UNIVERSITY OF TEXAS AT ARLINGTON

Project Report on the Implementation of Transaction manager

CSE 5331- Fall 2016

DBMS Models and implementation (Section 001)
Instructor: Sharma Chakravarthy
Ta: Abhishek Santra

Name: Mir Basheer Ali ID: 1001400462
Name: Deeksha Bhat ID: 1001174031

Overall Status:

In this project, we were asked to complete the implementation of the strict two-phase locking(S2PL) protocol with shared locks for read and exclusive locks for write. We were asked to implement following functions in zgt_tx.C

- readtx
- writetx
- aborttx
- committx
- do_commit_abort
- set_lock

As well as following functions in zgt_tm.C

- TxWrite
- CommitTx
- AbortTx

We used the omega server as well as local machine to implement the given project. The skeleton code that was given to us needed to be uploaded to the server and change the Makefile according to our path and run the make for program to compile.

While implementing Read and Write functions to perform read and write operation we used semaphores on the transaction to make other transaction wait on the transaction that held the lock until it either aborts or commits. If the lock is not present on an object that is demanded by the new transaction, then the new transaction would acquire lock and proceed further. But if the lock is held by the same transaction then it just will proceed further. While querying or modifying hash table we used semaphores on transaction manager object to lock the hash table i.e., zgt_p(0), zgt_v(0).

When the abort or commit, operation is performed we free all the locks held by the transaction and release the semaphore so that the transactions that are waiting on aborted or committed transaction would acquire the required object and proceed further. In case of abort however transaction object is not deleted from the list so that we can perform recovery operation in the later part.

In addition to this we have made some changes to remove_tx method to remove a transaction from list as it was not working properly.

All the functions are implemented and tested several times. Which seem to work correctly in almost all the cases.

Where You Encountered Difficulty:

- The most challenging task in the project was to figure out semaphore logic. For example: When the lock was released as a transaction would commit or abort if more than one transaction is waiting on that transaction then the program would hang or behave abnormally. We had to figure out the logic for making the waiting transaction carry on waiting on the transaction that has newly acquired lock.
- Next challenge was to understand when to lock and when to unlock on transaction manager i.e, when to carry on single threaded operation versus when to carry on the multithreaded operation.
- There were several other challenges like finding out if the hash table was modified or no before context switching of thread due to which we used lock on transaction manager before making any modification to hashtable.
- Another issue we found hard to debug was the program would hang once or twice while running it several times due to change in input sequence because of multithreaded programming.

FILE DESCRIPTIONS:

The implementation was an extension to the project skeleton provided. We have just modified remove_tx functionality for it to run properly.

DIVISION OF LABOUR:

We worked together in the implementation of the project. We started with going through the semaphore and multithreading concepts of c++ initially. We used to discuss algorithms and how could that be implemented. We spent around 8 hrs each week for about 2 weeks to complete this project. We came up with logic quickly however faced several issues while testing the program. It would work perfectly for the file with 2 transactions and would hang with more than two transactions. Also would work perfectly sometimes and behave abnormally several other times. However we worked and tried fixing most of the errors.

LOGICAL ERRORS AND HANDLING:

Error Description: Sometimes transaction link list would not be updated. For example: read operation on an object would take place before the transaction object is added into the link list.
Solution: We put a condition that until the transaction object was created don't proceed further with any other operation.

Error Description: Whenever the lock was released from a transaction all the waiting transaction would compete to acquire lock and the program would hang.
Solution: Make the transaction waiting for an object wait on the transaction that has newly acquired lock instead of waiting on the transaction that had the lock previously.

Error Description: Whenever we were accessing the hash table due to multi thread concept the multiple transaction was reading or writing on the hashtable, for example: multiple transaction were adding a lock on same object because while executing find() operation no transaction was holding the lock.
Solution: We locked the hashtable each time we access or modify it.

Error Description: When we remove a transaction without clearing the semaphore held by the transaction then the program was getting hanged.
Solution: We cleared the semaphore before removing the transaction from the transaction list.