Writeup for lab5 **Yuwei Wu**

CS392 Database Management System ACM Class, Zhiyuan College, SJTU

Prof. **Feifei Li** Due Date: June 16, 2019

Submit Date: June 14, 2019

Design decisions about lab5

- I implement a NO STEAL/FORCE buffer management policy. That is, in the evictPage method, I only choose the pages that are not dirtied by any transactions as candidate pages to evict and then just randomly choose one to evict. And once a transaction is completed, I force all dirty pages caused by the transaction to disk in the transaction-Complete method.

- To manage the transactions and locks, I create a LockManager class to allocate read and write permissions to each transaction. If there is no exclusive lock on the requested page, a shared lock will be allocated to the transaction if it requests for a read permission on the page. If no other transactions are holding write or read permission on the requested page, an exclusive lock will be allocated to the transaction if it requests for a read-write permission on the page. *shareLocks* and *exclusiveLocks* are two concurrent maps that map the transaction id to the page id on which holds the permission. Not until request is satisfied, the transaction will be waiting in the while loop.

- In the deadlock detection part, I implement dfs searching algorithm to search through the dependees of a transaction id which are other transactions blocking the current transaction. If the dependent of one transaction appears directly or indirectly in the list of the transaction's dependee, a cycle is formed which means a deadlock is detected. A list for recording visited transaction ids and a stack for recording dependent transaction ids through the search are used.

- To avoid two or more threads acquire lock on same page at the same time, I use a pageId2Lock map and do synchronizing on the lock got by the pageId.

- For the transactionComplete method, it is important that the id2Page map maintained in the bufferpool is not safe to use when we need to flush dirty pages of the transaction during one commit or revert changes if transaction is aborted. If we iterate through the id2Page.keySet() and find all page with isDirty()==tid, it is likely that not all the page dirtied by tid is found. For example, in the deleteTuple method in the BTreeFile, it deletes the tuple in the found page and then do redistribution on leaf pages, which will dirty some other pages and are likely to throw TransactionAbortException. If a TransactionAbortException is thrown right after tuple is successfully deleted in the page, the dirtied pages will not be returned to bufferpool to allow mark dirty by tid and then iteration in the TransactionComplete method can never revert the deletion as the dirtied pages can not be found. A possible **solution** for finding pages that dirtied by given tid is to maintain a tid2PageId list in the LockManger (can not be maintained in the bufferpool for similar reasons mentioned above). The tid2PageId map is to store any tid that acquire a read-write permission on the page recorded by its pid. It is obvious only pages that are held by tid with read-write permission need to be flushed to disk or revert when a transaction is completed. And the record is done every time the read-write permission is requested and is right before the time that may throw TransactionAbortException. Also, the pid can only be held by one tid with exclusion lock, which means the map is quite thread-safe.

API changes

- No API changes.

Missing or incomplete elements

- No missing parts.

Time spent and difficulties

- I spent about half day writing codes on lab5 and two days debugging.
- I found iteration on id2Page is quite dangerous as many threads are attempting to do modifications on it at same time. So in transactionComplete method, I get page from disk not by the id2Page map. If iterating id2Page map to get page, the page may be removed by other threads in the map as I get from pid.
- And there are also problems occurring in the insertTuple method in HeapFile class. When iterating pages to find empty slot, these pages should be accessed with read-only permission and once the page is full, the tid should release locks and move to next page.

- And in the dfs searching algorithm for deadlock detection, if every time the algorithm is to search through all the tids, it is quite time-consuming and will exceed time limitations if many threads is processing. And it is faster to do dfs starting from the tid that requests for permission.