CSE 444: Lab 4
Aakash Srazali (aaksra) | Cherry Winarto (cwinarto)

1. **Describe your implementation including any design decisions you made. Make sure to emphasize anything that is difficult or unexpected.**

   Implementing the recovery algorithm that aligns with ARIES was somewhat difficult due to the limited data structures that we have. Instead of changing log methods in LogFile.java, we utilized compensating log record (CLR) objects to maintain update records of aborting transactions or uncommitted transactions during a crash.

   The CLR object contained the transaction id, start offset of the UPDATE_RECORD log record to be undone, and the page's before image modified by the UPDATE_RECORD. We also incorporated a CLR_RECORD log record to log rollback/undo operations, making sure there are no repetition of changes during recurrent crashes and restarts. This CLR_RECORD includes the record type, transaction id, start offset of the undone UPDATE_RECORD log record, and start offset of the CLR_RECORD.

   During rollback or undo phases, we scanned the log file forwards from the respective offset. Stack<CLR> was used to store information about the UPDATE_RECORDS relevant for aborting transactions or uncommitted transactions. After reading the log file, we sequentially popped the CLRs from the stack, undoing the latest log records first, and wrote the CLR's before-image to the table file on disk.

   The recovery phase combined a redo and undo phases. The redo phase started from the checkpoint (if any), building a set of "loser transactions IDs" from all active transactions identified by the CHECKPOINT_RECORD. Scanning through the log records, updates are redone and the set of loser transactions is modified. These loser transactions exclude committed transactions and are essentially uncommitted transactions requiring undoing, and were handled in the subsequent undo phase.

2. **Discuss and justify any changes you made outside of LogFile.java.**

   Changes made outside of LogFile.java included changes in BufferPool.java where we implemented a STEAL, NO FORCE policy for undo-redo logging. For instance, for STEAL policy, the implemented eviction policy flushes and evicts any page (whether the page is dirty or not). A log record, UPDATE_RECORD, is created on the log which contains a page's before-image and after-image. We do this to force the log to disk which guarantees that the log record is on disk before the page gets written to disk since active transactions are allowed to have their dirty pages flushed to disk. This way, SimpleDB can rollback or recover in the event that the transaction aborts or the database crashes before the current active transaction is completed entirely.

   For the NO FORCE policy, a change is made to the transactionComplete function to no longer force dirty pages during commit. After each transaction commits, the updates immediately get forced to disk. So transactionComplete now creates and forces UPDATE_RECORD for every dirty page to allow SimpleDB to rollback changes on the entire page in the event that the transaction aborts or a database crash before we get the chance to write the updates to disk. transactionComplete also updates the

before-image of every dirty page with the page's current contents in case of a rollback and recovery for the next transaction that accesses and modifies this page.

3. **Describe a unit test that could be added to improve the set of unit tests that we provided for this lab.**

   A unit test that could be added is one that checks for the ability of SimpleDB to recover correctly after crashing during a recovery.

4. **If you have any feedback for us on the assignment, you can add it to the writeup, send us an email, or make a note and paste your note in the course evaluation form at the end of the quarter.**

   N/A