TEAM MEMBERS

LAUKIK KATHOKE ID: 1001558835  
AMBIKA PATI ID: 1001510620

2PL Protocol for Concurrency Control

In this project we demonstrate a more restrictive variation of 2PL that is rigorous 2PL using wait-die deadlock prevention protocol. It uses the concept of transaction timestamp which is an unique identifier assigned to each transaction. Suppose that transaction Ti tries to lock an item X but is not able to because X is locked by some other transaction Tj with a conflicting lock. Then to avoid the deadlock following rules are followed by wait-die protocol:

If TS(Ti) < TS(Tj) then Ti is allowed to wait

If TS(Ti) > TS(Tj) then Ti is aborted (dies).

Design of code

The entire code is designed based on three modules

1. Transaction table

It will contain the following attributes:

Tran\_ID – an integer value that will uniquely identify a transaction

Tran\_timestamp – an integer value that records the order of beginning

of transactions.

Trans\_state – state can be any of the one i.e(Active, blocked, aborted,

cancelled)

Items\_locked – The list of items that are locked by the transaction.

Oper\_waiting – The list of operations that are waiting to be executed.

Data Structure:

As the number of transactions can vary with different inputs, a data structure is needed that will be dynamically resizable to accommodate all the transactions. Also speed being another important factor in transaction processing, Hashmap is used to insert and update this table. It address both the benefits discussed above. ArrayList is used to maintain the waiting operations.

1. Lock table

It will contain the following attributes:

Item\_name: It consist of the name of the items that are currently locked

Lock\_state: It is used to store the value of lock state which can be

read(shared) lock or write(exclusive) lock.

Trans\_holding\_lock: It contains the value of transactions that are holding the

lock.

Trans\_waiting: It contains list of transactions waiting for the item to be

unlocked.

Data Structure:

Again, Hashmap seemed to be the better choice here and it is used to insert and update this table. Arraylist is used to store waiting transactions. List will be used to store and maintain list of transactions holding lock.

1. Rigorous\_two\_phase

The core method that will implement two phase locking protocol using wait-die deadlock prevention protocol. The input file will be read and the switch statement will be used to consider different cases(begin, read, abort, etc) for operations. A short summary of the action taken for each operation is printed as the output.

Programming Language & Softwares:

Java language will be used to develop the project.

Operating System used will be Windows 10.

IDE used will be Eclipse.

Pseudo-code

PROGRAM Rigorous\_two\_phase //Start program

tran\_Table //declaring transaction table

lock\_Table //declaring lock table

READ input file

PARSE the input

FUNCTION checkOper(input)

GET character at zero index //first character of input

SWITCH(character at zero index)

CASE b //For Begin operation

ADD Tran\_ID in Tran\_Table

CREATE Tran\_timestamp

SET Tran\_state to active

CASE r //For Read operation

IF Tran\_state is active

Request read lock on item

IF item is not locked

ADD item\_name in Lock\_Table

SET Lock\_state to ReadLock

ADD Trans\_ID in Trans\_holding\_lock

ADD Items\_locked in Tran\_table

ELSE

IF item is read locked

UPDATE Trans\_holding\_lock in

Lock\_Table

ELSE

IF item is write locked //conflicting opr

CALL WaitDie()

END IF

END IF

END IF

END IF

CASE w //For Write opeartion

IF Trans\_state is active

Request write lock on item

IF item is not locked

ADD item\_name in Lock\_Table

SET Lock\_state to WriteLock

ADD Trans\_ID in Trans\_holding\_lock

ADD items\_locked in Tran\_Table

ELSE IF item is read locked by same

Transaction

UPGRADE Lock\_state from

ReadLock to WriteLock in Lock\_Tabl

ELSE IF item is read locked OR write

locked by some different

Transaction //conflicting oper

CALL WaitDie()

END IF

END IF

END IF

END IF

CASE e //For end operation

IF Trans\_state is active

UPDATE current Trans\_state to Committed

UPDATE Lock\_Table to RELEASE all locks

UPDATE Trans\_Table to RELEASE all locks

ELSE IF Trans\_state is blocked

UPDATE current Trans\_state to

Blocked

ADD current operation to Oper\_waiting

ELSE IF Trans\_state is abort

UPDATE current Trans\_state to

Aborted

UPDATE Lock\_Table to RELEASE

all locks

UPDATE Trans\_Table to

RELEASE all locks

END IF

END IF

END IF

FUNCTION PSEUDOCODE:

1. Wait-Die function

FUNCTION WaitDie(conflicting\_Transaction)

IF(conflicting\_Trans\_timestamp is older than previous

Trans\_timestamp)

Conflicting\_Transaction Waits

UPDATE conflicting\_Trans\_state to blocked  
 ADD item corresponding to blocked\_Trans to oper\_waiting in   
 Tran\_Table  
 ADD blocked\_Trans to Trans\_waiting in Lock\_Table

ELSE

Conflicting\_Transaction Dies

UPDATE conflicting\_Trans\_state to aborted  
 IF aborted\_Trans in Trans\_holding\_lock in Lock\_Table  
 REMOVE aborted\_Trans  
 ENDIF

IF aborted\_Trans in Trans\_waiting in Lock\_Table  
 REMOVE aborted\_Trans

CALL Execute\_waiting\_Transactions

1. Execute waiting Transactions function

FUNCTION Execute\_tran\_waiting

FOR tran\_waiting in Lock\_Table

START waiting transaction

GET waiting\_operation of waiting\_transaction

PASS waiting\_operation to checkOper FUNCTION