CS 764: Midterm Exam, Fall 2022

The Computer Science Department at UW-Madison is committed to ensuring an environment of honesty, trust, fairness, respect, and responsibility. As such, students are expected to adhere to our high departmental standards of academic honesty and rigor.

By signing this agreement, I am affirming that:

- I understand the classroom policies regarding resource usage and collaboration on this exam.
- I understand that if I commit an act of academic misconduct, I am subject to disciplinary actions, including, but not limited to:
    - A failing grade on this exam
    - A failing grade in the course
    - A reprimand to be included in my disciplinary file

Sign your name below:

Your Full 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 kerenchen@cs.wisc.edu before **noon, 11/11/2022 (Friday)**.
4. Make sure your answers are precise, complete, and clear. The following are suggested ways to submit 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 post private questions on piazza.

(For Instructors Use)

Q1 (30 points): Join & parallel database	
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 & parallel database

a) **[6 points]** Consider joining two relations R and S (|S| >> |R|) for an on-disk database with limited memory such that only R can fit in main memory; **neither** relation is sorted on the join key. Between *sort merge join* and *simple hash join* (Lecture 2), which would you pick? Please give two reasons to justify your choice.

b) **[8 points]** Assume we are joining two relations S ⋈ R.

Relation R: 20GB

Relation S: 100GB

Fudge factor = 2, memory = 10GB

Assuming initially both relations are stored on disk, how much data will be read from and written back to disk, respectively, in order to join the two relations using hybrid hash join? (Note that the final results need **not** be written back to disk)

c) **[8 points]** Consider two relations R and S stored in **column-major** format as described in lecture 7.

R(a, b, c, d) # 1 billion tuples, each attribute is 4B

S(x, y, z) # 5 billion tuples, each attribute is 4B

Both relations are on disk initially. Assume sufficient memory capacity and that the columns are **not** encoded. What is the disk IO traffic when performing the following join query? (Note that the final results need not be written back to disk)

```
SELECT a, b, x, y
FROM R, S
WHERE R.a = S.x;
```

d) **[8 points]** Suppose we operate on a distributed database with 4 nodes, with 20 GB storage capacity each. The database is responsible for storing two relations:

R: 10 GB

S: 40 GB

How would you store the two relations across the nodes to minimize the network traffic when joining the two relations? What is the network traffic when joining the two relations?

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

For part a) and part b), suppose we are performing a nested-loop join with relations R and S (S ⋈ R). Relation R is the inner relation.

R: 10k records; each record is 64B (cacheline size)

S: 100k records; each record is 64B (cacheline size)

The CPU cache size is 512KB. The cache is initially empty.

a) **[7 points]** What will be the cache miss rate when performing the join using the **Least Recently Used (LRU)** policy?

b) **[7 points]** What will be the cache miss rate when performing the join using the **Most Recently Used (MRU)** policy?

For part c) and part d), consider the join of three relations S ⋈ R ⋈ T. Suppose the relations are as follows where each tuple has size of 10B

Relation R (a, b): 1 billion tuples (10 GB), R.a is the primary key
Relation S (c, d): 5 billion tuples (50 GB), S.c is a foreign key referring to R.a, S.d is the primary key
Relation T (e, f): 2 billion tuples (20 GB), T.e is a foreign key referring to S.d

And the query is written as

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

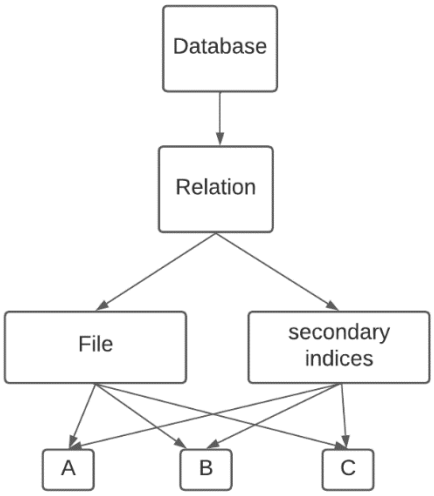
For simplicity, we assume all relations are in main memory when executing the query.

c) **[8 points]** Please draw all the possible query plans that can be executed in a pipelined fashion. Please always use the smaller relation as the inner relation. (Hint: You can consider the join order trees in P33, lecture 6.)

d) **[8 points]** For each query plan you draw above, what is the output size of each join operation?

Question 3. [30 points] Concurrency Control & Indexes

For part a) and part b), consider the following locking hierarchy for the Gray et al.'76 scheme (Lecture 9). A relation supports record referencing through a secondary index in addition to the primary index.



Additionally, consider the two following transactions using the representations from Berenson, et al.'95 (lecture 10)

T1: w1[A] w1[B]

T2: r2[B] w2[C] r2[B]

a) **[6 points]** What locks will T1 acquire if it runs alone under serializability? Please specify the lock type on each entity in the figure after T1 finishes issuing all requests.

b) **[6 points]** What locks will T2 acquire if it runs alone under serializability? Please specify the lock type on each entity in the figure after T2 finishes issuing all requests.

c) **[6 points]** Suppose both transactions T1 and T2 are executed with serializability guarantee. Assume T1 first issues the two write requests before T2 issues requests one by one. At which level in the figure will T2 first conflict with T1?

d) **[6 points]** Silo OCC replaces the global critical sections in conventional OCC (lecture 12) with per-record locks. Will this change ever lead to worse performance? Provide an example or a brief disprove to back your point.

e) **[6 points]** Consider the following data structures from Adaptive Radix Tree (ART). What would be one reason to design Node48 differently from Node4 and Node16?

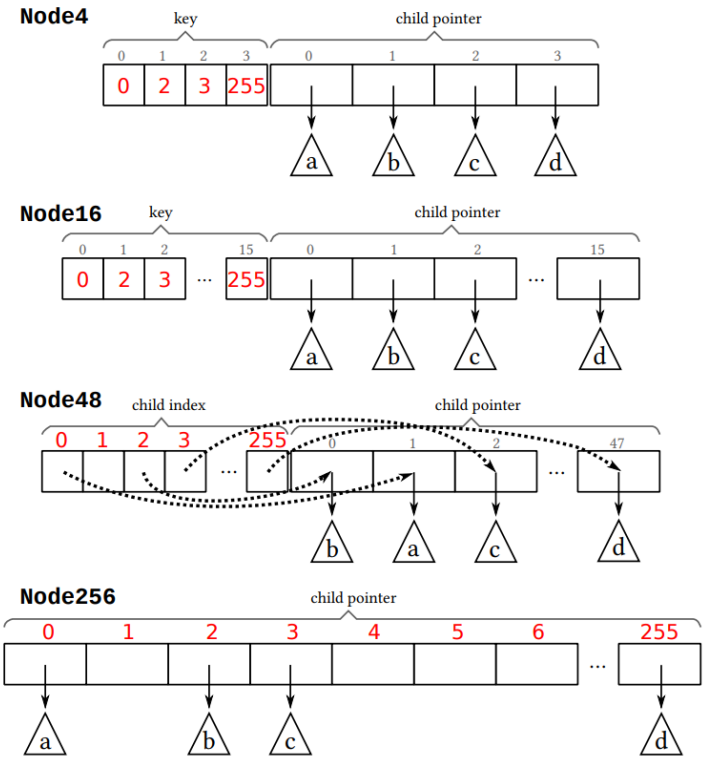


Fig. 5. Data structures for inner nodes. In each case the partial keys 0, 2, 3, and 255 are mapped to the subtrees a, b, c, and d, respectively.

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] T1 writes P2
3	[Update] T2 writes P5
4	Begin checkpoint
5	Transaction Table (T1, LastLSN=2), (T2, LastLSN=3)
6	Empty Dirty Page Table
7	End checkpoint
8	[Update] T1 writes P3
9	[Update] T2 writes P1
10	T2 commit
11	T2 end
12	T1 abort
13	CLR: Undo T1 LSN 8, undoNextLSN=2
14	
15	
16	
17	
18	
19	
20	
21	
22	
23	
24	

a) **[10 points]** Please fill in the contents of the Transaction Table and the Dirty Page Table after the analysis phase of the recovery process. (Hint: not all rows need to be filled in both tables)

Transaction Table

TransID	LastLSN

Dirty page table

PageID	RecLSN

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.

c) **[10 points]** ARIES have conservative designs for disk-based DBs with **No Force** and **Steal** policies. However, non-volatile memory DB may use **Force** with either **Steal** or **No Steal** policies. Namely, a transaction's modifications must be persisted when the transaction commits. Choose either **Force + Steal** or **Force + No Steal** as the new buffer management policy and write down at least one simplification that can be made to the original ARIES protocol.