Written Exam
Database Transactions and Processes
course code: 192110982

14 April 2011 (08:45 - 12:15), CR-2G

Maurice van Keulen

Remarks:

- Motivate yours answers. The motivation / argumentation plays an important role in grading the assignments.
- You may not consult books or notes, but only one page of A4 size, double-sided printed. The page may contain text (typed or hand-written) and (possibly reduced) images (copied from the book, other sources or hand-made).
- For each assignment, the number of points is given. They add up to 90. You get 10 points for showing up at the exam. The grade for the exam is determined by dividing the number of points by 10.
- There is also a practical assignment. The final grade for the course is determined by taking twice the grade of the exam and once the grade of the practical assignment.

Question 1: 2-Phase Commit (20 points)

Suppose we run a distributed transaction on three servers A, B, and C. A transaction manager running on server M is the coordinator for the 2-Phase commit protocol.

Suppose M started the 2-Phase Commit protocol, it has sent out the prepare messages to the three cohorts, they all received it, they all sent back their votes to commit, the coordinator received the votes to commit messages from A and B, but not from C, and then M crashed.

(a) Suppose it takes some time for the coordinator to come back on-line. Explain what actions the time-out protocol prescribes for a cohort to take when it time-outs waiting for a response from the coordinator.

(b) Explain what the coordinator would find in its log after the crash.

(c) Explain what actions the restart protocol prescribes for the coordinator to take right after it is brought back on-line again.

(d) What is the end result of these events: global commit or global abort? Explain your answer.

Question 2: Recovery (20 points)

Suppose the database crashed and upon restart we find the following situation. The database uses a *no-force commit policy*, pessimistic concurrency control, and sharp checkpointing. The database schema contains an integrity constraint: $C: x \neq y$.

Log:

	6	7	8	9	10	11	12	13	14	15	16	17
...	T_1	T_1	T_2	T_2	$T_1; T_2$	T_2	T_2	T_1	T_3	T_1	T_1	T_3
	B	U	B	U	CK	U	A	U	B	U	C	U
		p		q		q		x		y		x
		0 1		0 1		1 0		5 10		10 5		10 11

Each record from top to bottom:
LSN, transaction id, record type, database variable, before and after image
Type: B:begin transaction, U:update, C:commit, A:abort, CK:checkpoint.

Database pages:

Page 3
LSN:14
x=10
y=10

Page 8
LSN:10
p=1
q=1

(a) The 'C' and 'D' in 'ACID' guarantee consistency and durability. Database page 3 on disk, however, contains an inconsistent state, because it violates integrity constraint C. Does the above information contain an error, i.e., is it impossible to find the above situation on disk after a crash? If so, what is wrong in your opinion? If not, how are consistency and durability guaranteed? Explain your answer.

(b) The recovery protocol reconstructs a consistent database state. Present the steps of the protocol for obtaining a consistent database state in this situation. Also give the reconstructed database state (i.e., values of the variables x, y, p, and q in pages 3 and 8). Explain your answer.

(c) At LSN 12, transaction T_2 aborted. The log record with LSN 11 is, in fact, part of the rollback for this transaction. Suppose that this abort was not upon request by the application, but that the DBMS itself aborted the transaction. What is the most likely reason for the DBMS to have aborted this transaction given the information above? Explain your answer.

Question 3: Serializability and Isolation Levels (15 points)

Suppose we have the two transactions below. T_1 swaps the values of the two database variables db_x and db_y, and T_2 increases db_x by one, and decreases db_y by one. v, v_1 and v_2 are local variables in the application program (i.e., they are transient and not stored in the database).

$$T_1 \ : \ \text{BEGIN}; \underbrace{v_1 := db_x}_{r_1(x)}; \underbrace{v_2 := db_y}_{r_1(y)}; \underbrace{db_x := v_2}_{w_1(x)}; \underbrace{db_y := v_1}_{w_1(y)}; \text{COMMIT}$$

$$T_2 \ : \ \text{BEGIN}; \underbrace{v := db_x}_{r_2(x)}; \underbrace{db_x := v + 1}_{w_2(x)}; \underbrace{v := db_y}_{r_2(y)}; \underbrace{db_y := v - 1}_{w_2(y)}; \text{COMMIT}$$

(a) Give all possible schedules that are *conflict equivalent* with the schedule $T_1; T_2$. Explain your answer.

(b) Are the schedules of Question 3(a) *serializable*? Explain your answer.

Suppose initially $db_x = 5$ and $db_y = 10$, the database uses immediate-update pessimistic concurrency control with row-level locking, and both transactions run concurrently in isolation level "SERIALIZABLE". Because they run concurrently, it is not clear how the individual operations of both transactions are interleaved.

(c) What are all possible outcomes (in terms of values for db_x and db_y) after running these two transactions? Explain your answer.
(Tip: there are more than two possible outcomes!)

Question 4: Replication (15 points)

Given an Asynchronous-Update replication system with 4 replicas based on *multimaster replication* (also called, peer-to-peer or group replication).

(a) Suppose there is a variable x in this replicated database and its value is $x = 0$ in all replicas. We are now going to increase x three times in the following way: we do a read operation for x followed by a write operation of x with the value $x + 1$. Hence in total, we do 6 operations and for each a different replica is selected:

- First read $v = r(x)$ on replica 3
- First write $w(x := v + 1)$ on replica 3
- Second read $v = r(x)$ on replica 2
- (Effects of first write propagated to other servers)
- Second write $w(x := v + 1)$ on replica 2
- Third read $v = r(x)$ on replica 4
- (Effects of second write propagated to other servers)
- Third write $w(x := v + 1)$ on replica 4
- (Effects of third write propagated to other servers)

Since we do *asynchronous* updating, it takes some time for the updates to propagate. Suppose this happens as indicated above.

Give the value of x in each replica after each operation (which is a matrix with 9 rows and 4 columns). Explain your answer.

(b) In a setting with replicas, we may concurrently read and write the different replicas at the same time. Suppose all replicas contain $x = 0$ again and we perform the same operations again but now each time for two servers at once in the following way:

- First read $v_1 = r(x)$ on replica 1 and read $v_2 = r(x)$ on replica 2
- First write $w(x := v_1 + 1)$ on replica 1 and write $w(x := v_2 + 1)$ on replica 1
- Second read $v_1 = r(x)$ on replica 2 and read $v_2 = r(x)$ on replica 3
- (Effects of first set of writes propagated to other servers)
- Second write $w(x := v_1 + 1)$ on replica 2 and write $w(x := v_2 + 1)$ on replica 3
- (Effects of second set of writes propagated to other servers)

Give the value of x in each replica after each operation. Explain your answer.

Question 5: Chained transactions (20 points)

Imagine a typical DBMS with immediate-update pessimistic concurrency control. Given a transaction T running in isolation level SERIALIZABLE with three operations a, b, c. Suppose the application developer decided to use chained transactions for each, i.e.,

$$T : \underbrace{\text{BEGIN}; a; \text{COMMIT}}_{T_1}; \underbrace{\text{BEGIN}; b; \text{COMMIT}}_{T_2}; \underbrace{\text{BEGIN}; c; \text{COMMIT}}_{T_3}$$

(a) What is the most likely reason for the application developer to choose for chained transactions instead of one single transaction? Explain your answer.

(b) Which of the 'A', 'C', 'I', and 'D' properties are still preserved for chained transactions? Explain your answer.

(c) Another choice could be to run the three operations as a single transaction at the lowest isolation level READ UNCOMMITTED.

i) Explain the difference in locking behaviour.

ii) Which of the two choices do you expect to run faster?

iii) What are the advantages or disadvantages of chained transactions vs. a single transaction at READ UNCOMMITTED?

Explain your answers.