In this lab, we introduced a STEAL and NO-FORCE, a departure from the NO-STEAL and FORCE policy implemented in Lab 3. We incorporated logging to support this policy and ensure database recoverability during a crash, closely aligning with the ARIES protocol. Our first step was to generate a log entry for every modification to the database, including updates, commits, aborts, begins, and checkpoints. Under the NO STEAL policy, memory pages can be flushed anytime. Therefore, the primary adjustment we made before committing was to ensure the log was forced to disk after entry creation as necessary. Consequently, we recorded a log entry associated with every update, whether an insertion or deletion. We also periodically forced certain log records to disk, especially during transaction commits, aborts, or checkpoints.

If a transaction fails, it's necessary to revert all changes to the state of the pages before initiating the transaction. This is achieved by reversing all modifications made by the transaction to the versions of the pages before the updates were made. As part of the design, we introduced additional Compensation Log Records (CLRs) at the end of the log (just before the ABORT log entry) to document the reversals we performed, including the state of the pages before and after each reversal. Once all CLRs for the transaction's updates are recorded, we appended an UNDO record, and then the log is flushed to disk, ensuring all changes are securely saved.

In a similar process, during recovery, we identify all transactions that had not been completed or terminated since the last checkpoint and recognize that the updates from these transactions must be reversed. CLR and Update Log records are handled identically during this stage, with all actions reapplying from the last checkpoint. Subsequently, updates from the loser transactions are reversed, and CLR records are generated. Once this process is complete, the database system can proceed with the next set of updates as it would have before the crash, except that some transactions have been aborted and must be restarted.

The most challenging aspect of the lab was understanding the LogFile code and interpreting the method to read the file while accounting for the various format types. Furthermore, debugging in this lab proved significantly more intricate than Lab 3.

Changes outside LogFile.java:

We had to modify several aspects of our implementation to ensure the STEAL and NO FORCE policy operated correctly. One significant alteration was to the `BufferPool.transactionComplete(commit = true)` method. Initially, this involved scanning through dirty pages in the BufferPool, identifying those marked as dirty, determining the transaction responsible, and then incorporating the before image statements for those pages. However, under the STEAL policy, pages can be flushed without warning, and their dirty status reset to false, rendering my previous method for tracking changes by transactions ineffective. To address this, we leveraged the lockManager to identify pages modified by the transaction, allowing us to determine which pages' before images needed updating accurately. Moreover, we introduced a new function in the LockManager class to release all locks for a specific transaction. Previously, removing a transaction from the lock manager required the pageId of the page to be cleared. To facilitate the eviction of any page and the removal of locks without needing a page ID, we implemented a method that accepts a transaction and removes locks from any associated pages. Additionally, we revised our eviction strategy to a random eviction policy, wherein a page is randomly selected from the buffer pool for eviction, regardless of its dirty status.

Unit Test Addition: This test would involve performing a series of insert operations. Some transactions will be committed to ensure their changes are supposed to be durable. Others will not be committed to simulating incomplete operations before the transaction completes. After restart, the test verifies that only the changes from committed transactions are visible and persisted, while changes from uncommitted transactions are not present.

Feedback: No feedback.