EECS 484 Discussion 14

Topics Discussed so far...

- Disks, files and storage
  - Alternative 1/2/3
  - Clustered Index vs Unclustered index
  - B+ tree index vs Hash index
- Indexing
  - Insertion/deletion in B+ tree
  - Extendible hashing
  - Linear hashing
- Query Evaluation
  - Joins
- Query optimization
  - Query evaluation plan
  - RA equivalence
  - System R - left deep plans
  - System logs

Topics Discussed so far...

- Sorting
  - Formulas for #passes, ops
  - Optimizations: replacement sort, blocked I/O, double buffering
- Transactions and recovery
  - Serializability
  - Precedence graphs
  - Locks
  - WAL and ARIES
- Security and NoSQL
  - Access control policies
  - Discretionary access control
  - Mongo DB

Part I. Transactions

T1	T2	T3
R(A)		
W(A)		
	R(A)	
		R(B)
		W(B)
	R(B)	
	W(B)	
	COMMIT	
COMMIT		
		COMMIT

1. Is it recoverable?
2. Is it in ACA?
3. Is it possibly generated by 2PL?
4. Is it serializable?

Question 1 - Solution

1. Is recoverable? No
2. Is it in ACA? No
3. Is it possibly generated by 2PL? Yes
4. Is it serializable? Yes

Question 2

1. Why is writing to the log faster than simply flushing dirty pages upon commit? Give 2 reasons.

Question 2 - Solution

1. Why is writing to the log faster than simply flushing dirty pages upon commit? Give 2 reasons.
   1. Updates to log are smaller than full pages, even when considering the before/after images
   2. Log additions are sequential

Part II: Recovery

LSN LOG

1. start T1
2. start T2
3. update: T1 writes to P1
4. update: T2 writes to P1
5. start T3
6. update: T3 writes to P2
7. update: T1 writes to P1
8. update T2 writes to P2
9. T1 commit
10. T1 end
11. begin checkpoint
12. end checkpoint
13. update: T3 writes to P1
14. update: T2 writes to P3
15. update: T3 writes to P2
16. CRASH

Part II: Recovery

LSN LOG

1. start T1
2. start T2
3. update: T1 writes to P1
4. update: T2 writes to P1
5. start T3
6. update: T3 writes to P2
7. update: T1 writes to P1
8. update T2 writes to P2
9. T1 commit
10. T1 end
11. begin checkpoint
12. end checkpoint
13. update: T3 writes to P1
14. update: T2 writes to P3
15. update: T3 writes to P2
16. CRASH

a) Draw the transaction table and dirty page table after the analysis phase

a. Transaction Table(Include the transaction id and LSN columns)

TX_ID	LSN	Status
T3	6	U
T2	8	U

b. What is the dirty page table(Indicate the page id and LSN)

	Page ID	LSN
1	P1	3
2	P2	6

a. At what LSN will the ANALYSIS phase start processing the log?

LSN 11

b. At what LSN will the REDO phase start processing the log?

LSN 3

c. Imagine that you are building a new database system which implements ARIES-style recovery. Your testing reveals that recovery is taking far too long. What is one well-established way you can speed up the ANALYSIS phase?

Checkpoint more frequently

d. Again, you're building a new database system. You find a bug in the buffer manager that means dirty pages are rarely flushed. Which recovery phase --- ANALYSIS, REDO, OR UNDO --- will likely take a very long time to execute?

This will cause REDO to run slowly.

Part III: I/O Cost Optimization and Planning

- Consider a set of relations used by Yawlp.com to store restaurants and reviews:
  RESTAURANTS(restaurantid, name, zipcode)
  REVIEWS(reviewerid, restaurantid, rating, date)

- A restaurantid is 8 bytes, a name is 20 bytes, and a zipcode is 4 bytes.
- A reviewerid is 8 bytes, restaurantid is 8 bytes, rating is 4 bytes, and date is 4 bytes.

- There are 200 data pages for RESTAURANTS and 1200 data pages for REVIEWS.
- Each data page is 4 KB and is at 75% capacity.

- Zip codes are uniformly distributed between 0 and 99999 (inclusive). Ratings can be {1,2,3,4,5} with approximately 20% having each value

SELECT R.name FROM RESTAURANTS R ORDERBY R.zipcode;

This query can be executed by performing a sort of the tuples, followed by a projection on-the-fly. Assuming there is a 64KB buffer, please compute the cost of executing this query plan, including writing the final output relation. Further assume blocked I/O with a block size of 2 pages.

SORTING

Number of pages: 200

Sort cost: 2N * Num-passes

64K buffer with 4KB page size => buffer size B == 16 pages

block size = 2

When using non-blocked I/O, the num passes is **1 + ceiling($\log_{B-1}$ ceiling(N/B))**

When using blocked I/O, the num passes is **1 + ceiling($\log_{F}$ ceiling(N/B))**, where **F=floor(B/b)-1**

Block size b = 2

Using blocked I/O:

Number of passes == 1 + ceiling(logF ceiling(N/B)), where F == floor(B/b) – 1

F = floor(16/2) -1 == floor(8) -1 == 7

Thus, number of passes == 1 + ceiling($\log_7$ ceiling(N/B)) == 1 + ceiling($\log_7$ ceiling(200/16)) ==

1 + ceiling(log7 13) == 1 + 2 == 3

Using non-blocked I/O:

Number of passes == 1 + ceiling(logB-1 ceiling(N/B) == 1 + ceiling($\log_{16-1}$ ceiling(200/16)) ==

1 + ceiling($\log_{15}$ 13) == 1 + 1 == 2

Cost of sort == 2N * NUMPASSES

In case of blocked I/O: 2N * 3 == 2 * 200 * 3 == 1200

In case of non-blocked I/O: 2N * 2 == 2 * 200 * 2 == 800

- Now assume there are two indexes given to us: a hash index on zipcode, and an unclustered B+Tree on rating in REVIEWS.

- Additionally there are two clustered B+ trees: one on restaurantid for the RESTAURANT table, and one on reviewerid for the REVIEWS table.

- Assume that in all indexes we use “alternative 2” in which the payload of a B+Tree entry is a RID. RIDs take 28 bytes, and B+Tree leaf pages have a fill factor of 80%. Further assume that all of the internal nodes of B+Trees are already in RAM.

- Also assume the hash index has average bucket depth of 2.

b) How many tuples are there in RESTAURANTS? In REVIEWS?

# of tuples in RESTAURANTS

200 pages * 0.75 * 4KB / 32 == 19200 tuples.

# of tuples in REVIEWS

1200 pages * 0.75 * 4KB / 24 == 153600 tuples.

c) How many leaf pages are in the B+ tree index on rating? There are 153600 REVIEWS tuples. Each entry has 4 bytes for the rating, and 28 bytes for RID. Fill rate is 80%.

That means N tuples*size of data entry / (4KB * 0.8) == 153600*32/ 3.2*1024 == 1500 pages

SELECT R.name FROM RESTAURANTS R, REVIEWS W WHERE R.restaurantid = W.restaurantid AND R.zipcode = 48109 AND W.rating > 3;

The query plan selects tuples from R, then selects tuples from W, then joins the results using the Grace hash join. Finally, it performs the projection. The plan does not pipeline any results.

e) Compute the number of I/Os for selecting on R, including the intermediate write to disk. Be explicit about what index is chosen, if any.

SELECT R.name FROM RESTAURANTS R, REVIEWS W WHERE R.restaurantid = W.restaurantid AND R.zipcode = 48109 AND W.rating > 3;

The query plan selects tuples from R, then selects tuples from W, then joins the results using the Grace hash join. Finally, it performs the projection. The plan does not pipeline any results.

e) Compute the number of I/Os for selecting on R, including the intermediate write to disk. Be explicit about what index is chosen, if any.

There are 19200 tuples in R. Selection rate is 1/100,000. That means less than 1 tuple will be chosen on average. If there is a hash index on zipcode, this means just 1 I/O, to fetch the bucket page. 1 I/O to output it. Total == 3.

f) Please compute the number of I/Os for the final Grace Join.

After selection,

|R| = 1 page

|W| = 1200 * selectivity = 1200 * (⅖) = 480

Grace Hash complexity is 3(|R| + |W|)

|R| is 1 page

|W| is 480 pages

3 * (1 + 480) == 3 * 481 == 1443 I/Os