CS 764: Midterm Exam, Fall 2021

Please ***print*** your name below.

Last Name:	
First Name:	

Instructions:

1. This is an open-book, open-notes exam. You can use any material provided in this course or on the Internet.
2. You are **not allowed** to discuss the exam questions and solutions with others during the exam.
3. You have **48 hours** to complete this exam. Please send your solutions to yxy@cs.wisc.edu before **noon, 11/17/2021 (Wednesday)**.
4. Make sure your answers are precise, complete, and clear. The following are suggested ways to submit your solutions:
   - Directly type your solutions in this MS word document
   - Print out the exam and submit a photocopy of your solutions
   - Convert the exam into a pdf file and write your solutions on it
5. For any clarification questions, please either email the instructor: yxy@cs.wisc.edu or post private questions on piazza.

(For Instructors Use)

Q1 (30 points): Join & distribution	
Q2 (30 points): Buffer management & query optimization	
Q3 (30 points): Concurrency control & indexes	
Q4 (30 points): Durability	
TOTAL (120 points)	

Question 1. [30 points] Join & Distribution

a) [**4 points**] Please write down one advantage and one disadvantage of radix join (Lecture 3) compared to simple hash join (Lecture 2) for main-memory databases where all tables fit in memory.

Advantage:

Disadvantage:

For part b) and part c), we try to estimate the data traffic between memory and CPU when joining two relations, R and S.

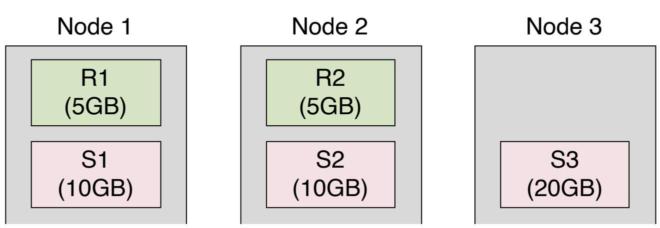
Relation R: each record is 10B; the table contains 10 million records (100MB in size)
Relation S: each record is 10B; the table contains 100 million records (1GB in size)

For simplicity, we assume both relations are in main memory when the program starts. For your calculation, please assume each random read or write incurs 64B (i.e., the cacheline size) of memory traffic; the output of the join is not written back to memory and thus can be ignored in your calculation.

b) [**8 points**] What is the read and write traffic, respectively, for simple hash join? You can assume the hash table is built on relation R and has the same size as relation R (i.e., 100MB); the hash table is entirely stored in memory (i.e., not cached in SRAM). Note that the writes and probes to the hash table are random memory accesses.

c) [**8 points**] What is the read and write traffic, respectively, for radix join? You can assume that one round of partitioning is sufficient for each partition of the smaller relation to fit in cache. Please also assume that partitioning requires scanning a relation twice — first time to collect partition size and second time to partition data (Lecture 3 slides, page 16). You can assume the CPU cache is used only for caching the partitioned hash tables.

d) [**10 points**] Consider the following data layout in a three-node cluster



Relation R is partitioned across Nodes 1 and 2 with 5GB of data on each node. Relation S is partitioned across all the three nodes with 10GB, 10GB, and 20GB on them, respectively.

For the following two conditions, please identify join methods that minimize the total network traffic. We consider the three methods discussed in Lecture 7, i.e., single-site, broadcast, and co-partition. Please explain the chosen method (e.g., how much data of what relation is sent from which source node to which destination node) and calculate the total network traffic. Please assume uniform data distribution.

Condition 1: S is partitioned based on the join key.

Condition 2: S is partitioned **not** based on the join key.

Question 2. [30 points] Buffer Management & Query Optimization

Consider the following sequence of 15 accesses to pages with IDs 1–5.

1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5

The database has a buffer pool that can hold 4 pages. Initially, all the pages are on disk and the buffer pool is empty.

a) [**10 points**] How many buffer pool misses will occur if Least-Recently Used (LRU) and Most-Recently Used (MRU) policies are used, respectively?

LRU:

MRU:

b) [**10 points**] We replace the disk in the system with non-volatile memory (NVM). Since NVM is byte addressable, one can directly access a page in NVM without loading it to DRAM, avoiding the overhead of replacement. We assume the cost of a buffer pool hit is 1, the cost of a miss without replacement is 2, and the cost of a miss with replacement is 3 (since it involves an NVM read and a DRAM write). In this context, is there a new cache management policy that achieves lower cost compared to LRU and MRU? If so, please explain how the new policy works.

Consider the following schema and SQL query

Relation R (a, b): 10 million tuples, R.a is the primary key

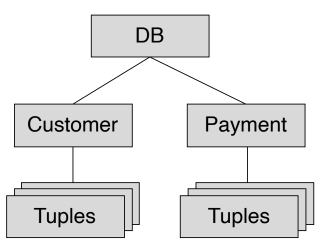
Relation S (c, d): 100 million tuples, S.c is a foreign key referring to R.a

```
SELECT *
FROM R, S
WHERE R.a = S.c
AND R.d = 5;
```

c) [**10 points**] Please estimate the number of rows in the output relation using the techniques in Selinger'79 (Lecture 6, Query Optimization).

Question 3. [30 points] Concurrency Control & Indexes

Consider the following locking hierarchy where a single database contains two tables: **Customer** and **Payment**. Each table contains a number of records.



Consider the following two transactions:

T1: read one Customer record and update the one corresponding Payment record.

T2: read all records in Payment and calculate sum

a) [**5 points**] What lock mode will T1 request on each entity in the hierarchy?

b) [**5 points**] What lock mode will T2 request on each entity in the hierarchy?

c) [**5 points**] Assume T1 has acquired all the locks but has not released any, and now T2 starts to acquire locks. Will T2's lock requests conflict with T1's locks? If so, please explain on which entity in the hierarchy will the conflict be detected.

d) [**5 points**] Several studies show that B+ tree achieves better performance than Adaptive Radix Tree (ART) for range scan operations. Please write down one aspect of ART that likely causes this performance difference.

e) [**10 points**] Page 223 of the classic OCC paper contains the following parallel validation algorithm.

```
for t from start tn + 1 to finish tn do
    if (write set of transaction with transaction number t intersects read set)
      then valid := false;
for i ∈ finish active do
    if (write set of transaction Ti intersects read set or write set)
      then valid := false;
if valid
    then (
      (write phase);
      ⟨tnc := tnc + 1;
       tn := tnc;
       active := active—{id of this transaction});
      (cleanup))
    else (
      ⟨active := active—{id of transaction}⟩;
      (backup))).
```

In line 5, both read and write sets of the validating transaction are tested against Ti's write set. Consider an alternative design that changes the highlighted "read set or write set" to "read set". Will the algorithm still work correctly? Please justify your answer.

Question 4. [30 points] Durability

A database uses ARIES as the logging protocol. Below is the content in the log when a crash occurs

LSN	Log entries
1	[Update] T1 writes P1
2	[Update] T2 writes P2
3	Begin checkpoint
4	Transaction Table (T1, LastLSN=1), (T2, LastLSN=2)
5	Empty Dirty Page Table
6	End checkpoint
7	[Update] T1 writes P3
8	T1 commit
9	T1 end
10	[Update] T2 writes P1
11	
12	
13	
14	
15	
16	
17	
18	
19	
20	

a) [**10 points**] When the crash occurs, can you tell what PageLSN is stored on disk in each data page (i.e., P1, P2, and P3) respectively? If so, please write down the PageLSN values for the corresponding pages; otherwise, please explain why the value cannot be determined based on the given information.

b) [**10 points**] The REDO and UNDO phases of ARIES may also update the log content. Please write down the log content after the whole recovery process. You can use the table above to fill in the new log records. We assume the database writes another checkpoint at the end of the recovery. In this question only (but not in part a), we assume the database does not flush any data pages to disk during recovery (namely, the database flushes only log records). You do not have to use all the empty lines in the table above.

Please follow the following format for CLR and Dirty Page Table records

CLR: UNDO Tx, LSN=y, UndoNxtLSN=z (LSN=y points to the update that is being undone)

Dirty Page Table (Px, RecLSN=y), (Pz, RecLSN=w), …

c) [**10 points**] Below is the unoptimized two-phase commit (2PC) process of a distributed transaction. The execution on the node sub1 is read-only. This process can be optimized using the "presumed abort" technique. Please draw the 2PC process when the optimization is applied.