6.830 Problem Set 3 (2014)

Assigned: Monday, October 27, 2014

Due: December 1, 2014, 11:59 PM
Submit to Stellar: `https://stellar.mit.edu/S/course/6/fa14/6.830/homework/)`

Revision history: Nov 20, 2014 – Updated Problem 3 to replace `WRITE A, x2 + x1` with `WRITE A, x2 + y2` in T2.

The purpose of this problem set is to give you some practice with concepts related to recovery, parallel query processing, two-phase commit, and other papers we read during the second half of the course.

1 Questions

1. [5 points]: Below, we provide 3 workloads. Each one contains several concurrent transactions (consisting of READ and WRITE statements on objects). For each workload, several possible execution interleavings are given. Your job is to indicate whether, for each of the interleavings:

- The interleaving has an equivalent serial ordering. If so, indicate what the serial ordering is.
- Whether the interleaving would be valid using lock-based concurrency control. Assume that locks, when needed, are acquired (with the appropriate lock mode) on an object just before the statement that reads or writes the object and that all locks are released during the COMMIT statement (and no sooner.) If the interleaving is not valid, indicate whether or not it would simply never occur or would result in deadlock, and the time when the deadlock would occur.
- Whether the ordering would be valid using optimistic concurrency control. If not, indicate which transaction will be aborted. Assume the use of the Parallel Validation scheme described in Section 5 of the Optimistic Concurrency Control paper by Kung and Robinson, and that the validation and the write phases of optimistic concurrency control happen during the COMMIT statement (and no sooner.)

Assume in all cases that written values can depend on previously read values. (The workloads are shown on the next page.)

Answer:

- **Interleaving 1:** T1, T2 is an equivalent serial order. This would not be valid under locking – T2 would not be allowed to WRITE B until after T1 had committed. This is a valid schedule under OCC, since the write set of T1 does not intersect the read set of T2, and T1 has committed before T2 starts to commit (enters its write phase).
- **Interleaving 2:** T1, T2 is an equivalent serial order, since all conflicting operations in T1 are ordered before all conflicting operations in T2. This schedule would not be allowed under either locking or OCC. In OCC, T1 will be aborted since its read set intersects T2s write set.
- **Interleaving 3:** There is no equivalent serial ordering. OCC would roll back T1. In locking, deadlock would result at time 6, since T3 cannot get an exclusive lock on A because T1 holds a shared lock on A.
- **Interleaving 4:** T2, T1 is an equivalent serial ordering. This would not be allowed under locking, since T1 would not be allowed to read B after T2 had written it. OCC allows this schedule, since T2's read set does not intersect T1's write set.
- **Interleaving 5:** T2, T1 is an equivalent serial ordering. This schedule is allowed by locking. T1 would abort in OCC, since it reads data that T2 wrote and the read phase of T1 overlapped the read and write phase of T2.

Workload 1

Transaction 1	Transaction 2
READ A	READ A
READ B	WRITE B
WRITE C	WRITE A

Interleaving 1:

```
1 T1:  READ A
2 T2:  READ A
3 T1:  READ B
4 T1:  WRITE C
5 T2:  WRITE B
6 T1:  COMMIT
7 T2:  WRITE A
8 T2:  COMMIT
```

Interleaving 2:

```
1 T1:  READ A
2 T1:  READ B
3 T2:  READ A
4 T2:  WRITE B
5 T2:  WRITE A
6 T2:  COMMIT
7 T1:  WRITE C
7 T1:  COMMIT
```

Workload 2

Transaction 1	Transaction 2	Transaction 3
READ A	READ A	READ A
WRITE A	WRITE B	WRITE A

Interleaving 3:

```
1 T1:  READ A
2 T2:  READ A
3 T3:  READ A
4 T2:  WRITE B
5 T2:  COMMIT
6 T3:  WRITE A
7 T3:  COMMIT
8 T1:  WRITE A
9 T1:  COMMIT
```

Workload 3

Transaction 1	Transaction 2
WRITE A	WRITE B
READ B	READ C

Interleaving 4:

```
1 T1:  Write A
2 T2:  Write B
3 T1:  Read B
4 T2:  Read C
5 T1:  Commit
6 T2:  Commit
```

Interleaving 5:

```
1 T1:  Write A
2 T2:  Write B
3 T2:  Read C
4 T2:  Commit
5 T1:  Read B
6 T1:  Commit
```

2. [3 points]: If your DBMS never STOLE pages (as discussed in class and in the paper by Franklin), how would that affect the design of the database recovery manager?

Answer: With a NO-STEAL policy, dirty pages will never be written to disk. Thus, there is no need for UNDO during recovery. REDO is still needed as some committed transactions may not have flushed all of their data to disk. We will still need the ability to UNDO a transaction that ABORTs – thus we will need an in-memory log.

3. [3 points]: Suppose you are told that the following transactions are run concurrently on a (locking-based, degree 3 consistency) database system that has just been restarted and is fully recovered. Suppose the system crashes while executing the statement marked by an "***" in Transaction 1. Suppose that Transaction 2 has committed, and the state of Transactions 3 and 4 are unknown (e.g., they may or may not have committed.) Assume that each object (e.g., X, Y, etc.) occupies exactly one page of memory.

Trans 1:	Trans 2:	Trans 3:	Trans 4:
x1 = READ X	WRITE Y, 0	z3 = READ X	a4 = READ A
WRITE X, x1 + 1	WRITE B, 0	a3 = READ A	z4 = READ Z
*** y1 = READ Y	x2 = READ X	WRITE A, a3 + 10	WRITE B, (a4-z4)
WRITE Y, y1 + x	WRITE Z, x2	z3 = READ Z	
	y2 = READ Y	WRITE Z, z3 - 10	
	WRITE A, x2 + y2		

A. Show an equivalent serial order that could have resulted from these statements, given what you know about what statement was executing when the system crashed. In addition, show an interleaving of the statements from these transactions that is equivalent to your serial order; make sure this serial order could result from a locking-based concurrency control protocol (again assuming that locks are acquired immediately before an item is accessed and released just before the commit statement.)

Answer: Any ordering in which T2 precedes T1, e.g., T2, T4, T1, T3 or T2, T1, T4, T3.

A resulting interleaving might be (there are many possible correct answers):

T2 WRITE Y, 0
T2 WRITE B, 0
T2 x2 = READ X
T2 WRITE Z, x2
T2 y2 = READ Y
T2 WRITE A, x2 + x1
T4 a4 = READ A
T1 x1 = READ X
T1 WRITE X, x1 + 1
T1 ***

B. Show all of the records that should be in the log at the time of the crash (given your serial order), assuming that there have been no checkpoints and that you are using an ARIES-style logging and recovery protocol. Your records should include all of the relevant fields described in Section 4.1 of the ARIES paper. Also show the status of the transaction table (as described in Section 4.3 of the ARIES paper) after the analysis phase of recovery has run.

Answer:

Given the above interleaving, the log contains:

LSN	Type	Tid	PrevLSN	PageID	Data
1	BEGIN	1			
2	BEGIN	2			
3	BEGIN	3			
4	BEGIN	4			
5	UPDATE	2	2	Y	OLD = ?, NEW = 0
6	UPDATE	2	5	B	OLD = ?, NEW = 0
7	UPDATE	2	6	Z	OLD = ?, NEW = x2
8	UPDATE	2	7	A	OLD = ?, NEW = x2 + x1
9	COMMIT	2	8		
10	UPDATE	1	1	X	OLD = ?, NEW = x1 + 1

After recovery, the transaction table contains:

Tid	lastLSN
1	10
3	3
4	4

C. Suppose you have 2 pages of memory, and are using a STEAL/NO-FORCE buffer management policy as in ARIES. Given the interleaving you showed above, for each of the 5 pages used in these transactions, show one possible assignment of LSN values for those pages as they are on disk before recovery runs. You should use the value "?" if the LSN is unchanged from the prior state of the page before these transactions began. Finally, indicate which pages will be modified during the UNDO pass, and which will be modified during the REDO pass.

Answer:

To determine which pages are dirty, we need to model the state of the buffer pool. Here we assume that an LRU policy is used for eviction.

Stmt	Buffer Pool	Dirty	Action
W Y	Y	Y	
W B	Y B	Y B	
R X	B X	B	Flush Y (LSN 5)
W Z	X Z	Z	Flush B (LSN 6)
R Y	Z Y	Z	
W A	Y A	A	Flush Z (LSN 7)
R A	Y A	A	
R X	X A	A	
W X	X A	X A	

Thus, the page table for the above interleaving would look like:

Page	Page LSN	Modified by REDO?	Modified by UNDO?
A	?	Y	N
B	6	N	N
X	?	Y	Y
Y	5	N	N
Z	7	N	N

2 Parallel Query Processing

The standard way to execute a query in a shared-nothing parallel database is to horizontally partition all tables across all nodes in the database. Under such a setting, distributed joins can be computed by repartitioning tables over the join attributes.

```
SELECT *
FROM R, S
WHERE R.a > v1 AND S.b > v2 AND R.c = S.d
```

You are given the following

- Both tables are 6,000 MB,
- The disk can read at 50 MB/sec (for the purposes of this problem, you may ignore differences between sequential and random I/O),
- The network on each node can transmit data at 40 MB/sec, regardless of the number or rate at which other nodes are simultaneously transmitting,
- A computer cannot send over the network and read or write from its disk at the same time,
- The selectivity of both selection predicates is 0.05,
- Each tuple in R joins with exactly one tuple in S,
- Each machine in your distributed database has 300 MB of memory.

Suppose both tables are stored on a single node.

4. [3 points]: Describe the best query plan for executing this query on that node.

Answer:

1. Perform a sequential scan of R. After filtering, there is 6000 MB * 0.5 = 300 MB of data.
2. Perform a sequential scan of S. After filtering, there is 6000 MB * 0.5 = 300 MB of data.
3. Since both tables are the same size and fit in memory after filtering, we can use either one as the inner relation and do an in-memory hash join.

5. [2 points]: Ignoring CPU costs, estimate the time to answer this query on a single node.

Answer: Time for query = time to scan R + time to scan S = (6000 MB / 50 MB/s) + (6000MB / 50MB/s) = 240 sec

Now, suppose that the tables are hash-partitioned on R.a and S.b across a 4 node distributed database.

6. [3 points]: Describe the best distributed query plan for executing this query.

Answer: In the following, we assume that one of the 4 nodes is a coordinator, and the other 3 are workers. At the end of the transaction, the coordinator receives results from the workers and merges the results.

1. At each node, scan and filter the stored partitions for R and S.
2. Since R.a and S.b are not join attributes, we need to repartition the tables on the join attribute. Each node has 150 MB of filtered data (75 MB of R and 75 MB of S). The best strategy is to make 6 logical partitions per table (each of size 50 MB) on the attributes R.c and S.d.

- The coordinator receives 3 partitions of R and 3 partitions of S totaling 300 MB, so it can perform an in-memory hash join.
- The other three nodes each receive one partition of R and one partition of S totaling 100 MB, so they can perform in-memory hash joins as well.
- If R.a and S.b are completely independent of R.c and S.d, then each worker node will need to send 5/6 of its data to other nodes, and the coordinator will need to send 1/2 of its data to other nodes.

3. Finally, the worker nodes send their join results to the coordinator, and the coordinator merges the results.

We also accepted answers in which each node sends equal amounts of data to each other node, but that approach is slightly slower than the one outlined above (see the next question).

7. [2 points]: Ignoring CPU costs, estimate the time to answer this query on the distributed database.

Answer: Time for query = time to scan partitions of R and S at each node (in parallel) + time to transmit at most 5/6 of filtered data from each node (in parallel) + time to perform joins (negligible since we are ignoring CPU costs) + time for worker nodes to send results to the coordinator (in parallel)

= ((6000 MB + 6000 MB)/4) / 50 MB/s + (150 MB * 5/6)/ 40 MB/s + 100 MB / 40 MB/s = 60 sec + 3.125 sec + 2.5 sec = 65.625 seconds

If we instead sent equal amounts of data between all nodes, the time for the query is as follows:

Time for query = time to scan partitions of R and S at each node (in parallel) + time to transmit 3/4 of filtered data from each node (in parallel) + time to perform joins (negligible since we are ignoring CPU costs) + time for worker nodes to send results to the coordinator (in parallel)

= ((6000 MB + 6000 MB)/4) / 50 MB/s + (150 MB * 3/4)/ 40 MB/s + 150 MB / 40 MB/s = 60 sec + 2.813 sec + 3.75 sec = 66.563 seconds

Now, suppose that the tables are hash-partitioned on R.c and S.d across a 4 node distributed database.

8. [3 points]: Describe the best distributed query plan for executing this query.

Answer:

1. At each node, scan and filter the stored partitions for R and S.
2. Since the tables are already hash partitioned on the join attributes, we do not need to repartition them. The filtered tables on each node fit in memory (150 MB total per node), so joins can be performed in memory.
3. Worker nodes send their join results to the coordinator, and the coordinator merges the results.

9. [2 points]: Ignoring CPU costs, estimate the time to answer this query on the distributed database.

Answer: Time for query = time to scan partitions of R and S at each node (in parallel) + time to perform joins (negligible since we are ignoring CPU costs) + time for worker nodes to send results to the coordinator (in parallel)

= ((6000 MB + 6000 MB)/4) / 50 MB/s + 150 MB / 40 MB/s = 60 sec + 3.75 sec = 63.75 seconds

3 Two Phase Commit

Recall that the two-phase commit protocol is used to process transactions over a distributed database on several nodes. In this set of questions, suppose you are running the following three transactions over five values, A, B, C, D, and E, which initially

have value 0. Assume the use of the basic (e.g., not presumed commit or presumed abort version) of the protocol described in the paper "Transaction Management in the R* Distributed Database Management Systems" by C. Mohan et al.

```
T1:
WA(1)
WB(2)
WE(3)
COMMIT

T2:
WB(4)
WC(5)
WE(6)
COMMIT

T3:
WC(7)
WD(8)
WE(9)
COMMIT
```

Here, the notation WA(1) means "write value 1 to A". You run these three transactions concurrently on three workers, W1, W2 and W3, as well as a coordinator, C. Data item A and C are on W1, B and D are on W2, E is on W3. The system crashes during the execution, and you know that each transaction has begun but do not know how far into its execution it has proceeded. Before recovery begins, you observe the a small piece of the logs on one or more of the workers and the coordinator. Based on just the information in each set of partial logs shown below, indicate whether each transaction has already or definitely will commit or abort, or whether its outcome is unknown.

10. [3 points]:
W1 log:

```
...
PREPARE T1 -- VOTE YES
...
```

W2 log:

```
...
T3 UP D
COMMIT T3
...
```

W3 log:

```
...
T2 UP E
PREPARE T3 -- VOTE YES
COMMIT T2
...
```

Answer:

1. T1 is unknown. Although W1 voted yes for T1, W2 and W3 may vote no.

2. T2 committed.
3. T3 committed.

There may be other log entries before and after those shown here. The notation "T2 UP E" means that T2 updated value E. Indicate whether each of T1, T2, or T3 will commit, abort, or its outcome is unknown.

11. [2 points]: C Log:

```
...
COMMIT T3
...
```

W2 Log:

```
...
PREPARE T1 -- VOTE NO
...
ACK T2
...
ACK T3
....
```

Indicate whether each of T1, T2, or T3 will commit, abort, or its outcome is unknown.

Answer:

1. T1 will abort.
2. T2 is unknown.
3. T3 will commit.