, say if it: i) violates (strong) consistency (and why), and ii) if it violates availability (and why).

Part (a): Allow only partitions with a quorum of servers (measured across both partitions) to execute writes and reads.

Violates strong consistency, as its eventual consistency. Yes available throughout the time though.

Part (b): Until partitions are repaired, don't allow any operations.

Part (c): Allow both NYC and NJ partitions to process reads, but only the NYC partition to process writes.

Part (d): Allow only the partition that has at least a quorum number of servers (measured across both partitions) to execute writes; reads can be executed in both.

Part (e): Allow reads and writes in a partition only if it has a quorum number of servers responsive (quorum measured only across servers within that partition).

Part (f): Until partitions are repaired, allow only reads but no writes.

Part (g): Allow only the partition that has at least a quorum number of servers (measured across both partitions) to execute reads; writes can be executed in both.

Part (h): Allow both NYC and NJ partitions to process writes, but only the NJ partition to process reads.

Question 7

(For this question you can only look up the linked paper) The folks at Berkeley and Stanford are surprised to know that you know about their invention DRF. In lecture we discussed Dominant Resource Fairness (DRF), but we did not discuss equations to derive "fair" allocations. Look at the equations in the original paper, especially Section 4 and 4.1. Here is the only paper you can access for this question: link. Then calculate fair allocations for each of the following cases (Cloud has 20 CPUs, 50 GB RAM):

Part (a): Job 1's tasks: 2 CPUs, 4 GB. Job 2's tasks: 2 CPUs, 4 GB.

Job1: CPU: 2/20 = 10/100 (CPU Intensive)
RAM: 4/50 = 8/100

Job2: CPU: 2/20 = 10/100 (CPU Intensive)
RAM: 4/50 = 8/100

Equations: => 2x+2y<=20
=> 4x+4y<=50
=> 2x=2y

Solving these equations we get x=5 and y=5.

Resources Allocated:
Job1 gets <10CPUs,20GB RAM>
Job2 gets <10CPUs,20GB RAM>

Part (b): Job 1's tasks: 4 CPUs, 3 GB. Job 2's tasks: 2 CPUs, 8 GB.

Job1:
CPU: 4/20 = 20/100 (CPU Intensive)
RAM: 3/50 = 6/100

Job2:
CPU: 2/20 = 10/100
RAM: 8/50 = 16/100 (Memory Intensive)

Equations:
4x+2y <= 20
3x+8y <= 50
5x = 4y

Solving these equations we get x=3 and y=4. Solving the above equations using a graphing calculator to find where the point coincides and then finding the closest integers that satisfy the above conditions.
Resources Allocated:
Job1 gets <12CPUs,9GB RAM>
Job2 gets <8CPUs,32GB RAM>

Part (c): Job 1's tasks: 2 CPUs, 8 GB. Job 2's tasks: 8 CPUs, 1 GB.

Job1:
CPU: 2/20 = 10/100
RAM: 8/50 = 16/100 (Memory Intensive)

Job2:
CPU: 8/20 = 40/100 (CPU Intensive)
RAM: 1/50 = 2/100

Equations:
$2x+8y=20$
$8x+1y=50$
$2x=5y$

Solving these equations we get $x=3$ and $y=1$, Solving the above equations using a graphing calculator to find where the point coincides and then finding the closest integers that satisfy the above conditions.
Resources Allocated:
Job1 gets <6CPUs,24GB RAM>
Job2 gets <8CPUs,1GB RAM>

References