CS542 Spring 2009 Midterm *and solutions*, 11 March, 2009
Prof. Chris Clifton

Time will be tight. If you spend more than the recommended time on any question, **go on to the next one**. If you can't answer it in the recommended time, you are either going in to too much detail or the question is material you don't know well. You can skip one or two parts and still demonstrate what I believe to be an A-level understanding of the material.

Notation: Throughout, lower case letters at the beginning of the alphabet $(a, b, c, ...)$ refer to sites. Numbers $(1, 2, 3, ...)$ identify the transaction. Upper case letters at the end of the alphabet (S, T, U, V, X, Y, Z) are object identifiers.

Note: It is okay to abbreviate in your answers, as long as the abbreviations are unambiguous and reasonably obvious.

*Solutions represent **a** solution, not necessarily the only or best solution.* **Scoring is also a general idea.** My expectation is that students with an A level knowledge of the material will score 24 or better, B at least 18, C at least 12.

1 Distributed Concurrency (18 minutes, 10 points)

You are given the following three transactions (expressed as a series of reads and writes on objects):

T_1: $r_1[U_a]$ $r_1[X_b]$ $w_1[V_a]$ $commit_1$

T_2: $w_2[X_b]$ $r_2[U_a]$ $r_2[Y_c]$ $commit_2$

T_3: $w_3[V_a]$ $r_3[U_a]$ $w_3[U_a]$ $commit_3$

Assume objects U and V are at site a, X is at site b, and Y is at site c. Assume that T_1 is run at site a, T_2 at site b, and T_3 at site c. Each site can execute one action per time step (which includes sending that action to another site); if a site receives an action, it is able to execute it and return the results in the next time step (and will do so before executing the next action for a local transaction, unless the concurrency control mechanism causes it to be delayed, in which case it can execute a different action.) A transaction must await the results of one action before performing the next.

1.1 Two-Phase Locking

Show the global order of actions (i.e., fill in the following table) assuming (non-strict, non-conservative) two-phase locking. (You don't need to show the send operations - I've just put the example in to help clarify what is happening.)

step	Site a	Site b	Site c
1	$r_1[U_a]$	$w_2[X_b]$	*send* $w_3[V_a]$ *to* 1
2	$w_3[V_a]$		
3	$r_2[U_a]$	T_1 *waits for lock on* X_b	
4	$r_3[U_a]$		
5	T_1 *waits for lock on* V_a		$r_2[Y_c]$
6	T_3 *waits for lock on* U_a	$commit_2$	
7	*Discover deadlock locally, abort* T_1	$r_1[X_b]$	
8	$w_3[U_a]$		
9	$r_1[U_a]$		$commit_3$
10			
11		$r_1[X_b]$	
12	$w_1[V_a]$		
13	$commit_1$		
14			

(You may not need all the space provided.)

Scoring: 1 point for each wait for a lock, 2 points for detecting the deadlock, 1 point for correcting for deadlock, 1 point each for getting each transaction complete up to the deadlock, one point for completing the schedule.

2 Failure/Recovery (13 minutes, 11 points)

Assume a site *a* has failed during execution of three-phase commit. On restarting and initiating recovery, the log at that site shows the following:

Tid	*Object*	*Old*	*New*	*Question number*
	Checkpoint			
T_1	Start			
T_1	X_a	3	4	1
T_2	Start			
T_2	X_a	4	5	2
T_3	Start			
T_3	Y_a	7	8	3
T_3	Start 3PC as participant			
T_1	Y_a	8	9	
T_3	pre-commit as participant			
T_3	commit			4
T_1	Start 3PC as coordinator			
T_2	Start 3PC as participant			
T_1	pre-commit as coordinator			5

2.1 Recovery process (9 minutes, 10 points)

For each of the log items with a *Question number* (*1..5*), explain what is done with that item during the recovery process. Note that it will probably be easier if you start with *5* and work your way back to *1.* Assume undo/redo logging (you can use redo/undo if you prefer, but please tell me you are doing so.)

1. *If T_1 aborted, write* 3 *to X_a in the undo phase. If committed, write r to X_a in the redo phase.*

   Scoring: 1 for noting either abort or commit, one for showing correct action for one of them.

2. *Since the commit process for T_2 has been started, I've presumably sent a yes vote. Therefore T_2 may have decided to commit - but also may have decided to abort. Check with other participants to determine. If commit, then write* 5 *to X_a in the redo phase. If abort, write* 4 *to X_a in the undo phase.*

   Scoring: 1 for noting either abort or commit, one for showing correct action for one of them.

3. *In the redo phase, write* 8 *to Y_a.*

   Scoring: 1 for noting committed, one for showing correct action.

4. *This means that T_3 will be redone.*

   Scoring: This is an easy two points.

5. *When the coordinator fails in the pre-commit phase, it needs to check with other participants to determine if the the transaction committed or not.*

   Scoring: One for noting that commit/abort needs to be determined, one for noting that other participants need to be asked.

2.2 Logging (3 minutes, 1 point)

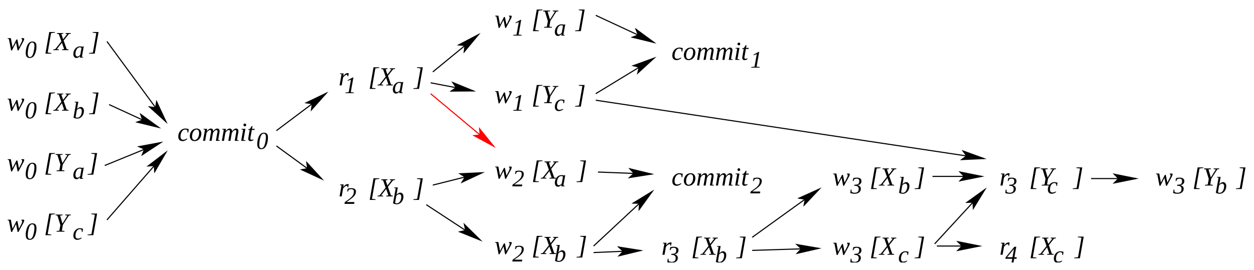
Is any of the information logged above unnecessary (e.g., no matter when a failure occurs, we could recover just as easily without that information as we can with it)? Explain.

Given that we are not using a fuzzy checkpointing scheme, we know that no transaction could have started before the checkpoint. To make sure we undo/redo all transactions, we must go back to the checkpoint. Therefore the start transaction could be determined implicitly from the first write by a transaction, and doesn't need to be logged.

Scoring: One point for correctly identifying and explaining why something can be ignored.

3 Replicated Data (8 minutes, 9 points)

You are given the following serialization graph for a replicated data history:



Note that transactions 3 and 4 are not complete, and may have further actions before committing or aborting. *The red line was erroneously omitted from the exam; scoring takes this into account.*

3.1 1-Copy Serializability (2 minutes, 3 points)

Is the above graph 1-copy serializable? Explain why / why not.

Yes. We can draw the serialization graph and see that $T_0 \rightarrow T_1 \rightarrow T_2 \rightarrow T_3 \rightarrow T_4$ is a valid serialization (assuming T_3 and T_4 complete without conflict.)

Scoring: One for showing knowledge of how to determine serializability, one for demonstrating the process, one for giving a schedule/proof of serializability at least through T_2.

3.2 Consistency Protocols (2 minutes, 3 points)

Does the above graph show a protocol that enforces write-all, write-all-available, or neither? Explain how you made this determination. Assume that there may be some site failures/recoveries in the history.

Note that T_3 writes X_b but not X_a, so this must be either write-all-available and site a has failed, or neither.

Scoring: One for noting not write-all, two for noting site a must have failed if write-all-available (or "neither".)

3.3 Site recovery (3 minutes, 3 points)

Assume that site a fails sometime after the last operation that performs an action at site a, and recovers immediately after the last operation in the displayed history is performed. What should the site do when it recovers?

Since site a holds a copy of replicated items X and Y, it needs to start a transaction $r[X_b]$ $r[Y_c]$ $w[X_a]$ $w[Y_a]$ $w[X_b]$ $w[Y_b]$ $w[X_c]$ $w[Y_c]$. Note that the writes at sites b and c seem redundant, but may be necessary to enforce serializability (e.g., some other process may be writing at the same time, not knowing a is up.)

Scoring: One for bringing X_a and Y_a up to date, one for noting this must be a transaction, one for ensuring correct ordering with respect to actions on other copies.