Ishita Gangal (ig1043) and Divya Juneja (dj1322)
Advanced Database design document
Final Project

Replicated Concurrency Control and Recovery (RepCRec) system is implemented in this project. Two phase locking for reading and writing of data and also multiversion read for read-only operations are applied. There are 1-20 variables amongst which odd variables are non-replicated and even ones are replicated on 10 sites given.

A cycle is maintained which indicated the variables allocated which transactions and which transactions are waiting for which variables. Deadlock detection algorithm is applied using the DFS approach and the youngest transaction is killed when cycle is detected in a graph. Commands of begin, end, write, read, readonly, fail, recover are taken care of, which trigger appropriate actions that need to be taken. Recovery of a site is done when recover() command is encountered. The odd variable commands encountered when the site was down are implemented after the recovery as they are non-replicated variables. The even indexed variables are not allowed to be read until they are written by some transaction.

<u>Implementation details:</u>

Commands are read and interpreted. If they begin with W, R, RO, end, begin, then Transaction manager is called which manages all the transaction related task. Otherwise, if the command starts with fail or recover, site manager is called which calls the functions of appropriate site instances in order to change the state of site, to update the variables on it, to store the blocked commands which should be executed later on the site recovery, to print the variable values available on each site when dump() is encountered.

Multiple transactions can acquire read locks until write lock is already acquired on it. We maintain locktable on each site which describes which variable has been acquired by which transaction and the type of lock.

A transaction is aborted and marked aborted when it access a variable and later if the site which holds that variable fails. We release the locks of aborted transactions then and there itself, enabling other transactions to acquire those locks. We check if there is any deadlock in the graph before the execution of any read or write command. ReadOnly commands enable the enforcement of multiversion concurrency control. If a transaction starts after another transaction ends, it reads the updated value by another transactions, otherwise reads the old and original value if it started before other transaction writing over variables ends, i.e. reads the last commited value before it began.

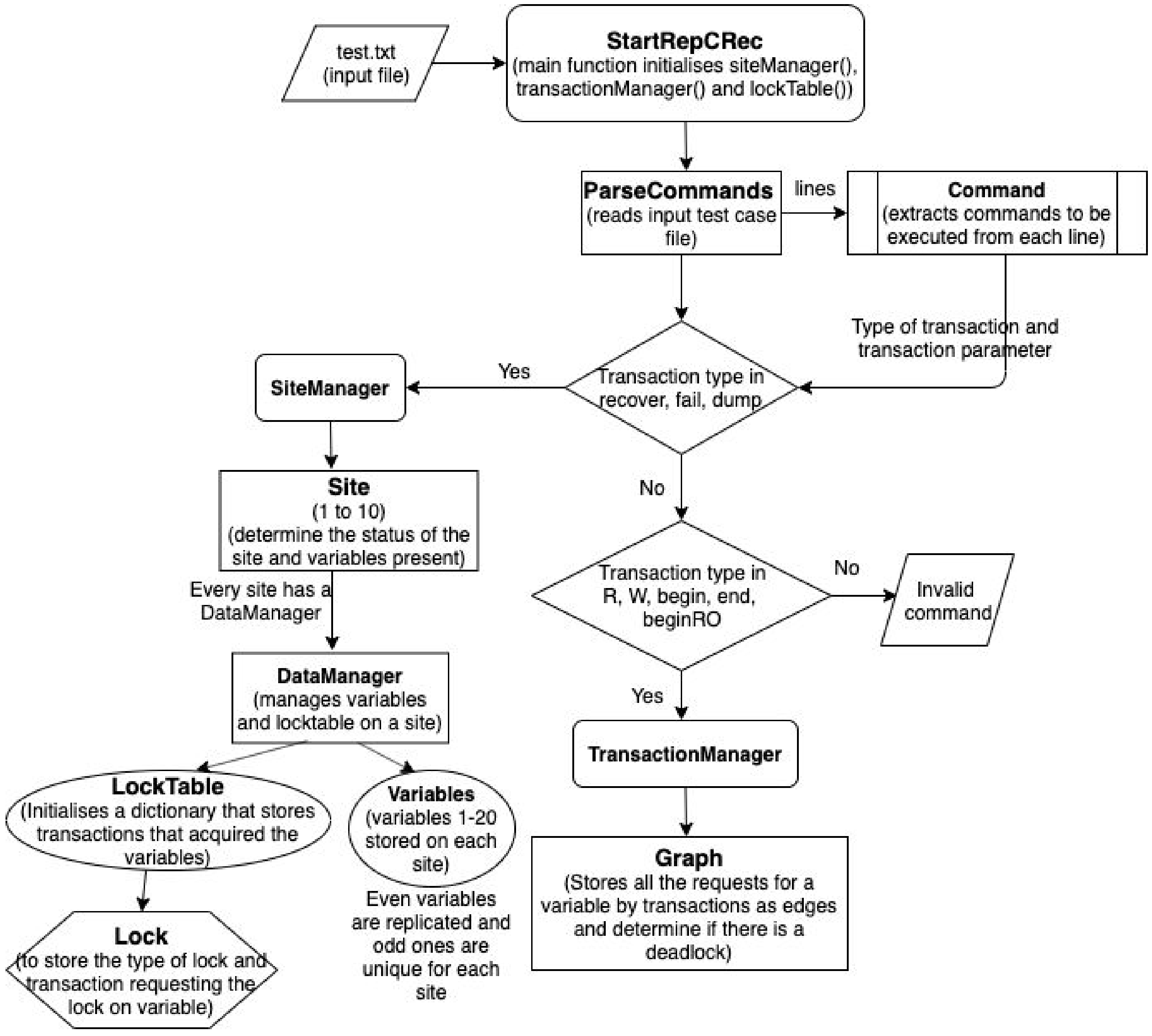
If a transaction has acquired write lock and later requests for read lock, it is granted. If a transaction already has a read lock and requests for an upgrade, we check for further steps.

Deadlock detection is done using the approach of DFS (Depth First Search) to detect the cycle, after which we also check the self loop to make sure of the edge cases when a transaction requests for upgrade of locks (like test case 20 and 11 given in handouts). If the transaction has read lock on a variable and requests upgrade request of write lock over it, we check if any other transaction has already requested a write lock. In that case, a deadlock is detected and youngest transaction is killed.

When end command is encountered, values of variables are written over the sites that are UP. When a site fails, its lock table is deleted from that site and appropriate transactions are aborted. When site recovers, blocked transactions of odd non-replicated variables are executed and it is flagged as recovering, not allowing replicated variables to be read unless they are written to.

Flowchart:

Class structure for this code has been attached in Class_Structure.pdf. Flow diagram of the code is as follows:



test.txt (input file)
StartRepCRec (main function initialises siteManager(), transactionManager() and lockTable())
ParseCommands (reads input test case file)
lines
Command (extracts commands to be executed from each line)
Type of transaction and transaction parameter
Transaction type in recover, fail, dump
Yes
SiteManager
Site (1 to 10) (determine the status of the site and variables present)
Every site has a DataManager
DataManager (manages variables and locktable on a site)
LockTable (Initialises a dictionary that stores transactions that acquired the variables)
Lock (to store the type of lock and transaction requesting the lock on variable)
Variables (variables 1-20 stored on each site)
Even variables are replicated and odd ones are unique for each site
No
Transaction type in R, W, begin, end, beginRO
No
Invalid command
Yes
TransactionManager
Graph (Stores all the requests for a variable by transactions as edges and determine if there is a deadlock)

Language used:

Coding has been done in Python and test cases have been attached including the 21 test cases given on website and few extra edge cases.

Everything is zipped using Reprozip.