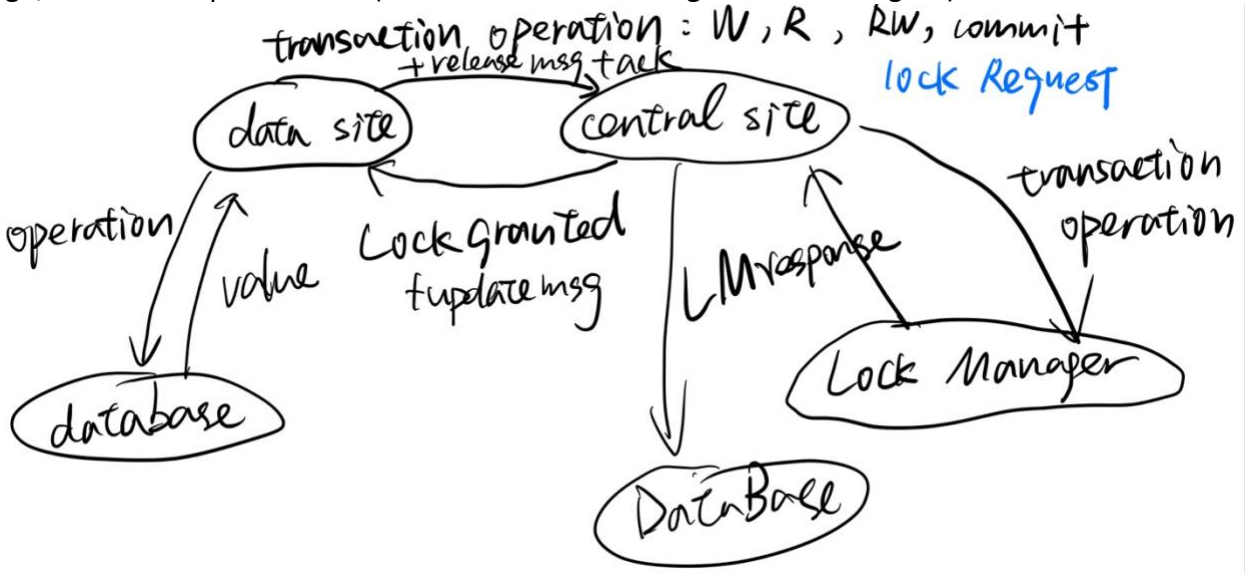
Project Report

1. Statement

   My project is about Centralized Two Phase Locking (2PL). The basic two-phase locking. Is about using 2 locks and the incompatibility in different situations to execute a bunch of transactions in serializable order, which avoid conflict between transactions. The centralized part is about utilization of two-phase locking in a distributed database. As the same as the requirement for this project, I use 4 sites with fully replicated database. One site is responsible for the arrangement of concurrent transactions and handle 2 phase locking, called central site. For other 3 sites that send lock request to central site is data site. The aim of the project is to figure out how 2 phase locking working and the communication between central site and data sites.

2. Design, methods or procedures. (Overview of whole design is in below figure)



1) Function for each part
   a. Database: In my database, there is one table that contains 2 columns: item(string) and value(integer). In detail, item contains a-z 26 items and the values for them is from 0-25.

      For test purpose, every time rebooting the whole system, databases are reset to original version as above by deleting the original table and create a new one and insert all data items into it.

      The service provide for database is read (read item and return value of the item), write (update value of data item with provided value), and insert (insert item and value into table).
   b. Data site: it is the site that is responsible for handling transaction and print out what happen in database. It sends operations to central site and wait for response. The data site sends operations to database.
   c. Central site: it is responsible for handle lock request from the data sites and send to lock manager and send message to data site. In all, central site is just a transfer station of messages for data site and lock manager.

d. Lock manager: it is the real part to handle 2-phase locking. It contains the lock table and wait-for graph for detect and resolve deadlock.

2) Communication:
There are 2 ways of communication between different part. For communication between sites and database, I use SQLite jdbc drive to modify or read data from the database. For communication between data sites and central site, between central site and lock manager, I use RMI. For lock manger and central site, at central site, I can call function in the lock manager because I know when to call it. But central site doesn't know when lock manager is ready to grant lock, so I use RMI function to do that when lock manager is ready. However, because both central site and data sites doesn't know when each other will reply, I just use RMI to just send the message but no requirement for reply for each operation. To implement the real-time communication, I register service on both side for data sites and central site so that they can send message to each other.

3) Operation:
Operation has data item, type and transaction ID and site ID, and value(for write and read_write).
There are 4 different types of operation: read (s_lock), write (x_lock), read_wirte(x_lock) and commit.
Read_write means this operation have read and write operation at the same time. In my design, I treat it as write. In the lock manager, this operation will have X_lock, same as write operation. In the data site, when it receive granted lock message, it will read from database first, and use this read value for next write operation.
Commit: At the data site, when send an operation, central site will send back a lockmsg. Only for commit operation, we need to check this return message. If return message is null, it needs to resend commit operation until it return original commit operation. Lock manager has a special function for commit operation. For each transaction, the first commit's meaning is this transaction's operations is load into lock table. The second and afterwards commit for this transaction means we need to unlock related operations in lock table.

4) Lock table details in lock manager:
lock table: I maintain 2 maps for lock table. In all, for each data item, there are 2 list: acquire and wait. Acquire list show operations that acquire this data item. Wait list shows operations that wait for this data item.
When a lock request message arrives, it adds this operation to the end of the linked list for the data item, if the linked list is present. Otherwise, it creates a new linked list, containing only the operation for the request.
It always adds an operation to acquire list on a data item that this acquire list doesn't have any operation. But if there is an operation that acquire this data item, the lock manager adds this operation to acquire list only if it is compatible with the lock types that are currently in acquire list, and all earlier requests have been released and removed from acquire list. Otherwise, this operation needs to be added to end of wait list of that data item.

When it receives release message, it releases (deletes) operation in the acquire list. Here, we need to find out if we need to move operations from wait list to acquire list. When acquire list is empty, we need to move one write or several read (or one read) from wait list to acquire list. Then send moved operations to data site as granted lock message. The movement of operation will lead to update in the relation of transaction waiting.

Commit: When it receives commit more than transaction number, we can start to process commit, which is unlock the operations because the first commit's meaning is this transaction's operations is load into lock table. Also, if this transaction is waiting for another transaction, we cannot start to process commit. These two cases can return null back to data site.

Transaction table: this table holds all operations for this transaction.

Processing commit: because we check if this transaction is waiting for another transaction. So we make sure we can unlock all the operations in this transaction without worrying about if its operation is in one wait list. We will send every granted lock message to data site.

5) Deadlock detection and solution in lock manager

   I maintain an edge table shows the wait-for graph. For example, edge[i] [j] = 1, it means that transaction i is waiting for transaction j. Every time there is a modification to any wait list, we need to update the wait-for graph. If there is a movement of operations from wait list to acquire list, moved operation's transaction needs stop waiting for original transaction acquiring this item. When adding an operation to wait list, if wait list is empty, the wait relation if from first wait transaction to transaction acquiring this item. But when the wait list is not empty, the wait relation is from this transaction to last transaction in wait list.

   Detection: For every modification to any wait list, we need to use bfs to search if there is a cycle in the wait-for graph. During traversal of the graph, I will mark transaction in 3 types. 0 means it is not visiting yes. 1 means during visiting. 2 means has visited. If at any point in the loop we try to access a transaction with state "1", we can indicate that there is a cycle in the wait-for graph;

   Solution: we can just abort the transaction that trigger this deadlock by deleting all the operation related to this transaction from acquire and wait lists. Also, it will remove the wait relation for this transaction so that other transaction can function correctly.

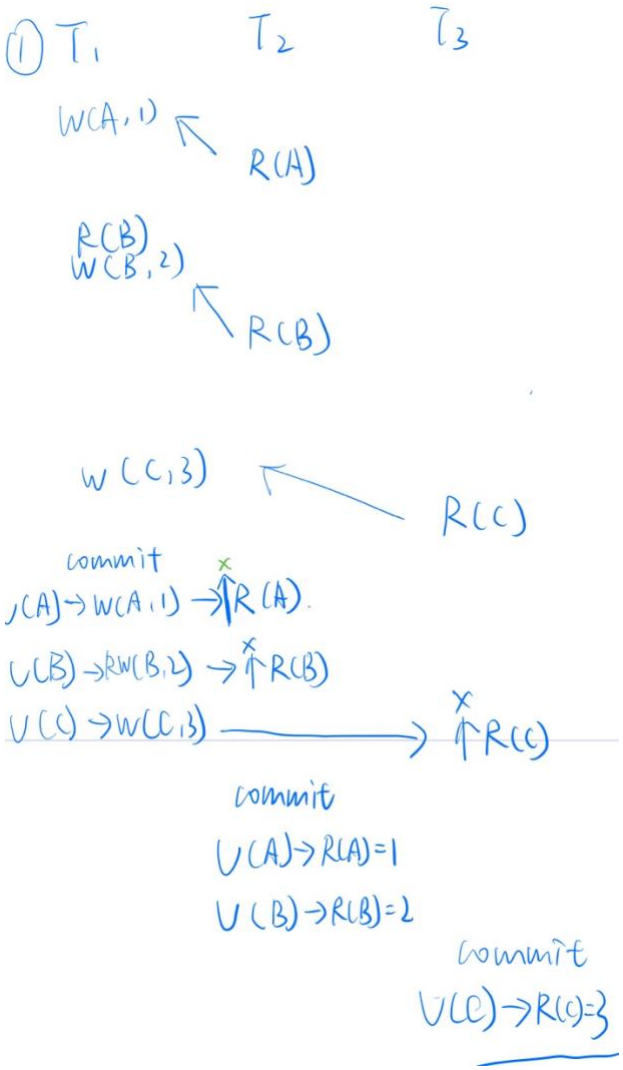
6) Procedure:
   1. Data sites and central sites established communication with each other. For each site it will connect to its database. For lock manager, it will connect to the central site.
   2. Data site read from txt file to put operation of a transaction into record. Then data site will send each operation to central site.
   3. Central site forward operations to lock manager to form a lock table and build wait relation.
   4. When lock manager receive commit, it will start to unlock the operation and send lock manager response to central site.
   5. Central site will send granted message to related data site.

6. Data site receive granted message, it will process it by call function in database (read or write). Then it will send release message to central site.
7. Central site receive release operation. If this operation is a write type, central site will update its database. It also sends update message to data sites other than this date site.
8. And central site will send more granted message to data site for operations that changed waiting status because of the deleted operation in the acquire list.
9. Central site forward this release message to lock manager. Lock manager will delete operation from lock table and delete some wait relation.

3. Test:

1)Test for serializability

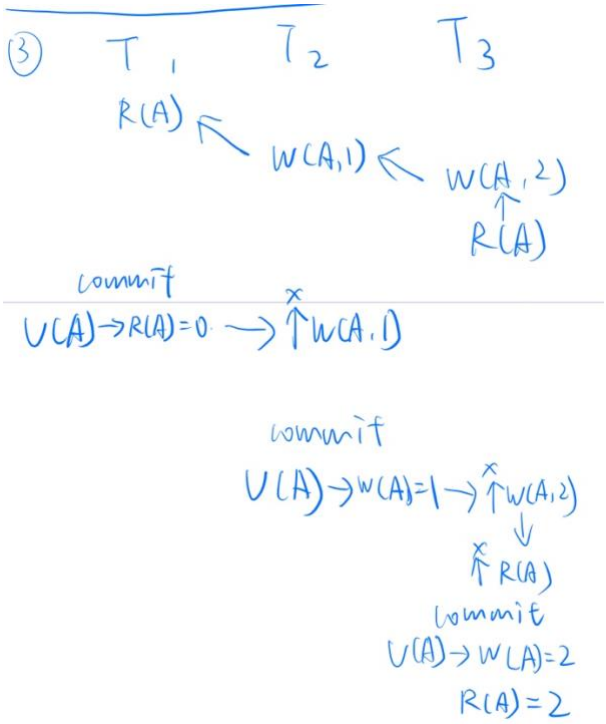
Test 1:



for each site, it holds one transaction. T1 for site1. For top to down, it is the timeline. The array shows that what operation is waiting for another operation. So T1 will be execute first because t1 is not waiting for any transaction. Unlock(a) move R(a) in T2

into acquire status. Unlock(b) move R(b) in T2 into acquire status. Unlock(c) move R(c) in T3 into acquire status. Then T2 can execute, unlock(a) and unlock(b). Then is the T3.

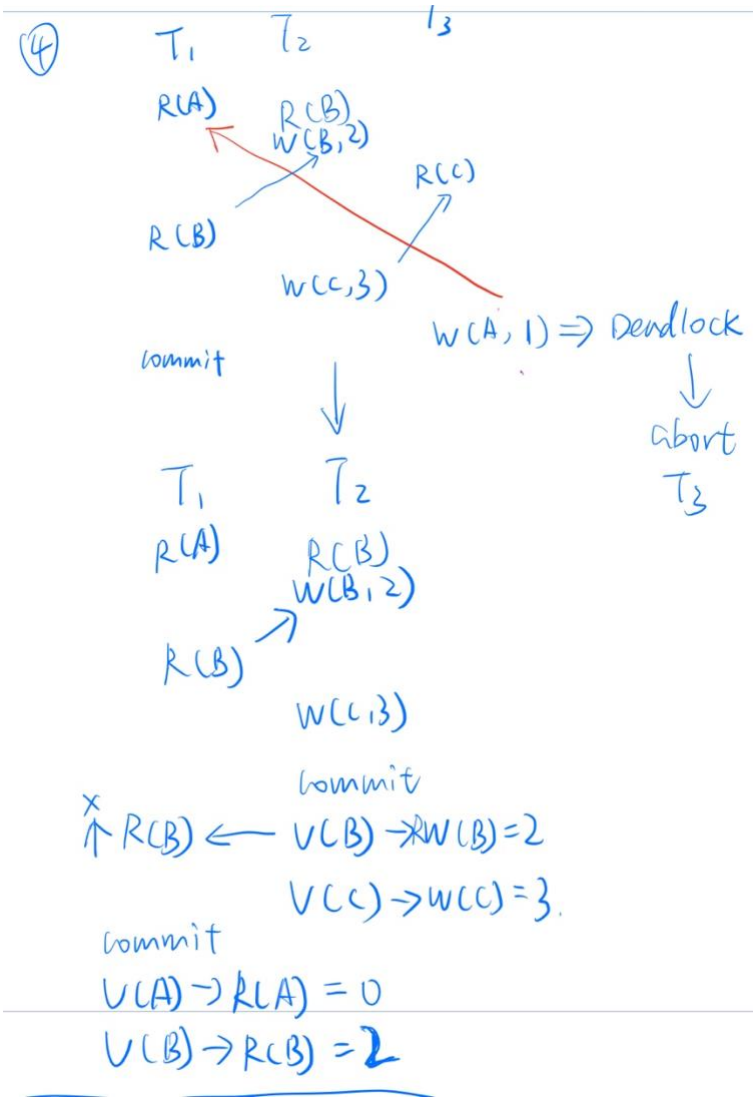
Test2: chain of unlock



We can see that T1 is the first transaction to execute. T1 unlock(a) leads to moving R(a) in T2 into acquire status. Then T2 is execute, T2 unlock(A) leads to moving w(a) in T3 into acquire status. Then T3 executed.

2)Deadlock detection and resolve test

When W(a) in T3 add to wait list, it trigger the cycle of T1->T2->T3->T1. Therefore the T3 is aborted. Then there is only the T1 and T2. The execution order is shown in below gigure.



4. Challenge
    1. It is the first time I use RMI. For a function, argument is passed message and return value is return message. My first design is based on this. But it has a fatal disadvantage. 2 phase locking has 2 phases of growing and shrink. If one operation from data site to central site doesn't require return. If there is no return message, then data site cannot get granted lock message. Then I figure out, we can just send message without returning. I can establish connection on both sides so that they can send message whenever they want and don't need to wait for response.
    2. It is hard to figure out how to find a simple way of graph. When I check online for implement graph, there are many websites mention needs to have edge, vertical and graph which is too much work. Then I find out what I need is the just a wait relation. An 2-demension array can also handle this situation.
    3. It is also hard to figure out how to detect cycle by myself. I admit I use the way on the internet to use status to keep track of passed transaction.