Janani Vijayarajan

CS 542 Distributed Databases
Project Report

1. Statement of the problem

Concurrency controlling techniques ensure that multiple transactions are executed simultaneously while maintaining the ACID properties of the transactions and serializability in the schedules.
Concurrency control is provided in a database to:
(i) enforce isolation among transactions.
(ii) preserve database consistency through consistency preserving execution of transactions.
(iii) resolve read-write and write-read conflicts.

Various concurrency control techniques are Two-phase locking Protocol , Timestamp ordering Protocol, Multi version concurrency control and Validation concurrency control.

The lock based approach is simple to implement and one of the most common locking methods is 2 Phase Locking. Our project aims to implement the Centralized 2-Phase locking method.

A transaction is said to follow the Two-Phase Locking protocol if locking and unlocking can be done in two phases:
Growing Phase: New locks on data items may be acquired but none can be released.
Shrinking Phase: Existing locks may be released but no new locks can be acquired.

Most importantly, 2-PL ensures serializability. But still, there are still some drawbacks of 2-PL such as deadlocks and cascading rollback are possible and our project contains the feature of deadlock detection when simultaneous transactions are executed.

2. Design, methods or Procedures

Since our project is implementing Centralized Two phase Locking, we have two main components: Central Site and Data Sites. This project was coded using Java. We have implemented the strict two phase locking variation.

The idea of Centralized 2PL is to delegate lock management responsibility to a single site only. This means that only one of the sites has a lock manager, the

transaction managers at the other sites communicate with it rather than with their own lock managers. This approach is also known as the primary site 2PL algorithm.

Communication between the central site and the data sites to execute simultaneous transactions in different data sites was established using Java RMI.

The central manager is associated with a lock manager for managing and sharing the resources of locks for different transactions which request locks for the data items in the database.

Each data site is associated with a data manager and a transaction manager for processing the transactions. Request processing will take place at the data site where the transaction is submitted. The Data Manager module consists of creating the database and connecting to it and takes care of reading and writing the updated data value to the database. The Transaction Manager module takes care of processing the operations in the transactions.

All transactions arriving at different sites are sent to the central site. Apart from lock manager, a wait for graph is maintained at the central site to detect deadlocks occurring situations. Also a queue is maintained at the central site to manage the request and release of locks. The waitforgraph module for deadlock detection is based on the JBOSS API and it was integrated with our project.

The database used in this project is SQLite3. When the data site is running, the database will be automatically created. The write operations performed inserts the value of data item when that particular data item is operated for the first item else updates the value of the data item in the database. Mathematical operations such as addition or subtraction are allowed in the transactions. The updates are performed in such a way that the most recent write operation value will be updated for every data item respectively in the database.

Most importantly, all the updates are ensured at all sites and they are posted in the same order.

The overall implementation is based on the C2PL-TM, C2PL-LM, C2PL-DP algorithms from section 11.3.1 from the textbook “Principles of Distributed

Databases". The textbook link can be found here: https://vulms.vu.edu.pk/Courses/CS712/Downloads/Principles%20of%20Distributed%20Database%20Systems.pdf

The logic of Transaction Manager is:

Algorithm 11.1: Centralized 2PL Transaction Manager (C2PL-TM) Algorithm

```
Input: msg : a message
begin
    repeat
        wait for a msg ;
        switch msg do
            case transaction operation
                let op be the operation ;
                if op.Type = BT then DP(op)          {call DP with operation}
                else C2PL-LM(op)                     {call LM with operation}
            case Lock Manager response               {lock request granted or locks released}
                if lock request granted then
                    find site that stores the requested data item (say H_i) ;
                    DP_Si(op)                        {call DP at site S_i with operation}
                else                                 {must be lock release message}
                    inform user about the termination of transaction
            case Data Processor response             {operation completed message}
                switch transaction operation do
                    let op be the operation ;
                    case R
                        return op.val (data item value) to the application
                    case W
                        inform application of completion of the write
                    case C
                        if commit msg has been received from all participants then
                            inform application of successful completion of transaction ;
                            C2PL-LM(op)              {need to release locks}
                        else                         {wait until commit messages come from all}
                            record the arrival of the commit message
                    case A
                        inform application of completion of the abort ;
                        C2PL-LM(op)                  {need to release locks}
    until forever ;
end
```

The logic of Lock Manager and Data Processor is given below:

Algorithm 11.2: Centralized 2PL Lock Manager (C2PL-LM) Algorithm

```
Input: op : Op
begin
    switch op.Type do
        case R or W                          {lock request; see if it can be granted}
            find the lock unit lu such that op.arg ⊆ lu ;
            if lu is unlocked or lock mode of lu is compatible with op.Type
            then
                set lock on lu in appropriate mode on behalf of transaction
                op.tid ;
                send "Lock granted" to coordinating TM of transaction
            else
                put op on a queue for lu
        case C or A                          {locks need to be released}
            foreach lock unit lu held by transaction do
                release lock on lu held by transaction ;
                if there are operations waiting in queue for lu then
                    find the first operation O on queue ;
                    set a lock on lu on behalf of O ;
                    send "Lock granted" to coordinating TM of transaction
                    O.tid
            send "Locks released" to coordinating TM of transaction
end
```

Algorithm 11.3: Data Processor (DP) Algorithm

```
Input: op : Op
begin
    switch op.Type do                        {check the type of operation}
        case BT                              {details to be discussed in Chapter 12}
            do some bookkeeping
        case R
            op.res ← READ(op.arg) ;          {database READ operation}
            op.res ← "Read done"
        case W                               {database WRITE of val into data item arg}
            WRITE(op.arg, op.val) ;
            op.res ← "Write done"
        case C
            COMMIT ;                         {execute COMMIT }
            op.res ← "Commit done"
        case A
            ABORT ;                          {execute ABORT }
            op.res ← "Abort done"
    return op
end
```

In this project, we have implemented the strict 2PL version of two phase locking. In case a deadlock is detected between two transactions, the most recent transaction is aborted and the other finishes its execution. We have used the wait for graph dependency graph to check for cycles in the graph for deadlock detection.

3. Experiments

A number of experiments were conducted in order to ensure the correctness of the output.

The first experiment was to verify the output in presence of deadlock situations. To increase the chance of deadlocks, I tried to run the same transactions in every data site and the transactions were correctly executed by aborting the transaction causing the deadlock and other transactions were processed successfully.

The second experiment was trying to run very long transactions which also increases the deadlock scenarios as the transaction which is operating would require more time to release all the locks acquired. Even in this experiment, the transactions were correctly executed by aborting the transaction causing the deadlock and other transactions were processed successfully.

The third experiment I tried is to run a very large number of transactions in various data sites to check the robustness of the system. It took around 2 hours to finish them all. No exceptions or crashing occurred during this test.

Data Collected:

The following output shows the output snippets of two different data sites when deadlock is detected twice while running the transactions in two data sites.

Data Site 1:

```
Connecting to Central Site at localhost:45
Starting transaction 10001
Transaction 10001 is committed
Transaction 10001 completed
Starting transaction 10002
Transaction 10002 is committed
Transaction 10002 completed
Starting transaction 10003
Transaction 10003 is committed
Transaction 10003 completed
Site 1 waiting for next transaction
```

Data Site 2:

```
Connecting to Central Site at localhost:45
Starting transaction 20001
 Writing the info to the datasite 2
Transaction 20001 is aborted
Starting transaction 20002
 Writing the info to the datasite 2
Transaction 20002 is aborted
Starting transaction 20003
 Writing the info to the datasite 2
Transaction 20003 is committed
Transaction 20003 completed
Site 2 waiting for next transaction
```

It can be seen from the above output that transactions 20001 and 2002 are aborted in data site 2 when deadlock was detected. The sites remain in a blocked state when the lock manager cannot process them. Checking for deadlocks is done every 10000 milliseconds in the central site. When a deadlock is detected, the most recent transaction is aborted.

```
Checked for Deadlocks in the waitforgraph and the number of Deadlocks found so far: 2
```

After all the transactions are completed, the database is updated with the most recent update.

```
C:\Users\S Abhishek\Desktop\2PL\2phaselocking>sqlite3 c2pl.db
SQLite version 3.38.2 2022-03-26 13:51:10
Enter ".help" for usage hints.
sqlite> select * from datastore;
A|4
Y|8
B|60
X|4
sqlite>
```

4. Experiences

The problems that I faced during the development of the project was establishing connections between central and different data sites using Java RMI. The official Java RMI tutorial was helpful to figure out a way to solve the errors. Another problem that I encountered was figuring out the classpath settings to successfully run the Java RMI connections. Finding solutions for these problems was a time consuming task.

The most interesting part of the project that I enjoyed was designing the architecture of the project. It consisted of designing the connections between different components of the project as it consists of several modules such as Central Site, Lock Manager, Data Manager, Transaction Manager and Deadlock Detection. .

I also enjoyed creating various transaction files for various scenarios to detect deadlocks for checking the robustness of the project.

5. Observations

By running different experiments, I observed with the increase in the number of data sites, there is also a linear increase in deadlock occuring scenarios. Also, this project is good enough for small transactions, but if we run larger transactions, it is severely affected by deadlocks and therefore aborts many transactions.

For future works, the project could be modified with additional features like:
(i) The aborted transactions should be stored and restarted for later execution.
(ii) Deadlock prevention and Avoidance schemes can be added to reduce the number of aborted transactions.