
   a. [10 points] During recovery, Aries reconstructs the transaction table, attempting to bring it to the same state it was in at the time of the crash. However, it may “miss” some transactions. Describe how this could happen, and explain why transactions missed in this way do not cause a problem for recovery.

   Aries will “miss” transactions that started before the crash but created no log records that were written to disk. By the assumption of the write-ahead log protocol, such a transaction cannot have committed and cannot have made any changes to pages that made it to disk, so it is safe to ignore such transactions.

   b. [10 points] Instead of writing CLRs during the undo phase of recovery, Joe 764 proposes two alternatives. One is to do no logging whatsoever during UNDO, the other is to write “normal” log records (treating the undo operation as a normal data modification.) Explain why both of his alternatives are bad.

   Writing no log records can lead to errors with logical logging, since if there is a subsequent crash, the system has no idea if an UNDO has been performed or not. Writing “normal” log records avoids this problem, but has the issue that with repeated crashes the log will grow without bound.

   c. [5 points] Does the pageLSN of a page modified during the REDO phase need to be updated to account for this modification? Why or why not?

   Yes. If not, there is no record of whether or not the REDO has been performed, which can lead to errors if there are subsequent crashes.

2. [25 points] Parallel DBMS.

   a. [10 points] What is an argument for using a shared-nothing architecture for a parallel DBMS instead of using a shared-disk architecture?

   There were many possibilities. One is that with shared-disk, the network bandwidth must equal the combined bandwidth of all the disks, or there may be network bottlenecks. Another argument is that with shared-disk, much more care must be taken to handle consistency of data that can be cached in multiple places and accessed by any node in the system at any time.
b. [10 points] Give an example of when a repartition operation is required, and describe the three conceptual stages of the repartitioning.

Suppose we are performing the join of R and S, and neither R nor S is partitioned on the join attribute. Then we need to repartition both R and S before the join. The three phases are split (each node splits its portion of the table into fragments), shuffle (redistribute the fragments), and merge (combine the shuffled fragments at their destinations.)

c. [5 points] Define linear scaleup, say why it is desirable, and explain one factor that might keep a system from achieving linear scaleup in practice.

Linear scaleup means that if a problem can be solved on a single processor in $T$ seconds, a problem $n$ times larger can be solved in $T$ seconds on $n$ processors. Interference for shared resources can keep a system from achieving linear scaleup, as can the use of a non-linear running time algorithm.


a. [10 points] Two-phase commit is a blocking protocol in that it is possible that a participant in the commit might have to wait for another participant to recover before knowing what to do. Give an example of a case in which a participant must block in this way.

A good example of this is a case where the coordinator sends out a “prepare,” a subordinate votes “yes”, then the coordinator crashes. The subordinate must then wait for the coordinator to recover before it knows what to do.

b. [10 points] Explain why the “collecting” record is required in presumed-commit two-phase commit, but not in presumed-abort two-phase commit.

In presumed commit, the “no knowledge” answer by the master is “commit.” So if the master crashes after issuing a “prepare,” it needs to know which subordinates were participating in the transaction so that it can be sure they all know the transaction is aborted (if it misses one subordinate, and the subordinate asks, the master will answer “commit.”)

In presumed abort the “no knowledge” answer is “abort” so there is no need to ensure that all subordinates know the result of the transaction – they can learn it one by one by asking the coordinator.

c. [5 points] Jane 764 is confused about the “read-only” optimization for two-phase commit. Her thinking is “if a subordinate is read-only, then why can’t the coordinator just ignore it during the commit protocol, because it really doesn’t matter whether the read-only subordinate commits or not.” What is Jane missing?
The short answer is that the coordinator does not know ahead of time which subordinates were read-only. In fact, the subordinates themselves may not know before running if they are read only – they could have a conditional update where the condition is not satisfied so there is no update.

   a) [10 points] The System-R query optimizer only considers left-deep trees. Can you come up with an explanation of why they chose left-deep trees instead of right-deep trees?

   One explanation is that given the join algorithms System-R had at the time, left-deep trees can be pipelined but right-deep trees cannot.

   b) [10 points] Query optimizers make use of estimates of cardinalities when estimating the cost of plans. Suppose you were able to magically “reach in” and convert some of the cardinality estimates to their true, exact values. Would this necessarily result in better query plans? Why or why not?

   One answer is that query optimizers make decisions based on their estimates of the relative costs of two plans. It is possible that these estimates can be in error in a consistent way – that is, the estimate for P1 is too low, the estimate for P2 is too low, but the optimizer is right that P1 is faster than P2. If you reach in and “fix” the estimate for P1, the optimizer might incorrectly assume that P2 is faster than P1.

   c) [5 points] Why is it important to keep track of “interesting orders” instead of just calculating the single best one-table plan for each table in the query?

   This is really an attempt to mitigate the errors a greedy optimization algorithm can make. If one only remembers the cheapest plan at each level, one might miss plans in which a single table access is slower than the best option for that table, but the overall cost is good because the slower single table access results in tuples ordered in a way needed further up the tree. Keeping “interesting order” plans around tries to avoid this problem.