CS W186 - Spring 2020
Guerrilla Section 5
Recovery and Distributed Transactions

Sunday, April 26, 2020

1 Conceptual Recovery

1. Consider a scenario where we update the recLSN in the dirty page table to reflect each update to a page, regardless of when the page was brought into the buffer pool. What bugs might you see after recovery? Select all that apply. Explain your reasoning.

   (a) Some writes of committed transactions would be lost.

   (b) Some writes of aborted transactions would be visible in the database.

   (c) The system tries to commit or abort a transaction that is not in the transaction table.

   Answer: a, b. a is correct because during the REDO phase of recovery, some UPDATE log records that reflect writes that never made it to disk will be skipped. Similarly, b is correct, because some CLR's that reflect UNDO's that never made it to disk will be skipped. c is incorrect because even if REDO begins at a later LSN, the system does not add any new transactions to the transaction table during REDO.

2. Suppose that you are forced to flush pages in the DPT to disk upon making a checkpoint. Which of the following cases are now guaranteed? There is one correct answer. Explain your reasoning.

   (a) We can skip one of the three phases (analysis/redo/undo) completely

   (b) We must start analysis from the beginning of the log

   (c) Redo will start at the checkpoint.

   (d) Redo must start from the beginning of the log

   (e) Undo can start at the checkpoint

   (f) Undo must run until the beginning of the log

   Answer: c. In general, we redo everything from the earliest recLSN in the DPT to get back unflushed changes from before crash. Since we can guarantee that all changes up to a checkpoint have been flushed, all unflushed changes from before the crash happened after the checkpoint. Therefore, we can redo can start from the checkpoint.

3. If the buffer pool is large enough that uncommitted data are never forced to disk, is UNDO still necessary? How about REDO? Explain your reasoning.

Answer: UNDO isn't necessary in terms of undoing operations on disk. Having a buffer pool large enough to hold all uncommitted data means we don't have to STEAL (allow an uncommitted transaction to overwrite the most recent committed value of an object on disk). Since all the updates will be sitting in the buffer pool at the time of crash, no changes will be made to disk, so no operations need to be undone. REDO is still necessary. REDO is needed to get back unflushed changes from before the crash. If everything is held in the buffer, this must be redone.

4. If updates are always forced to disk when a transaction commits, is UNDO still necessary? Will ARIES perform any REDOs? Explain your reasoning.

   Answer: Only UNDO is necessary. UNDO is necessary because we still have to finish aborting transactions that were in progress and weren't committing. The updates that these transactions have gotten through so far must be rolled back. As for REDOs, ARIES might perform some redoes because there may be transactions still in progress at the time of a crash, but these will be undone in the UNDO phase.

2 Recovery Practice

For this question, you will want to have the details of the ARIES protocol handy, so we suggest you have the slides or some notes to look at while doing this question.

The year is 2029. Power outages in Berkeley are so common now that PG&E does not even send out warnings anymore - instead, they just pull the plug whenever they want.

Our database has just restarted from one such power outage. You look at the logs on disk, and this is what you see:

LSN	Record		
10	T1	update	P1
20	T2	update	P2
30	T1	update	P2
40	T1	update	P3
50		begin-checkpoint	
60	T1	update	P4
70		end-checkpoint	
80	T1	commit	
90	T2	update	P1

You load up the checkpoint and see:

Transaction Table

Txn ID	Last LSN	Txn status
T1	40	running
T2	20	running

Dirty Page Table

Page	recLSN
P1	10
P2	30

1. What is the latest LSN that this checkpoint is guaranteed to be up-to-date to?

   Answer: LSN 50 - the `begin-checkpoint` record.

2. What do the transaction table and dirty page table look like at the end of analysis, and what log records do we write during analysis?

LSN	Record
100	T1 end
110	T2 abort

Transaction Table

Txn ID	Last LSN	Txn status
T2	110	aborting

Dirty Page Table

Page	recLSN
P1	10
P2	30
P4	60

3. The next phase of ARIES is redo. What LSN do we start the redo from?

   Answer: LSN 10 - the oldest recLSN in the dirty page table.

4. From that record, we will redo the effects of all the following records, except we will not redo certain records. What are the LSNs of the records we do NOT redo?

   Answer:
   LSN 20 is not redone; the recLSN of P2 is already higher than 20, so the effects of LSN 20 are already on disk.
   LSN 40 is not redone; page P3 is not in the dirty page table, and thus also already on disk.
   LSNs 50, 70, and 80 are not update operations, so we don't do anything for them.

5. The last phase of ARIES is undo. What do we do for this phase? Answer this question by writing out the log records that will be recorded for each step.

   Stop after you write your first CLR record (make sure your CLR record specifies the nextLSN!).

LSN	Record
120	T2 CLR nextLSN: 20

Click! The lights go out, and you realize PG&E has pulled the power yet again... during ARIES recovery no less!

Five minutes later, the power comes back online. You inspect the log, and are glad to see that **all the log records you wrote have made it to disk**.

6. You load up the checkpoint. What does the transaction table and dirty page table look like?

   Answer: Same as the checkpoint from before! We never made another checkpoint.

7. You run the analysis phase. What do the transaction table and dirty page look like at the end of analysis?

Transaction Table

Txn ID	Last LSN	Txn status
T2	120	aborting

Dirty Page Table

Page	recLSN
P1	10
P2	30
P4	60

8. You run the redo phase. In order, what are the LSNs that we redo?

   Answer: 10, 30, 60, 90, 120. Importantly, note that we redid T1 even though it committed, and that we ARE redoing CLRs.

9. Now we run the undo phase. What do we do? (Answer again with the log records that you have to add.)

LSN	Record
130	T2 CLR nextLSN: null
140	T2 end

3 Conceptual Distributed Transactions

1. For each of the following statements, mark whether it's true or false and explain your reasoning.

   (a) Resolving a deadlock at a local node will always resolve distributed deadlock.

   (b) Suppose we make the changes needed so only a majority of participants need to vote yes for a transaction to commit. Participants can write an ABORT record in phase 1 of 2PC.

   (c) Presumed abort as presented in lecture no longer works in the scenario described in 1b.

   (d) If **any** machine sees COMMIT record in its log upon recovery, the transaction will commit. Additionally, describe how to distinguish whether a machine with a COMMIT record is a coordinator or participant.

   Answer:
   a is false. Distributed deadlock occurs whenever the union of waits-for graphs across different nodes. Resolving one deadlock may not solve deadlock in the union of the nodes' waits-for graphs.
   b is false. Participants can't unilaterally abort anymore in this new scenario. The aborting participant can't determine the result of the vote by itself, so the participant must receive the result of the vote from the leader.
   c is true. If a participant crashes after voting no in phase 1, with presumed abort, no ABORT record will be flushed to disk. Coming back online, the participant would ordinarily presume abort, but in the scenario described, this may not be the case if the majority still votes to commit.
   d is true. Whether participant or coordinator, a node seeing a COMMIT record in its log on recovery must take the actions. If the commit record holds the participantIDs, then we know the machine must be a coordinator. Additionally, if this coordinator doesn't have an END record and only a COMMIT, it must send the result of the vote to all the participants. If there is a PREPARE record alongside the COMMIT record, then we know the node must be a participant.

2. For the following scenarios, describe what will happen without presumed abort, then describe what will happen with presumed abort.

   (a) Participant recovers and sees just a phase 1 ABORT record

   (b) Participant recovers and sees a PREPARE record

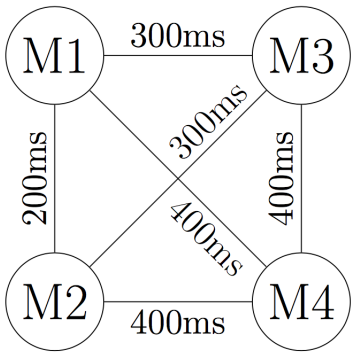
   (c) Participant recovers and sees a PREPARE and phase 2 ABORT record

   (d) Coordinator recovers and sees an ABORT record

   Answer:
   a. Without presumed abort: Send "no" to the coordinator. With presumed abort: Nothing! The coordinator will presume abort.
   b. The case is the same with or without presumed abort. The participant will ask the coordinator the result of the vote and continue the process.
   c. Without presumed abort, we'll send ACK back to the coordinator. With presumed abort, participants don't need to do anything after an abort record that was the result of the vote.
   d. Without presumed abort: Inform the participants about the outcome of the vote. With presumed abort: Nothing! (Participants who don't know the decision will just ask later.)

4 Two Phase Commit Practice (Fall 2017 Final Question 3)

Our database runs on 4 machines and uses Two-Phase Commit. Machine 1 is the Coordinator, while Machines 2, 3, and 4 are Participants. Suppose our machines are connected such that the time it takes to send a message from Machine i to Machine j is 100·max(i,j) milliseconds (see graph below). Assume these communication latencies are symmetric: it takes the same amount of time to send from i to j as it takes to send from j to i. For example, sending a message between Machine 2 and Machine 4 takes 400 milliseconds in either direction. Assume that the transaction will commit (i.e. all subordinates vote yes), and that everything is instantaneous except for the time spent sending messages between two machines.



1. What is the first message Machine 1 sends?

   (a) VOTE YES
   (b) PREPARE
   (c) COMMIT
   (d) None of the above

   Answer: (b) PREPARE. The coordinator (i.e. Machine 1) begins the first round of two-phase commit by sending a PREPARE message to all of the subordinates.

2. What is the second message Machine 1 sends?

   (a) VOTE YES
   (b) PREPARE
   (c) COMMIT
   (d) None of the above

   Answer: (c) COMMIT. The coordinator (i.e. Machine 1) begins the second phase of two-phase commit by sending a COMMIT message to all of the subordinates. Note that the problem states that all subordinates vote yes in the first round which is why the coordinator sends a COMMIT message (instead of an ABORT message) to start the second round.

3. How much time passes from when Machine 1 sends its first message to when Machine 1 sends its second message?

   Answer: This is just the maximum time for a round-trip between Machine 1 and Machines 2–4, since Machine 1 sends its second message (COMMIT) once its first message (PREPARE) reaches each of the other machines and each of the machines respond back with a VOTE YES message. The time for a round-trip between Machine 1 and Machine i is $2 \cdot 100 \cdot i$, and the largest of these is with Machine 4: $2 \cdot 100 \cdot 4 = 800$ms.

4. What is the first message Machine 2 sends?

   (a) VOTE YES
   (b) PREPARE
   (c) COMMIT
   (d) None of the above

   Answer: (a) VOTE YES. A subordinate (e.g. Machine 2) responds to a PREPARE message from the coordinator with a VOTE YES or VOTE NO. The problem states that all subordinates vote yes, so Machine 2 sends a VOTE YES message.

5. What is the second message Machine 2 sends?

   (a) VOTE YES
   (b) PREPARE
   (c) COMMIT
   (d) None of the above

   Answer: (d) None of the above. The second message Machine 2 (a participant) sends is an ACK.

6. How much time passes from when Machine 2 sends its first message to when Machine 2 sends its second message?

   Answer: Let's have Machine 1 send the PREPARE message at time 0. This PREPARE message will reach Machine 2 at time 200ms, so Machine 2 sends its first message (VOTE YES) at time 200ms. Machine 1 sends the COMMIT message at time 800ms (when it hears back from Machine 4), and this message takes 200ms to reach Machine 2. Machine 2 therefore receives the COMMIT message at time 1000ms, and sends its second message (ACK). The time passed is therefore 1000ms - 200ms = 800ms.

7. True or False. A transaction is considered committed even if over half of the participants do not acknowledge the commit.

   Answer: True. A transaction must commit once the COMMIT message is sent out, even if all the participants promptly crash repeatedly and do not respond with ACKs as a result.

Now suppose that our implementation of 2-Phase Commit has an off-by-one bug where the Coordinator receives, but does not use, Machine 4's vote. That is, Machine 4's vote does not affect whether or not the transaction commits or aborts. Answer True or False for the following questions:

8. A transaction that should normally commit may be aborted instead.
9. A transaction that should normally abort may be committed instead.
10. A transaction that should normally commit may be committed properly.
11. A transaction that should normally abort may be aborted properly.

Answers:
8. False. If the transaction would normally commit, then Machines 2 and 3 must have been functional and voted yes, so the transaction would still have to commit with this bug.
9. True. If Machine 4 votes no and other participants vote yes, then the transaction should be aborted but commits anyways.
10. True. If all machines vote yes, then the transaction commits even if we ignore Machine 4's vote.
11. True. If Machine 2 or 3 votes no, then the transaction aborts in both cases.