CS145 Fall 2022 Homework 3

10 November 2022

HW 3 is due on Tuesday 11/29 at 11:59PM (No Late Days Allowed)

Question 1 [8 points] - Notebook & Transactions

Here's a Notebook summarizing key equations on IOCosts and SMJ/HPJ. Also, for Transactions, the speedup for LOGs for small writes. For this question, play with the Notebook(Please note this notebook is view only. You can make a copy of it and run it for your viewing).

1.1 (4 points)

Summarize 2 examples you experimented with, and any interesting conclusions you drew.

1.2 (4 points)

Suggest 2 ideas to improve the Notebook. E.g.. specific new functions, or new examples.

Question 2 [4 points] - The Problem with Crashes/Aborts

Examine the following transaction:

1. Read AccountA
2. Read AccountB
3. AccountB = AccountB + AccountA
4. AccountA = 0

For each step, explain how the output of the program(the states of AccountA and AccountB) would change if the computer crashes right before that step is executed.

Question 3 [8 points] - ACID

For 3.1 to 3.3, it is sufficient to provide a letter choice without an explanation.

3.1 Atomicity (1 point)

What is **NOT** a possible result from an atomic transaction?

(A) All changes are made

(B) Some changes are made

(C) No changes are made

3.2 Consistency (1 point)

Which of the following examples is related to consistency in transactions?

(A) A teller looking up a balance at one time should not be allowed to see a concurrent transaction involving a deposit or withdraw from the same account. Only when the withdraw transaction commits successfully and the teller looks at the balance again will the new balance be reported.

(B) When processing medical tests, there's a whole set of rules that medical professionals have to follow. Doctors and their staff have to fill in a requisition properly. The specimen collection center has do verify that information, take samples and pass everything on to a lab. The lab that performs the tests must ensure that everything is valid before performing the test.

(C) To transfer funds from one account to another involves making a withdraw operation from the first account and a deposit operation on the second. If the deposit operation failed, you don't want the withdraw operation to happen either.

(D) A data warehouse can still keep the contents of its database even facing a system crash or other failure.

3.3 Isolation (1 point)

Which of the following statements about isolation is correct?

(A) To maintain the isolation of transaction, the transactions must be executed serially.

(B) Isolation guarantees that once a transaction has committed, its effects remain in the database.

(C) Isolation can be achieved by wisely scheduling the transactions as if they are executed serially.

(D) Isolation of the transactions can be achieved by Write-Ahead Logging

3.4 Durability (2 points)

Given the following transaction T:

T	R(A)	R(B)	A := A+B	B := A+B	COMMIT

Let A = 1 and B = 2 at the start of the execution. We run this transaction on a database that does not guarantee Durability.

If our computer crashes after committing T, what are the possible values of A and B after it restarts? Please list all possible pairs of values for A and B.

3.5 ACID and Scheduling (3 points)

How can transaction scheduling affect the ACID properties of a DBMS? Note all properties that may be affected. Explain by providing an example.
Here is an example of initial states and transactions. You can create your own example if you want.

- Initially, A = 1, B = 1.
- T1: A *= 2, B *= 2.
- T2: B -= 1, A -= 1.

Question 4 [8 points] - Write-Ahead Logging (WAL)

Transaction T performs the following operations:

1. R(A)
2. R(B)
3. B := B+1, e.g. W(B)
4. A := A+1, e.g. W(A)

4.1 (4 points)

Initially, *A = 0, B = 1*, and each write operation increases the record's value by 1. Assume that the main memory and disk both start out empty. Fill out the following table to describe the step-by-step changes to both the data and the log on main memory and disk during each step of T and for each of the 2 steps in which WAL writes the log and data to disk (6 steps total). Note that not all cells may need to be filled out.

Step	Data-RAM	Log-RAM	Data-Disk	Log-Disk
1				
2				
3				
4				
5 (Write to Log-Disk)				
6 (Write to Data-Disk)				

4.2 (2 points)

Imagine we alter the WAL protocol so that we first commit T, then write the data and log records to disk. Describe which ACID property we lose and give an example why.

4.3 (2 points)

Now, say we alter our WAL protocol so instead it writes the data to disk before the corresponding log record. Describe what ACID property we lose, and give an example why.

Question 5 [6 points] - Conflict Serializability

Consider the following schedules, each involving two transactions T1 and T2. For each schedule, specify whether or not the schedule is conflict serializable and why. If the schedule is conflict serializable, give the equivalent serial schedule.

5.1 Schedule 1 (2 points)

T1		R(B)	W(C)			R(C)
T2	W(B)			R(C)	W(A)	
Timestamp	0	1	2	3	4	5

5.2 Schedule 2 (2 points)

T1	R(B)	W(C)				R(C)
T2			W(B)	R(C)	W(A)	
Timestamp	0	1	2	3	4	5

5.3 Schedule 3 (2 points)

T1	R(A)		R(B)			W(A)		
T2		R(A)		R(C)	W(C)		R(B)	W(B)
Timestamp	0	1	2	3	4	5	6	7

Question 6 [3 points] - Conflict Types

Let's say you are buying a $2.00 cup of coffee at Coupa Cafe. But just after you've paid and your debit card transaction is being processed, Stanford decides to levy a draconian tax of 2% of both your account and Coupa Cafe's account. Consider the following simplified interleaving of transactions:

T1			Your acct -= $2.00		Coupa's acct += $2.00			
T2	Coupa's acct*= .98						Your acct*= .98	
Timestamp	0	1	2	3	4	5	6	7

You can assume that each transaction above consists of a read and a write, in that order, with each read and each write taking up a unit of time. For example, the read in "Coupa's acct*= .98" occurs at timestamp 0, and the write occurs at timestamp 1.

What are all the pairs of actions that conflict and what type of conflict do they form? Identify them by their timestamp.

Question 7 [9 points] - Two-Phase Locking

For each of the following schedules, specify whether the schedule is conflict serializable and whether it is possible for the schedule to be executed in the exact given order under strict 2PL. Make sure to explain your answer. Remember that the 2PL protocol grabs an X (exclusive) lock for writing and an S (shared) lock for reading.

7.1 Schedule 1 (3 points)

T1	R(A)	W(A)		R(B)	
T2			W(A)		W(B)
timestamp	0	1	2	3	4

7.2 Schedule 2 (3 points)

T1	R(A)		R(B)	W(B)	
T2		R(A)			R(A)
timestamp	0	1	2	3	4

7.3 Schedule 3 (3 points)

T1	R(A)		R(A)		
T2		W(A)		W(B)	R(B)
timestamp	0	1	2	3	4

Question 8 [4 points] - Deadlocks

In lecture, we saw that 2PL is susceptible to deadlocks. Provide a series of lock requests made by some transactions T1, T2, and T3 over shared resources A, B and C such that a deadlock occurs. Draw the corresponding waits-for graph for your solution.

Question 9 [25 points] - Hashing

As we saw in lecture, hashing is very important for managing large data systems. For example, it is used to map from data/search keys to the location where that data is stored (memory address, IO block, machine/disk). Another common application is evenly dividing a set of inputs into buckets in order to process them in parallel, for example when counting large numbers of tuples. In this problem, we'll get a little more intuition about hash functions and collision resolution by looking at some examples. (Note: there are some question numbering changes on Gradescope, please fill the answer on the right place)

Question 9.1 [5 points] - Some Intuition about Hash Collisions

The details of how different hash functions work, including how they are implemented and their statistical properties, are mostly beyond the scope of this class. However, a feature common to all hash functions is collisions. When dealing with hashing, it's helpful to have some intuition about the frequency of collisions. Recall that a collision occurs whenever for two distinct elements a and b and a hash function $h(x)$, $h(a) = h(b)$.

Assume you have a hash function (the actual process by which it operates is unknown) with the following properties:

- The outputs (hash values) are of size 8 bits (there are 256 possible hash values/buckets).
- The hash function distributes its outputs evenly across the entire output range (this property is very desirable in hash functions and is called **uniformity**).

What is the probability of having at least one collision if we are hashing 5 inputs? Give your answer as a numerical value.

What about for 10 inputs? 20 inputs? 50 inputs?

NOTE: One of the interesting (and somewhat counter-intuitive) facts about hash functions is that the probability of collision rises much faster than intuition might suggest. Even when hashing relatively small numbers of inputs (w.r.t the output range), the collision probability rapidly approaches 1. See https://en.wikipedia.org/wiki/Birthday_problem for more interesting discussion on this problem.

Question 9.2 [6 points] - Thinking about Collision Resolution

In most applications of hashing, how collisions are handled is an important part of the implementation. One example of this is in the implementation of hash partition join, which we discussed in lecture. Consider the case where we wish to join two relations R and S on a shared attribute A. The first step in the process, partitioning, is done using a hash function. We hash tuples from both relations using their attribute A in order to divide them up evenly into a finite number of buckets B.

In this process of partitioning, hash collisions may occur. Explain why having a large number of hash collisions in our buckets would harm the efficiency of the hash partition join algorithm.

In light of this, a method is needed to address collisions. As described in lecture, this can done with **double hashing**. If the initial partition resulted in collisions, these can be resolved by doing a second pass and re-hashing each collided element with a new hash function (on attribute A). The result of this second hash is used as an offset to determine how many bins to move the tuple over. For example, if a tuple is hashed initially to bin 3 and there is a collision, if $h_2(tuple) = 1$ then the tuple will be moved to bin $3 + 1 = 4$ (if this again results in a collision, the tuple can be shifted again by $h_2(tuple)$ until the collision is resolved). In practice, a common choice for this second hash function is $Hash_2(key) = K - (key \mod K)$, where K is a prime smaller than the number of buckets B.

You are given $B = 5$. You are also given the following relation R.

A	C	E
5	93	Red
7	28	Purple
3	70	Orange
11	545	Blue
6	88	Brown

The initial hash function is $h_1(A) = A \mod B$. The second hash function is $h_2(A) = 3 - (A \mod 3)$.

As part of the first step of a hash partition join, partition R using the given hash function. Assume that the tuples are hashed into buckets in the order they appear in the table (from top to bottom). Resolve any collisions with the provided second hash function. You can indicate your answer in the form of a table or by listing the tuple(s) in each bucket (B0, B1, ...).

Bonus Note: There are many other collision resolution strategies beyond those mentioned here, each with different advantages and disadvantages which can be analyzed probabilistically. If you're interested, CS166 covers hash tables and collision resolution in more depth.

Question 9.3 [4 Points] - Computing Counts of Pairs

As seen above, hash functions are useful whenever we need to (relatively) evenly distribute data. Another such application is dividing across multiple machines to process in parallel.

Imagine you have created a music app. Users of your app can login, start a session, and then play songs. The database which forms the backend to your app contains a table with tuples (user_id, session_id, song_id) which represent every time a user has played a song. You wish to see which pairs of songs are most frequently listened to together in the same session.

You are given the following information:

- There are 10 million users
- There are 1 thousand songs
- There are 4 billion sessions
- The user IDs are 3 bytes
- The song IDs are 2 bytes

- The session IDs are 4 bytes
- The avg. number of unique songs played per session is 5

Assume all your data is stored on HDDs and use the values for disk seek/scan times from the top of the assignment. Assume that data is stored sequentially on disk.

Calculate the total time required to compute the counts for each distinct pair(this involves three steps: 1. Reading the data from HDD; 2. Writing the pairs to HDD. 3. Sorting the pairs, scanning pairs and making GROUPBYs). Do not count pairs with the same song (i.e. Song A - Song A) and assume that when counting we do not count different orderings of the same songs as distinct pairs (i.e. if Song A and Song B are played in the same session, we only count the pair Song A - Song B and do not count the pair Song B - Song A). Assume that it takes 1 hr. to complete the sorting step(includes sorting all pairs using the sort merge join algorithm, scanning sorted pairs and making GROUPBYs).

Question 9.4 [6 Points] - Parallelizing Counting with Hashing

Now imagine you have a hash function which maps from any 64-bit input uniformly to a 32-bit hash value.

Think about how you could use it to parallelize the counting across n machines.

What would the total required time to compute the counts for each pair be once you've used your hash function to divide the pair tuples across 6 separate disks (which can write in parallel)?

Question 9.5 [4 Points] - Thinking a Little More About Hashing

Consider the application of hashing where we want to parallelize a task across n machines (where n is reasonably small, say 100) versus the task of using hashing as a way to map keys to a location (e.g. generating IDS).

For each of these two applications, is a low collision rate important? For each of these two applications, is uniformity important? Why?