EECS 484 Discussion 13

Agenda

- Problems on Transactions
- Pointers on Project 4
- HW4 solutions

Part I. Transactions

Definitions….

- Recoverable schedule - Transactions commit only after all transactions whose changes they have read commit.
- Avoid cascading aborts (ACA) - Transactions read only the changes of committed transactions.

Problem 1

T1	T2
R(X)	
	W(X)
R(X)	
R(Y)	
W(Y)	
	R(Y)
	W(Y)
	COMMIT
COMMIT	

1. Is this schedule recoverable?
2. Does it avoid cascading aborts?
3. Could it be generated by 2PL?
4. Is it conflict-serializable?

Solution

1. Is this schedule recoverable? No
2. Does it avoid cascading aborts? No
3. Could it be generated by 2PL? No
4. Is it conflict-serializable? No

Problem 2

Give an example of a schedule of two transactions, with variables X and Y, that is recoverable but not ACA.

T1	T2
R(X)	
W(X)	
	R(X)
	W(X)
COMMIT/ABORT	
	COMMIT/ABORT

Problem 3 – Recovery

Consider the following log and the Aries recovery system.

LSN	LOG
21	begin checkpoint *Transaction Table:* **XACT** | **LastLSN** T1 | 12 *Dirty Page Table:* **Page** | **recLSN** P2 | 18
22	end checkpoint
23	UPDATE: T3 writes P1
24	UPDATE: T2 writes P4
25	UPDATE: T2 writes P2
26	COMMIT : T1
28	END : T1
	xxx CRASH xxx

Draw the Transaction Table and Dirty Page Table as they appear at the end of the Analysis phase

Consider the following log and the Aries recovery system.

LSN	LOG
21	begin checkpoint *Transaction Table:* **XACT** | **LastLSN** T1 | 12 *Dirty Page Table:* **Page** | **recLSN** P2 | 18
22	end checkpoint
23	UPDATE: T3 writes P1
24	UPDATE: T2 writes P4
25	UPDATE: T2 writes P2
26	COMMIT : T1
28	END : T1
	xxx CRASH xxx

Draw the Transaction Table and Dirty Page Table as they appear at the end of the Analysis phase

XACT table:

T3, 23 U

T2, 25, U

DPT:

P2, 18

P1, 23

P4, 24

Consider the following log and the Aries recovery system.

<table>
<tr><th>LSN</th><th>LOG</th></tr>
<tr><td>21</td><td>begin checkpoint
Transaction Table:
<table>
<tr><th>XACT</th><th>LastLSN</th></tr>
<tr><td>T1</td><td>12</td></tr>
</table>
Dirty Page Table:
<table>
<tr><th>Page</th><th>recLSN</th></tr>
<tr><td>P2</td><td>18</td></tr>
</table>
</td></tr>
<tr><td>22</td><td>end checkpoint</td></tr>
<tr><td>23</td><td>UPDATE: T3 writes P1</td></tr>
<tr><td>24</td><td>UPDATE: T2 writes P4</td></tr>
<tr><td>25</td><td>UPDATE: T2 writes P2</td></tr>
<tr><td>26</td><td>COMMIT : T1</td></tr>
<tr><td>28</td><td>END : T1</td></tr>
<tr><td></td><td>xxx CRASH xxx</td></tr>
</table>

Identify the ‘loser’ transactions, if any

Consider the following log and the Aries recovery system.

<table>
<tr><th>LSN</th><th>LOG</th></tr>
<tr><td>21</td><td>begin checkpoint
Transaction Table:
<table>
<tr><th>XACT</th><th>LastLSN</th></tr>
<tr><td>T1</td><td>12</td></tr>
</table>
Dirty Page Table:
<table>
<tr><th>Page</th><th>recLSN</th></tr>
<tr><td>P2</td><td>18</td></tr>
</table></td></tr>
<tr><td>22</td><td>end checkpoint</td></tr>
<tr><td>23</td><td>UPDATE: T3 writes P1</td></tr>
<tr><td>24</td><td>UPDATE: T2 writes P4</td></tr>
<tr><td>25</td><td>UPDATE: T2 writes P2</td></tr>
<tr><td>26</td><td>COMMIT : T1</td></tr>
<tr><td>28</td><td>END : T1</td></tr>
<tr><td></td><td>xxx CRASH xxx</td></tr>
</table>

Identify the 'loser' transactions, if any

T2, T3 are loser transactions

Consider the following log and the Aries recovery system.

LSN	LOG
21	begin checkpoint *Transaction Table:* **XACT** | **LastLSN** T1 | 12 *Dirty Page Table:* **Page** | **recLSN** P2 | 18
22	end checkpoint
23	UPDATE: T3 writes P1
24	UPDATE: T2 writes P4
25	UPDATE: T2 writes P2
26	COMMIT : T1
28	END : T1
	xxx CRASH xxx

Where (at which LSN) does the REDO phase of recovery begin?

Consider the following log and the Aries recovery system.

LSN	LOG
21	begin checkpoint *Transaction Table:* **XACT** | **LastLSN** T1 | 12 *Dirty Page Table:* **Page** | **recLSN** P2 | 18
22	end checkpoint
23	UPDATE: T3 writes P1
24	UPDATE: T2 writes P4
25	UPDATE: T2 writes P2
26	COMMIT : T1
28	END : T1
	xxx CRASH xxx

Where (at which LSN) does the REDO phase of recovery begin?

18

Consider the following log and the Aries recovery system.

<table>
<tr><th>LSN</th><th>LOG</th></tr>
<tr><td>21</td><td>begin checkpoint
Transaction Table:
<table>
<tr><th>XACT</th><th>LastLSN</th></tr>
<tr><td>T1</td><td>12</td></tr>
</table>
Dirty Page Table:
<table>
<tr><th>Page</th><th>recLSN</th></tr>
<tr><td>P2</td><td>18</td></tr>
</table>
</td></tr>
<tr><td>22</td><td>end checkpoint</td></tr>
<tr><td>23</td><td>UPDATE: T3 writes P1</td></tr>
<tr><td>24</td><td>UPDATE: T2 writes P4</td></tr>
<tr><td>25</td><td>UPDATE: T2 writes P2</td></tr>
<tr><td>26</td><td>COMMIT : T1</td></tr>
<tr><td>28</td><td>END : T1</td></tr>
<tr><td></td><td>xxx CRASH xxx</td></tr>
</table>

How many compensation log records (CLRs) are written during the UNDO phase of recovery?

Consider the following log and the Aries recovery system.

LSN	LOG
21	begin checkpoint *Transaction Table:* **XACT** | **LastLSN** T1 | 12 *Dirty Page Table:* **Page** | **recLSN** P2 | 18
22	end checkpoint
23	UPDATE: T3 writes P1
24	UPDATE: T2 writes P4
25	UPDATE: T2 writes P2
26	COMMIT : T1
28	END : T1
	xxx CRASH xxx

How many compensation log records (CLRs) are written during the UNDO phase of recovery?

3

Part 2. Project 4

Project 4 Testcases

```
StorageEngine/sampleDBFile.txt
1 write 5 0 one
2 write 3 0 two
2 write 1 0 three
1 write 13 0 four
2 commit
1 abort
checkpoint
end
```

Project 4 Testcases

```
2       -1      1       update  5       0       xxx             one
3       -1      2       update  3       0       xxx             two
4       3       2       update  1       0       xxxxx   three
5       2       1       update  13      0       xxxx            four
6       4       2       commit
7       6       2       end
8       5       1       abort
9       8       1       CLR             13      0       xxxx            2
10      9       1       CLR             5       0       xxx
                -1
11      10      1       end
12      -1      -1      begin_checkpoint
13      12      -1      end_checkpoint  {}
                    { [ 1 4 ] [ 3 3 ] [ 5 2 ] [ 13 5 ]}
```

Some hints

How to write test cases?

- Write a case for every method you implement. Eg: Checkpoint
- Write a case for a combination of multiple methods. Eg: Checkpoint+abort
- Write case which require page flushes

Part 3. HW4 solutions

Question 1

- For this question, you will consider sorting a large dataset on a modern desktop machine. Suppose that your dataset consists of a 900GB file and that you have reserved 4GB of memory (RAM) to assist in the sorting. Suppose that your system uses a page size of 16 KB. In this question, 1GB = 1024 * 1024 * 1024 Bytes, 1KB = 1024 Bytes.
- How many passes will be required to sort this dataset?

(For the purposes of this question, assume that you do not use optimizations such as double buffering or blocked I/O.) Use the scheme in Section 13.3 of the book. *Show your work to receive full credit.*

Solution

a) How many passes will be required to sort this dataset?

- Data size = 900GB
- N = # pages in the file = 900 * 1024 * 1024 KB / 16 KB
  = 58,982,400 [size of one page = 16 KB]

- B = # pages that can fit in RAM
  = 4 GB * 1048576 KB/GB / 16 KB/page = 262144 or 2^18
- # passes = 1 + ceil($\log_{B-1}(N/B)$) = 2

Solution

b) How many total I/Os (page reads and writes) will be required to perform the sort?

- We read and write back the entire file once per pass.
- #I/Os = 2N * #passes = 2N * 2 = 235,929,600

Solution

c) How many passes will be necessary if we use a blocking scheme (Section 13.4.1) where we use a buffer block size of b = 64KB (which is 64 x 1024 Bytes) and replacement sort is used for the initial pass?

- q = Buffer size / page size = 64 / 16 = 4 pages
- F = floor(B / q - 1) = 262144 / 4 - 1 = 65535
- #runs after pass 0 = ceil(N/2(B-2)) = ceil(N/2B) = 113
- #subsequent passes = ceil($\log_F$ (113)) = 1
- Total number of passes = 2

Question 2

- Consider a table *Students*, where there is a B+ tree on attribute *gpa*, and the following query:
  - SELECT * FROM *Student* ORDER BY *gpa*
- Furthermore, consider the following parameters:
  - height of the B+ tree: 3 (2 non-leaf index nodes and 1 leaf index node at each path)
  - # of index leaf pages: 400
  - # of data record pages: 6,000
  - # of tuples per page: 50
- Given that the first level (root) of the B+ tree is kept in memory at all times, provide the cost of executing the above query by using the B+ tree (showing your work) when:

Solution

a) The index is clustered.

- Total cost = Cost to get to the first leaf node + Cost to scan all leaf pages + Cost to scan all data record pages
- Since the index is clustered, we read each data record page exactly once.
- The cost to scan all data record pages = Total number of pages in *Student*
- Total cost = (3-1) + 400 + 6000 = 6402

Solution

b) The index is not clustered.

- Total cost = Cost to get to the first leaf node + Cost to scan all leaf pages + Cost to scan all data record pages
- Since the index is not clustered. In the worst case, we read a different page for each tuple.
- Total cost to scan all data record pages = Total number of tuples in *Student* (why?) = 50 * 6000
- Total cost = (3-1) + 400 + 300,000 = 300,402

Solution

c) Would you consider doing an external sort instead of using the B+ tree? Justify.

- Cost of external sort = 2 * 6000 * #passes
- Cost is atleast 12,000
- If a clustered index is present, it is much faster to just use that.
- If no clustered index is present, in general, external sorting is much faster (we will need ~50 passes for the sorting cost to be comparable to unclustered index scan cost)
- Alternate answer: When the selectivity is high, an external sort will perform better than using an index

Question 3

- For this question, suppose that you are given two relations, R and S, and that you want to perform a natural join between R and S. Suppose that R contains 50,000 records on 5000 pages (||R|| = 50,000; |R| = 5,000), and S contains 150,000 records on 15,000 pages (||S|| = 150,000; |S| = 15,000).

Solution

Block Nested-Loops Join

Suppose that you have B = 1002 buffer pages available. You may assume that R is the outer relation.

How many (a) total reads and (b) total writes are required (ignoring the cost of writing the final result)?

- Reads = |R| + |S| * ceil(|R|/(B-2))
  = 5,000 + 15,000 * ceil(5000/1000)
  = 5,000 + 75,000 = 80,000
- No writes are required for nested-loops join

Solution

Grace Hash Join

How many total reads are required?

- 2(|R| + |S|) = 2 * (5000 + 15,000) = 40,000

How many total writes are required (ignoring the cost of writing the final result)?

- 1(|R| + |S|) = 20,000

Solution

Now, for your new device, suppose that you are given the relative performance of reads and writes. That is, you know X, such that X = cost(write) / cost(read). Given your analysis above, for which values of X would you choose the block nested-loop join? What about the hash join? *Please explain your answer and show your work.*

- Let R denote the cost of a read, and let W denote the cost of a write. X = W/R
- Cost(BNL) = 80,000R
- Cost(GHJ) = 40,000 + 20,000*X
- BNL cheaper is cheaper if:
  - Cost(BNL) < Cost(GHJ)
  - 80,000R < 40,000 + 20,000*X => X = W/R > 2

Question 4

- Consider the following relational schema and SQL query:

Students(sid, sname, gpa)
Takes(sid, cid)
Class(cid, cname, ctype)

```
SELECT S.sname, C.cname
FROM Students S, Takes T, Class C
WHERE S.sid = T.sid AND T.cid = C.cid AND S.gpa > 3.5
    AND C.ctype = 'Social'
```

Solution

4 (a) How many different join orders does a System R style query optimizer consider (assuming cross-products are disallowed) when deciding how to process the given query? List each of these join orders.

- 4 in all, only left-deep join orders are considered

Join(Join(S, T), C)

Join(Join(T, S), C)

Join(Join(C, T), S)

Join(Join(T, C), S)

Solution

4 (b) What potential indexes match this query? List each of those indexes (some may consist of multiple attributes).

- A clustered index on Students.gpa might help with the range search.
- An index on Class.ctype might help with the equi-conditon.
- Clustered indexes on S.sid, T.sid, T.cid, C.cid might allow us to use the index to retrieve sorted data to input into a sort-merge join.

Question 5

For the above query in Question 1, consider a query plan that works as follows:
Join(Join(Selection(S), T), Selection(C)) with projections cascaded and pushed in as far as possible.

We use Grace Hash Join for both joins, a clustered B+Tree index with search key of GPA to access S, an unclustered hash index with search key of ctype to access C, and a scan to access T. Make the following assumptions:

The B+Tree is of height 2 with 40 leaf nodes, and with only the leaf nodes on disk (i.e., the internal nodes are stored in RAM).

The GPA is stored to 2 decimal places (e.g., 3.26) and that there are 400 distinct values for the GPA attribute, that the lowest GPA attribute is 0.00, and the highest is 4.00.

The ctype attribute takes 50 distinct values.

Assume each attribute in each relation is 8 Bytes and each RID is 24 Bytes.

Assume page size of 4Kbytes and a memory buffer size of 5 pages.

Assume that double buffering is not used. I/O block size is just one page.

Assume 20K tuples in the Class relation, and 300K tuples in the Takes relation.

Assume all indexes use alternative 2.

Assume a fill factor of 0.75 for each page in the B-tree.

Assume that all hash functions used divide the data set so that the largest hash bucket is 10% larger than the average.

For each join, you may choose either operand as inner, whichever results in lower cost, and you should pipeline the plan execution as far as possible.

Solution

- Step 1: Compute the number of pages in each relation
  - Size of each B-Tree index entry = 32 bytes (8B key + 24B RID).
  - # Entries per index page = fill factor * entries per page
    = 0.75 * 4096/32
    = 96 entries per page
  - Total entries in 40 index pages = 40 * 96
    = 3840
    = Total tuples in S
  - Tuple size of S is (3 attributes) 24 bytes
  - # pages of S = (3840 * 24) / 4096 = 23 pages
  - # pages of T = (300 *1024 * 16) / 4096 = 1200 pages

Solution

- Step 2: Cost to scan S
  - Based on gpa range, we estimate that 0.5/4 of the index pages and data pages will be accessed in S
  - Read cost = 0.5/4 (40 + 23) = 8 I/O
  - Materializing Selection(S)
    - Only S.sid and S.sname are required for later operations, so each tuple is 16B
    - # tuples selected = 1/8 * 3840 = 480
    - # pages = 480 / (4096/16) = 2 pages
    - Write cost = 2 I/O
  - Total Cost = 8 + 2 = 10 I/O

Solution

- Step 3: Cost of Grace Hash Join between S and T
  - Read cost: 2 (|S| + |T|) = 2 * (2 + 1200) = 2404
  - Write cost: |S| + |T| = 2 + 1200 = 1202