CSE-4331/5331-001

DBMS Project- 2

Transaction Manager Implementation

Prof. Abhishek Santra

3/19/2024

Team 15

Muna Bhattarai-1001746212

Abhishek Patel-1002033618

Jeet Sheth-1002175315

Brief Overview of the project

The overall program runs smoothly. Required code was added in zgt_tm.C and zgt_tx.C files. All the test cases given- C2Tsz.txt, deadlock.txt, NoC2T.txt, RR.txt, RRW.txt, and S2T.txt are tested and associated logfiles get created when the test is run. The major functions that were coded during the project is given-

Readtx/ Writetx

It invokes a function start_operation, locks the transaction manager and calls 'get_tx' function to retrieve information based on tid. If the transaction is found, it proceeds to check its status. If the transaction status is TR_END, TR_ABORT, or TR_WAIT, it invokes do_commit_abort_operation with the corresponding status and exits the thread after releasing the lock. If the transaction status is TR_ACTIVE, it sets a lock using tx->set_lock and exits the thread after releasing the lock. If none of the conditions match, it prints "Invalid Tx State". In writetx, Exclusive lock is initiated and in readtx Shared lock is initiated.

Process_read_write_operation

The function takes four parameters: tid (transaction ID), obno (object number), count, and mode. It starts the operation associated with the transaction and locks the transaction manager using start_operation and zgt_p(0). If the transaction is active, it sets a shared lock and exits the thread. If the mode is read, it sets shared locks ('S'); if the mode is write, it sets exclusive locks ('X'); and if the mode is neither 'r' nor 'w', it prints "Invalid Mode".

Aborttx

It starts the operation associated with the transaction and locks the transaction manager using start_operation and zgt_p(0). It writes a log record indicating that the transaction abort has been initiated, including transaction ID and type. It retrieves transaction information using the get_tx function based on the provided transaction ID (tid). If the transaction is not null, it proceeds to abort the transaction using do_commit_abort_operation with the TR_ABORT status and releases the lock on the transaction manager. If the transaction is valid, it writes a log record indicating that the transaction abort has been finished. If the transaction is null, it prints "Invalid Tx" and logs the attempt to abort an invalid transaction.

Committx

It extracts transaction information (tid and Txtype) from the arg parameter and writes a log record indicating the initiation of the transaction commit. Then, it starts the operation using start_operation and lock the transaction using zgt_p(0). If the transaction exists (tx != NULL), it commits the transaction using do_commit_abort_operation with the TR_END status, indicating a successful commit, releases the lock on the transaction manager (zgt_v(0)), finishes the operation using 'finish_operation', and writes a log record indicating that the transaction commit has been finished successfully. If the transaction does not exist (tx == NULL), it logs the transaction status and an error message indicating an attempt to commit a non-existent transaction.

Do_commit_abort_operation

It takes two parameters: t (transaction ID) and status (transaction status, either TR_ABORT or TR_END). If the status is TR_ABORT, it logs "AbortTx" and prints "Abort Tx" indicating the transaction has been aborted. If the status is TR_END, it logs "CommitTx" and prints "Commit Tx" indicating the transaction has been committed. If the status is neither TR_ABORT nor TR_END, it retrieves the transaction information using get_tx(t). If the transaction exists (tx != NULL), it updates the transaction status to the provided status (tx->status = status). It calls tx->free_locks() to release all locks associated with the transaction and logs to “FreedLocks”. It calls tx->remove_tx() to remove the transaction from the transaction list and logs to “RemovedTx”. If the transaction was waiting (tx_waiting != -1), it releases the semaphore associated with the transaction to wake up sleeping transactions. If the transaction does not exist (tx == NULL), it logs the attempt to commit/abort an invalid transaction and prints an error message.

Set_lock

It acquires a lock on the transaction manager by calling zgt_p(0) and searches for a node in the hash table (ZGT_Ht) based on the provided segment number (sgno1) and object number (obno1). It retrieves the transaction associated with the provided transaction ID (tid1) by calling the get_tx function. If the node is not found in the hash table (node == NULL), it adds the transaction to the hash table and performs a read or write operation on the object (perform_read_write_operation). Then, it releases the lock on the transaction manager and returns 0. If the node is found in the hash table, it checks if the transaction has already locked the object (temp != NULL). If the transaction has not locked the object yet, it checks various conditions to determine if it should wait or perform the operation immediately. If waiting is required (TR_WAIT status), it sets the appropriate parameters in the transaction list, sets a semaphore for the transaction, releases the lock on the transaction manager, performs the read or write operation on the object, and then re-acquires the lock on the transaction manager. If waiting is not required, it performs the read or write operation immediately. Finally, it returns 0 to indicate successful execution of the method.

Perform_read_write_operation

This function increases the read/write transaction by 7 and decreases the read transaction by 4 depending on the lockmode status.

Encountered Difficulty

1. It was difficult for us to correctly identify which transaction is holding lock on which object. We used print statements to check everything.
2. Program running without error but not providing expected output. This was more time consuming when we did not get the expected output on the log file.

File Description

2 test cases were created- test3 and test4.

Division of Labor

The project was done in a team. In total 17 hours were spent by each student together while working on the project. The team met at a designated location and worked together rather than splitting tasks among themselves.

Code Review and Understanding	2 hours
Working on functions	6 hours
Fixing bugs	6 hours
Testing	2 hours
Document	1 hours

Logical errors:

1. Segmentation Fault
   Program runs without error but when the same code is run continuously sometimes segmentation fault occurs. This is because when we run same program continuously, C shift the memory stack to higher and higher memory location and at some point goes beyond memory limits trying to access a memory location that does not exists resulting in segmentation fault.

2. Deadlock does not hang for rare cases
   In an extremely extremely rare condition, when the transaction in deadlock.txt file executes both transaction in a serial manner, all the operation on both the transaction get executed with no deadlock. However, deadlock is suppose to happen in this test.

3. Input Error
   The program is extremely depended on the fixed schema of the input file. If the format is not correct, the terminal gives the ‘Input Error’ message. We need to make sure that all the transactions need to be executed for proper result.