Outline: RepCRec v1

Team: Patrick Yuen, Sanaya Bhatena

Goal:

This application will simulate a distributed database, complete with multiversion concurrency control, deadlock detection, replication, and failure recovery.

Algorithms:

- 2-phase Locking for Read/Write Transactions
- Cycle Detection for Deadlock Detection
- Multi-version reads for Read Only Transactions

System Specifics:

There are 20 Integer Variables with 10 Sites. Ex: x<Variable>.<Site>

- Odd variables exist only at Site: 1 + i % 10
- Even Variables are replicated at all sites: 1- 10

Input:

All input will be either read from the standard input or a file.

User Commands:

```
begin(<Transaction>)
beginRO(<Read only Transaction>)
R(<Tranasction>, <Var>)
W(<Tranasction>, <Var>, <Value>)
dump() //Display Everything
dump(<Site>) //Display all variables at Site
dump(<Variable>) //Display variable at all sites
end(T1) //End Transaction
fail(<Site>) //? Can't read/write to it anymore
recover(<Site>) //? Now can read/write
```

Example: sample_test.txt

```
begin(T1)
begin(T2)
fail(3)
fail(4)
R(T1,x1); W(T2,x8,88)
end(T1)
recover(4)
recover(3)
R(T2,x3)
end(T2)
```

Output:
All output will be to a specified file. Output will always contain time step by timestep the output of each operation and whether transactions have been aborted/committed

Ex:

```
...
t: 4 -- Input [ read(T1, x6, 2); write(T2, x6, 2); write(T3, x4, 2) ]
t: 4 -- Output: [ T1:Read@x6 -> 2; T2: Aborted due to deadlock ; T3:Write@x4 -> 2;  ]
…
End Status:
T1 Committed
T2 Aborted
T3 Committed
Etc…
```

High Level Design Document: *Some lower level functions likely to change

Language: Java

Main Work Flow:

```
Step 1: Main Driver parses file, line by line
          V
Step 2: Transaction Manager executes each command
          V
Step 3: Transaction Manager executes each operation in command
          V
Step 4: Request Lock from Lock Manager
           => If denied  Place in appropriate queue, goto step 3
          V
Step 5: DataManager Performs Operation
          V
Step 6: Determine if Deadlock from Lock Manager
          => If Deadlock, abort youngest and provide locks to next transactions in
queue, execute their operations, then goto Step 2
          V
Step 7: Check if more operations exist
          => if more operations exist, goto step 3
          V
Step 8: Check if more commands exist
          => if more commands exist, Goto Step 2
          V
Step 9: End
```

Data Types:

Transaction:

- Int timeStamp //Timestamp when transaction was created
- String id //Transaction ID
- List<Operation> operations_remaining //List of blocked operations
- List<Integer> waitsFor //List of blocked Transaction indexes

ReadOnlyTransaction: (extends Transaction)

- Int[] commitedValues //Copy of all commited values

Operation:

- String transactionId //Transaction ID
- Int Variable //Variable to Access
- Int Value //Value to write to variable
- ENUM Type: Read or Write

Result:

- Boolean success //If result was successful
- Boolean deadlock //If operation created a deadlock
- String output //Output of Successful result

DataElement:

- Int lastCommitedVal //Last Commited Value
- Int value //Current Value

Lower Level Components:

RepCRec_Driver.java:

Purpose: Main Driver to initialize DB and Parse Input File

CmdLine Arguements: <FileName for Output> [FileName for Input (Optional)]

Methods:

void initDB()

- Input: NA
- Output: NA
- Effects: Initializes all Sites and variable values

void run(String input)

- Input: Entire String input of Execution Sequence
- Output: NA
- Effects: Parse out commands, feeds them to Transaction Manager, and writes output to output file

Transaction Manager:

Purpose: Create new Transactions, execute operations, and synchronize output from each site

Global Variables:
Site[] sites //Array of all 10 Sites
Int time //Current Time Tick
List<Transaction> transactions //List of all Current Transactions
List<Queue<Operation>> waitingOperations

Methods:

List<String> runAllSteps(String input)
- Input: String of simultaneous commands
- Output: List of Results from each command
- Effects: Execute to Waiting operations if possible, then Execute each step

String execute(Operation, Transaction)
- Input: Operation and corresponding Transaction
- Output: String of input
- Effects: Request locks for each corresponding site, and attempt to execute operation, if cannot, then place operation on queue, then see if operation creates deadlock

Result read(Operation, Transaction)
- Input: Operation and corresponding Transaction
- Output: Result of Operation
- Effects: Request locks and attempt to execute operation

Result write(Operation, Transaction)
- Input: Operation and corresponding Transaction
- Output: Result of Operation
- Effects: Request locks and attempt to execute operation

List<Site> getSites(Operation)
- Input: Operation
- Output: List of Corresponding Sites to variable
- Effects: NA

void fail(Integer site)
- Input: Site Number
- Output: NA
- Effects: Abort every transaction that has a write lock in that site and fail site

void abort(Transaction) //Remove all locks and pop transactions off queue

Input: Transaction

Output: NA

Effects: Goto each variable you have a write lock on a reset value to last committed value, unlock each lock

void commit(Transaction)

Input: Transaction

Output: NA

Effects: Goto each variable you've written to and write value to committed value, unlock each lock

void dump()

Input: NA

Output: NA

Effects: Output all DB state to STDOUT

void dump(int variable)

Input: NA

Output: NA

Effects: Output variable state to STDOUT

void updateWaitingOps()

Input: NA

Output: NA

Effects: If can acquire lock, execute first operation in each queue

List<Transaction> createDeadLock(Operation) //DFS:

Input: Operation

Output: Null or list of transactions

Effects: DFS on transactions in internal waitsFor data structure to find all transactions involved in cycle

Site.java:

Purpose: Object representation of each Site

Global Variables:

LockManager lockManager //Creates Locks on Variables:

DataManager dataManager //Interface for Writes/Reads Data

Boolean recovery //If replicated variable, cannot read but can write if true

Boolean failed //deny all operations

Methods:
void fail()
- Input: NA
- Output: NA
- Effects: set fail flag, Clear all locks and values

Lock Manager

Purpose: Manage/Assign Locks and place operations in queues

Global Variables
HashMap<Integer, Integer> readLocks //variable =>Transaction Index
HashMap<Integer, Integer> writeLocks //variable =>Transaction Index

Methods:
boolean lock(Operation)
- Input: NA
- Output: If can access appropriate lock
- Effects: Attempt to access proper lock and return result

boolean writeLock(Operation)
- Input: NA
- Output: If can access write lock
- Effects: Create Write lock if can access

boolean readLock(Operation)
- Input: NA
- Output: If can access read lock
- Effects: Create read lock if can access

void unlock(Integer variable)
- Input: NA
- Output: NA
- Effects: Unlock lock at variable

void unlockAll()
- Input: NA
- Output: NA
- Effects: Unlock locks for all variables

Data Manager:

Purpose: Interface to perform operations

Global Variables:

```
DataElement[] elements //where all data is stored

Methods:
Integer read(Operation)
        Input: NA
        Output: Integer output of variable
        Effects: read actual value

void write(Operation)
        Input: NA
        Output: NA
        Effects: write value
```