Sunday, June 16, 2019

Lab5 Report

Jiangbei Li

- Design Decision
    - LockManager
        - In the class, I create 4 maps to save the locks
            - Map1&2: PageId -> TransactionId (Read&Write)
            - Map3&4: TransactionId -> PageId (Read&Write)
        - I also create two maps to save the query that waits lock
            - Map5&6: PageId -> TransactionId (Read&Write)
        - I create another map to save a mutex for each page to keep synchronized: Map7: PageId -> mutex
        - I use concurrentHashMap instead of HashMap and HashSet.
        - Function: grantLock
            - Use synchronized for the mutex of the page.
            - If there exist ww, wr, rw, then add the query to wait map; else add lock to the tid and pid.
        - Function: releaseTidLock
            - Release all the locks that the TransactionId holds and the waiting query of the TransactionId
        - Function: releaseLock
            - Release the lock for the tid and pid.
        - Function: holdsWriteLock
            - Return whether the TransactionId hold write lock for the page.
        - Function: holdsLock
            - Return whether the TransactionId hold lock for the page.
        - Function: deadLockOccur
            - Detect the deadlock for the buffer
    - BufferPool

  - Add class member lockManager to keep lock.
  - Function: getPage
    - Call grantLock to request a lock for the query
    - If not grantLock, then call deadLockOccur to detect deadLock and keep calling grantLock until granted
  - Function: releasePage and holdsLock
    - Call the corresponding function in lockManager.
  - Function: transactionComplete
    - If the commit is True: For all pages that the tid has write lock on it, I flush these pages and mark them not dirty.
    - If the commit is not True: I call getBeforeImage and fetch the old page from Disk to buffer.
- Deadlocks
  - I create directed dependency graph for each transaction and page
  - For one transaction, if it has a lock, and another transaction is waiting for the lock (ww, wr, rw), then there is an edge from the first transaction to the second.
  - From the input transaction, I use BFS(Breadth first search) to traverse the graph: if there is a cycle for the start vector then there will be a deadLock.
- Difficulty and time
  - I spend four days on this lab
  - I use hashSet firstly but it is not concurrent, then I use concurrentHashMap and use the keyset to replace hashSet.
  - I forgot to write writePage in HeapFile. I comment on a necessary code in insertTuple in HeapPage. Then the inserted tuple will have no recordId
  - When running TransactionTest, I catch the TransactionAbortedException ahead of the TransactionTestClass. Then the query will not rerun.
  - The BTreeTest is very hard and I cannot find a good way to debug.
- Weakness
  - The running time is very long for BTreeTest and may fail with a very small possibility. I am trying to make it faster.