CS 4320/5320 Homework 5

Spring 2016

Due Monday, 5/9/2016 at 11:59 pm

- Update 05/02/16: The hint for Question 3e has been modified.

This assignment is out of 100 points and counts for 10% of your overall grade.

1 Serializability (32 points)

Consider the schedules shown below. $R_i(A)$ and $W_i(A)$ denote the action of transaction i reading and writing object A respectively. Similarly, Commit_i and Abort_i denote the action of transaction i committing and aborting respectively.

1. $R_2(X)$, $W_3(X)$, Commit_3, $R_1(X)$, $W_2(X)$, Commit_2, $W_1(X)$, Commit_1
2. $R_2(X)$, Commit_2, $R_1(X)$, Commit_1
3. $R_2(X)$, $W_1(X)$, $R_2(X)$, Commit_1, Commit_2
4. $W_2(X)$, $R_1(X)$, Commit_1, $W_2(X)$, Commit_2
5. $W_2(X)$, $W_3(X)$, $R_1(X)$, $W_2(X)$, Commit_2, Commit_1, Commit_3
6. $W_3(X)$, $R_2(X)$, Commit_2, Commit_3, $R_1(X)$, Commit_1
7. $W_3(Y)$, $R_2(X)$, $W_2(Y)$, Commit_2, $W_3(Y)$, $R_1(Y)$, Commit_3, Commit_1
8. $R_1(X)$, $W_3(X)$, $W_2(X)$, Commit_2, $R_1(X)$, Commit_3, Commit_1
9. $R_2(X)$, $W_3(X)$, Commit_2, $R_1(X)$, Commit_3, Commit_1
10. $W_3(Y)$, $W_1(Y)$, Commit_3, $R_2(X)$, Commit_1, Commit_2
11. $R_1(X)$, $R_1(X)$, $R_2(X)$, $R_1(X)$, $W_3(X)$, Commit_1, Commit_3, $R_2(X)$, Commit_2
12. $W_2(X)$, $W_1(X)$, Commit_1, $W_2(X)$, Commit_2

(a) Complete the following table by placing a checkmark in row i column p if schedule i satisfies property p. We have inserted a zeroth row corresponding to a fictitious schedule not shown here. The row says that schedule 0 is view-serializable, not conflict-serializable, recoverable, avoids cascading aborts, and not strict. *(Hint: there is one schedule corresponding to each entry in the Venn diagram in Figure 17.9 on page 576 of your textbook.)*

	view-serializable	conflict-serializable	recoverable	avoids cascading aborts	strict
0	✓		✓	✓	
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					

(b) Show the conflict-graph for *every* schedule.

(c) Give a serialization for *every* view-serializable schedule. It is sufficient to list the order of the transactions in the serialization; you do not have to write out the actions. For example, the schedule in Figure 17.1 on page 551 of the textbook can be serialized T_1, T_2, T_3.

2 Concurrency Control (28 points)

2.1 Pessimistic Concurrency Control

Consider the following sequences of actions. Unlike the schedules in Question 1, the order of the actions in these sequences represent the order in which each action is requested, not the order in which they are executed. The actions may not be executed in the order they are requested. Some actions are executed immediately while others are blocked and executed after other requests have been processed.

1. $R_1(A), R_1(A), R_2(B), W_2(B), R_1(C), R_2(C), R_3(C), W_1(C), \text{Commit}_1, R_3(B), \text{Commit}_3, \text{Commit}_2$
2. $W_1(A), W_2(B), W_3(C), R_1(B), \text{Commit}_1, R_2(C), \text{Commit}_2, R_3(A), \text{Commit}_3$

Let $S_i(Y)?$, $S_i(Y)\checkmark$, and ~~$S_i(Y)$~~ denote the action of transaction i requesting a shared lock on object Y, being granted a shared lock on object Y, and releasing a shared lock on object Y respectively. Similarly, let $X_i(Y)?$, $X_i(Y)\checkmark$, and ~~$X_i(Y)$~~ denote the same actions for an exclusive lock.

(a) Assume the DBMS is using *non-conservative strict 2PL* with deadlock detection. For schedule 1, show the order in which all lock requests, lock acquisitions, lock releases, reads, writes, commits, and aborts happen. If a deadlock occurs, abort the transaction that arrived latest. **Also draw the waits-for graph for the deadlocked system.** Assume shared locks are used for reads, exclusive locks are used for writes, and that a shared lock on object Y can be upgraded to an exclusive lock on object Y so long as there are no other shared locks on object Y.

To get you started, we've provided the first eight actions.

T_1	T_2	T_3
$S_1(A)$?		
$S_1(A)$✓		
$R_1(A)$		
$R_1(A)$		
	$S_2(B)$?	
	$S_2(B)$✓	
	$R_2(B)$	
	$X_2(B)$?	

(b) Same as (a) for schedule 2.

(c) Now assume the DBMS is using *conservative strict 2PL*. For schedule 1, show the order in which all lock requests, lock acquisitions, lock releases, reads, writes, and commits happen. Assume that before a transaction performs its first action, it requests a shared lock for every object it reads and an exclusive lock for every object it writes. Also, assume that locks are requested in the order the objects are read or written by the transaction.

Again, we've provided the first eight actions to get you started.

T_1	T_2	T_3
$S_1(A)$?		
$S_1(A)$✓		
$X_1(C)$?		
$X_1(C)$✓		
$R_1(A)$		
$R_1(A)$		
	$X_2(B)$?	
	$X_2(B)$✓	

(d) Same as (c) for schedule 2.

2.2 Optimistic Concurrency Control

Consider a database using optimistic concurrency control. Whenever a transaction attempts to commit, the database atomically performs its verify phase and write phase, if the verify phase succeeds. Or, it atomically performs its verify phase and aborts the transaction, if the verify phase fails. Which of the following schedules can be executed by the database without aborting any transaction?

1. $W_1(A), \text{Commit}_1, W_2(A), \text{Commit}_2$
2. $R_1(A), R_2(A), W_1(B), \text{Commit}_1, \text{Commit}_2$
3. $R_1(A), R_2(A), W_1(A), \text{Commit}_1, \text{Commit}_2$
4. $R_1(A), W_2(A), W_1(A), \text{Commit}_1, \text{Commit}_2$
5. $R_1(A), W_2(A), W_1(C), R_2(B), R_3(C), W_3(C), \text{Commit}_1, \text{Commit}_2, \text{Commit}_3$
6. $R_1(A), W_2(A), W_1(B), \text{Commit}_1, R_2(A), R_3(B), \text{Commit}_2, W_3(B), \text{Commit}_3$

3 ARIES (26 points)

Consider a database using ARIES that performs the following operations before crashing.

$$W_1(A), \overline{W_2(B)}, \overline{W_3(C)}, \text{Checkpoint}, W_2(D), W_1(A), \text{Commit}_1, W_3(C), \overline{W_2(D)}, W_2(B), W_3(A)$$

Assume each object X is on its own page with id P_X. For example, object A is on page P_A, object B is on page P_B, and so on. For every overlined action, the page being written by the action is also force written to disk. For example, immediately after $\overline{W_2(B)}$, P_B is forced to disk. Besides those immediately following an overlined action, no force-writes have been performed. The Checkpoint action represents a begin_checkpoint action followed immediately by an end_checkpoint action.

(a) Draw the dirty page table, transaction table, log, and pageLSN on disk (if one exists) of pages P_A, P_B, P_C, and P_D immediately before the checkpoint.

We have drawn the ARIES state after the first *two* actions for you in Figure 1. **Don't forget to process the third action!**

pageID	recLSN
P_A	0

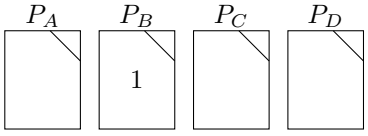
(a) Dirty Page Table

transID	status	lastLSN
T_1	running	0
T_2	running	1

(b) Transaction Table

LSN	transID	type	pageID	undoNextLSN	prevLSN
0	T_1	update	P_A		$\perp$
1	T_2	update	P_B		$\perp$

(c) Log



(d) pageLSN of each page on disk.

Figure 1: ARIES state after first two actions.

(b) Draw the dirty page table, transaction table, log, and pageLSN on disk (if one exists) of pages P_A, P_B, P_C, and P_D immediately before the crash.

(c) Which log records were written to stable storage before the crash?

(d) Draw the dirty page table, transaction table, log, and pageLSN on disk (if one exists) of pages P_A, P_B, P_C, and P_D after the Analysis phase of ARIES.

(e) For each update log record processed in the Redo phase of ARIES, state whether the update is redone or not. If the action is not redone, note which of three conditions on the bottom of page 590 of your textbook causes the action to not be redone. *(Hint: each condition will be cited at least once.)*

(f) Draw the dirty page table, transaction table, log, and pageLSN on disk (if one exists) of pages P_A, P_B, P_C, and P_D after the Undo phase of ARIES.

4 Distributed Protocols (14 points)

(a) Consider the 2PC protocol presented on page 759 of your textbook. We'll refer to this version of 2PC as naive 2PC. An execution of naive 2PC in which all subordinates vote no is shown in Figure 2. There is a single coordinator labeled c and three subordinates labeled s_1, s_2, and s_3. Time progresses rightward through the figure; the leftmost subfigure depicts the first half of the voting phase, and the rightmost figure depicts the second half of the termination phase. Nodes are shaded red when they perform a forced write.

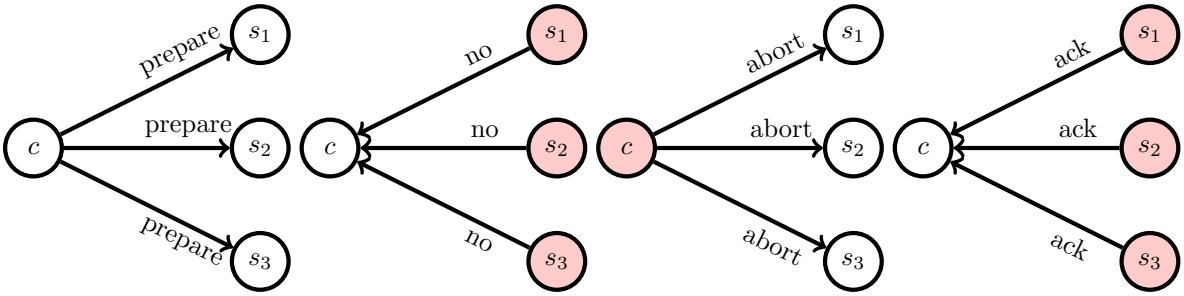


Figure 2: Naive 2PC execution in which all subordinates vote no. In the first half of the voting phase, the coordinator does not perform a forced write and sends a prepare message to the subordinates. In the second half of the voting phase, all three subordinates force-write an abort log record and send a no vote. In the first half of the termination phase, the coordinator force-writes an abort log record and sends an abort message to subordinates. In the second half of the termination phase, all three subordinates force-write an abort log record and send an ack to the coordinator.

Similarly, the execution of 2PC with presumed abort—as presented on pages 761 and 762 of your textbook—in which all subordinates vote no is shown in Figure 3.

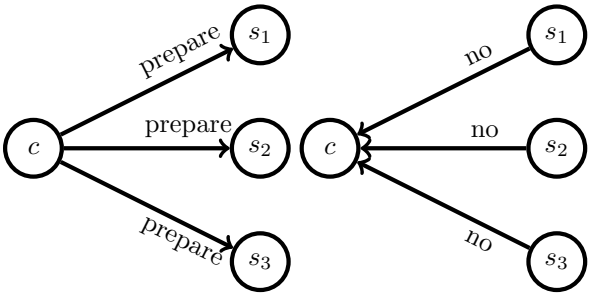


Figure 3: 2PC with presumed abort execution in which all subordinates vote no.

(i) Assuming no site or communication failures, draw the execution of naive 2PC and 2PC with presumed abort in which s_1 and s_2 vote yes but s_3 votes no.

(ii) How many messages are sent in the naive 2PC execution?

(iii) How many messages are sent in the 2PC with presumed abort execution?

(iv) How many force-writes are performed in the naive 2PC execution?

(v) How many force-writes are performed in the 2PC with presumed abort execution?

(vi) Assuming no site or communication failures, what must s_1, s_2, and s_3 vote in order for both naive 2PC and 2PC with presumed abort to send the same number of messages and to perform the same number of force-writes?

(b) Imagine a variant of 2PC with presumed abort in which the coordinator *does not* force-write a commit log record after receiving a yes vote from every subordinate. Describe an execution of this protocol which leads to an erroneous situation in which some subordinates commit and some subordinates abort. In particular, make sure to describe how many subordinates there are, what the subordinates vote, what messages are sent between the coordinator and subordinates, if and when nodes crash, if and when nodes restart, etc.

(c) Consider the execution of Paxos given in Figure 4 with two proposers (A and B) and three acceptors (X, Y, and Z). Proposer A successfully has its value $v = a$ accepted by a majority of acceptors. Proposer B sends a prepare request with proposal $n = 2, v = b$. Acceptor Y and Z both respond with the highest valued proposal they have accepted: $n = 1, v = a$. Proposer B then sends an accept request with some value v and id $n = 2$. Finally, Acceptor Y and Acceptor Z both accept some value v'.

(i) What value v does Proposer B send in its accept request?

(ii) What value v' do Acceptors Y and Z send in their accept responses?

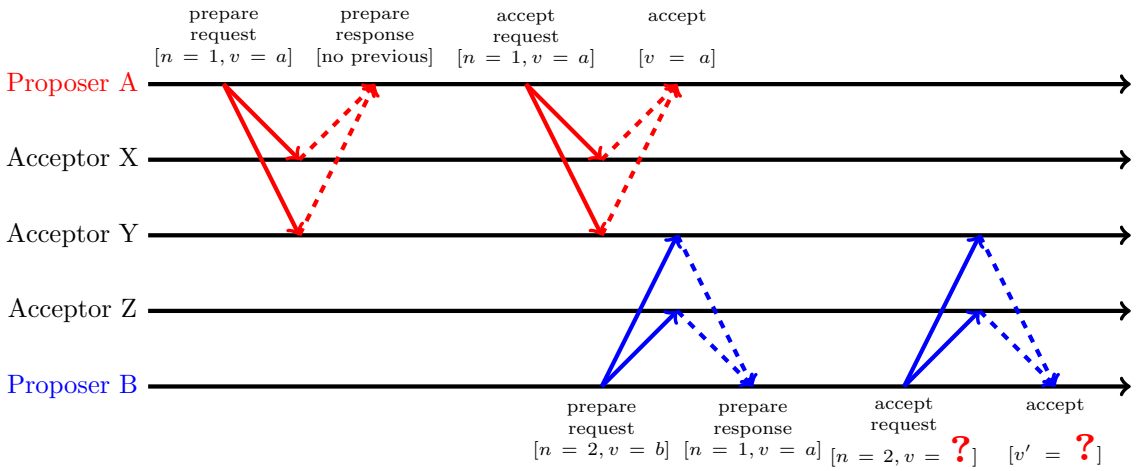


Figure 4: Example Paxos execution.

(d) Imagine a variant of Paxos in which a value is considered chosen when it is accepted by *at least one* acceptor rather than a *majority* of acceptors. Draw an execution of this variant of Paxos that results in two different values being chosen.

5 Submission Instructions

Please submit a single PDF file to CMS with your answers to every question. As usual, your solutions must be typed; scans of hand-written solutions will not be accepted. If applicable, include an acknowledgments section as required under the academic integrity policy.