CS245 Final Exam Solutions – Summer 1999

Part I. (18 points) Indicate whether each of the following statements is TRUE or FALSE. If you chose FALSE, please *correct* it or briefly *explain why.*

1. TRUE.

2. FALSE. The real world actions in a transaction should be executed *after* the transaction commits.

3. TRUE.

4. TRUE.

5. TRUE.

6. FALSE. We can assign votes to each data copy. Then, each write action only needs to write enough copies to cover more than half of the total votes.

7. FALSE. The more views we maintain, the better the query performance, but the higher the update and storage cost. There is a tradeoff.

8. TRUE.

9. FALSE. Double buffering reduces the total processing time from $n(R+P)$ to $nP+R$, when R is less than P. (R represents the time to read a block; P represents the time to process a block; n represents the number of requests.)

Part II. (32 points) Circle the *one* correct answer from the listed multiple choices. Points $= 4 \times$ number right $- 1 \times$ number wrong.

1. **B.** With undo logging or undo/redo logging, if we omit the ABORT record, we may repeat the undo process unnecessarily. For redo logging, however, we do not need to do anything for a transaction as long as there is no COMMIT record of that transaction in the log.

2. **C.** Note that an *increment* action does not conflict with other *increment* actions as described in the class.

3. **A.** In (i), Q' is equivalent to Q, and Q' is always more efficient, because we do not need to performance the join operation as in Q. In (ii), Q' is equivalent to Q. However, it is not always more efficient (e.g., when $R \bowtie S$ is very unselective, while $\sigma_{B=5}(R)$ is very selective). In (iii), Q' is not even equivalent to Q. So, we cannot replace Q with Q'.

4. **B.** Since S_1 is conflict-serializable, its precedence graph $P(S_1)$ is acyclic, and the topological order of the nodes in $P(S_1)$ implies a serial schedule S_s to which S_1 is conflict-equivalent. Since $P(S_1) = P(S_2)$, $P(S_2)$ must have the same topological order as $P(S_1)$. That means S_2 is conflict-equivalent to S_s, and therefore it is conflict-equivalent to S_1 and conflict-serializable. However, S_2 may not be a serial schedule itself. So, only (ii) and (iii) are true.

5. **A.** According to the timestamp-based protocol, the action $r_2(A)$ will be delayed because T_1 just wrote A and T_1 has not committed. $r_2(B)$ is also delayed. After $w_1(B)$ completes, T_1 commits. Then, T_2 can go ahead to perform $r_2(A)$, $r_2(B)$, and $w_2(A)$. So, T_2 commits too.

6. **D.** (i) is possible because of the following scenario: During phase 1 of the two-phase commit, Z sent a "ready" vote to the coordinator, but X sent "don't commit". During phase 2, the coordinator sent "abort" to all component sites. Y received the message and aborted, while Z had not received the message before the disconnection. (ii) is possible because of the following scenario: During phase 1 of the two-phase commit, Z was down before it can log or send the "don't commit" message to the coordinator. After a timeout, the coordinator assumed that Z did not want to commit, and sent "abort" to all sites. Y received the message and aborted. (iii) is impossible.

7. **C.**

8. **B.**

Part III. (40 points) Write your final answer for each question.

1. Atomicity; Consistency; Isolation; Durability.

"Atomicity" means that *either all or none* of the actions in a transaction should be executed.

2. The schedule is view-serializable. The view-equivalent serial schedule is $r_1(A)\ w_1(A)\ w_2(A)\ r_3(A)\ w_3(A)$

3. 211 blocks (with 200 leaf nodes, 10 inner nodes, and on root node).

4. Use eight memory buffers to hold the root and seven inner nodes. Use two buffer as temporary input and output buffer. Perform an index lookup to find the tuple with the given key value. The expected # of I/Os to read the index nodes and the record block is $1 \times 0.7 + 2 \times 0.3 + 1 = 2.3$ I/Os.

5. 1539 blocks.

6. 0.5 ms.

7. T_1 committed; T_3 aborted; T_2 committed.

8. $min(\lceil 2^{k-3}/r \rceil - 1, 4)$. Since we use k-bit hash key with no duplicates, each bucket has at most $\lceil 2^{k-3}/r \rceil$ blocks. Because the threshold is $T = 0.8r$, there are at most $6 \times 0.8r$ records in the hash table. So, a bucket has at most $\lceil 4.8 \rceil = 5$ blocks (all records in one bucket). Therefore, the max number of overflow buckets is $min(\lceil 2^{k-3}/r \rceil - 1, 4)$.

Part IV. (60 points) Answer the following questions and *briefly* show your reasoning steps when necessary. Clearness and conciseness will be considered when grading this part.

1. (a) There are three possible logs that satisfy the 2PL. The filled in log records are as follows:

LG_1:	LG_2:	LG_3:
...	...	...
$\langle T_2,\ \mathbf{A},\ \mathbf{1},\ 3\rangle$	$\langle T_2,\ \mathbf{B},\ \mathbf{0},\ 3\rangle$	$\langle T_2,\ \mathbf{B},\ \mathbf{0},\ 3\rangle$
...	...	...
$\langle \mathbf{T_1},\ B,\ \mathbf{0},\ 4\rangle$	$\langle \mathbf{T_1},\ B,\ \mathbf{3},\ 4\rangle$	$\langle \mathbf{T_2},\ B,\ \mathbf{3},\ 4\rangle$
...	...	...
$\langle T_3,\ A,\ \mathbf{3},\ 8\rangle$	$\langle T_3,\ A,\ \mathbf{1},\ 8\rangle$	$\langle T_3,\ A,\ \mathbf{1},\ 8\rangle$

(b) Before recovery, the possible values of A and B on disk are:

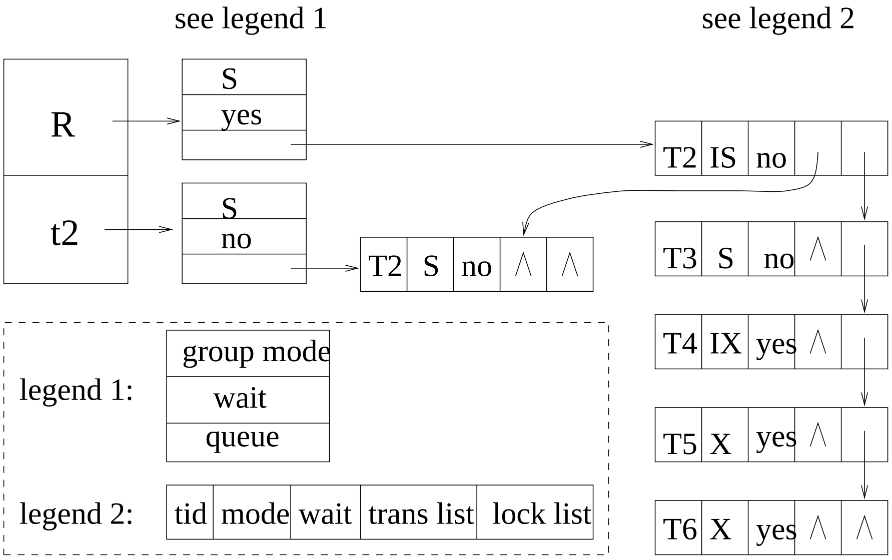
	A	B
LG_1	3/8	0/4/6
LG_2	1/8	3/4/6
LG_3	1/8	3/4/6

(c) Only LG_1 is recoverable. LG_2 and LG_3 are not recoverable, because T_1 read the value of B written by uncommitted T_2, and committed. None of the schedules avoids cascading rollback because either T_1 or T_3 read dirty data.

2. (a)

transaction	**lock on R**
T_1	X (or SIX)
T_2	IS
T_3	S
T_4	IX
T_5	X (or SIX)
T_6	X

(b)



(c)

time	actions
0	l_1
10τ	u_1, l_2, l_3
20τ	u_2, u_3, l_4
30τ	u_4, l_5
40τ	u_5, l_6
50τ	u_6

3. We can extend the hash-join algorithm as follows: First, hash the two relations as in the hash-join algorithm. Assuming that R is the build relation, for each buckets R_i in R, store the entire bucket R_i in memory. Add a *matched* bit to each tuple entry in the bucket, and set it to 0. Read blocks in the corresponding bucket S_i in S into memory one-by-one. For each S tuple t_S, if there is a matching tuple t_R in R, output the joined tuple, and set the *matched* bit to 1. Otherwise, output t_S padded with NULL for the R attributes. After reading all tuples in S_i, scan the tuples of R_i in memory. If the *matched* bit of a tuple is 0, output that tuple padded with NULL for the S attributes.

4. (a) In plan 1, # of bytes (S tuples) transfered from Y to X is $100000 \times 200 = 20$ MB. In plan 2, # of bytes (R tuples) transfered from X to Y is $1000 \times 300 = 0.3$ MB, and # of bytes (join result) transfered from Y to X is $25000 \times 400 = 10$ MB. Plan 2 is better.

(b) Use the semi-join scheme introduced in the class.

First, transfer $U = \pi_C(R)$ from X to Y. Then, join U with S, and send the matched tuples $W = S \bowtie U$ from Y to X. Finally, join R and W, and output the results.

# of bytes transfered from X to Y is $200 \times 100 = 0.02$ MB. # of bytes transfered from Y to X is $200/4000 \times 200 = 1$ MB.