Design Document for Final Concurrency Control and Recovery Project

Fall 2022

Tanran Zheng (TZ408) & Daria Xu (XX2085)

Overview

This project aims to implement a distributed database system, which has following features:

(1) multiversion concurrency control;
(2) deadlock detection (cycle detection with youngest victim selection);
(3) replication (available copies algorithm using strict two phase locking);
(4) failure recovery (under available copies algorithm).

Our program mainly consists of two parts: the data management part and the transaction management part. The data management part includes the objects of:

I. Data variables
II. Locks
III. Site: responsible for multiversion concurrency control and other bookkeeping.
IV. Data manager (DM): contains all Sites. It is responsible for managing replication, failure recovery, and translating requests on variables to requests on copies using the available copy algorithm[1].

The transaction management part includes objects of:

I. Transaction
II. Transaction manager (TM): responsible for translating the read/write requests to transactions, and sending the transactions to DM to translate requests on variables to requests on copies. TM is also responsible for deadlock detection[2].
III. Waitlist manager: maintaining the waitlist.

A separated waitlist object is also included to manage the wait list.

Please see Figure 1 for diagrams of major components. The project will be implemented using Python (version 3.6).

[1] We define DM differently than how the syllabus does. In this project, DM is responsible for the implementation of the available copy algorithm
[2] Per footnote 1, TM in this project is only responsible for sending transactions to DM and deadlock detection. The DM will perform the available copy algorithm instead.

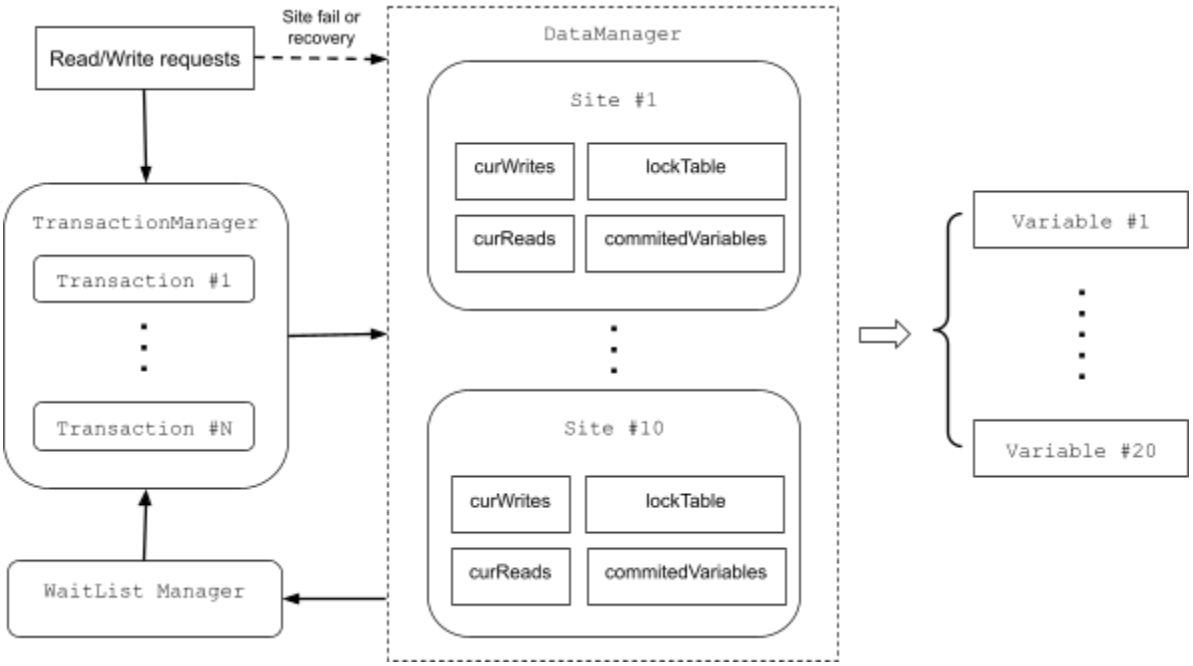

Figure 1 Diagram of major components

Valid operations and corresponding part to execute the operation:

- Begin: transaction manager
- beginR0: transaction manager
- Fail: data manager
- recover: data manager
- R (read-only): transaction manager
- W (read-write): transaction manager
- end: transaction manager
- dump: transaction manager

Details of Major Components

Data Management

Class Variable

```The data variable object.
```

Variable

Name
Value
lastCommittedTime

Class Lock

```The lock object. Stored in the lock table of each site.
```

Variable

State: R_Lock, RW_Lock # read or write lock
Transaction: own by which transaction
Lineup: if there are transactions waiting for this lock to release

Class Site

```The site object, representing each site. Contains lock table and transaction information of this site, as well as functions to operate this site.
```

Variable

Dictionary lockTable
    Key: variable
    Value: lock object

Dictionary curWrites: store write actions that are not yet committed
    Key: transaction name
    Value: {variable: value}

List curReads: store transactions (name) that are currently accessing variable on this site

Dictionary committedVariables
    Key: variable name
    Value: variable object

Dictionary copies: store the copy of variables for RO Transaction

Major Functions

Recover():

```Recover this site
```

Fail():

```
Fail this site, notify all transactions that are accessing this site(curReads and curWrites) to abort, clear the curReads, the curWrites and the lock table.
```

write(t, x, v):
```
    Parameters:
        t: transaction name
        x: name of the variable to write on
        v: the value to write

    Add to dictionary curWrites (curWrites[t][x] = v)
```

read_only(t,x):
```
    Parameters:
        t: transaction name
        x: name of the variable to read
    Returns:
        The value of x

    Read variable from the recorded copies.
```

read(t,x):
```
    Parameters:
        t: transaction name
        x: name of the variable to read
    Returns:
        The value of x

    Read variable from the committed variable, and add t to the curReads
```

lock_variable(t, x, type):
```
    Parameters:
        t: transaction name
        x: name of the variable to lock
        Type: the type of lock

    Check if lock can be acquired, then create a lock object if the lock can be acquired and add it to the
    lock table. Otherwise, return the list of transaction that is blocking this lock.
```

Commit(t):
```
    Parameters:
        t: transaction name
```

Commit the variable from curWrites to committedVariables.
```

**Absort(t):**
```
Parameters:
t: transaction name

Remove any t from curWrites and curReads. Remove any locks issued by t from the lock table.
```

## Class DataManager

```The object that contains all sites. It is also responsible for replication, failure recovery, and translating transactions (given by TM) on variables to requests on copies using the available copy algorithm.
```

Variable

List sites=[Site1, Site2, …, Site10]
Initialize each site following the rule: The odd indexed variables are at one site each. (i.e. 1+ (index number mod 10)). For example, x3 and x13 are both at site 4. Even indexed variables are at all sites. Each variable xi is initialized to the value 10i (10 times i).

Major Functions

Recover(i):
```
Parameters:
i: Site number

Recover site i
```

Fail(i):
```
Parameters:
i: Site number

Fail site i
```

RequestReadOnly(t, x):
```
Parameters:
t: transaction name
x: variable name
Returns:

```
        If read successfully, returns (True, the value read);
        otherwise, return (false, list of transaction blocking this read)

    Check each available site, then read from the site.
```

RequestRead(t, x):

```
    Parameters:
        t: transaction name
        x: name of the variable to write on
    Return:
        If read successfully, returns (True, the value read);
        otherwise, return (false, list of transaction blocking this read)

     Check each available site, then read from the site.
```

RequestWrite(t, x, v):

```
    Parameters:
        t: transaction name
        x: name of the variable to write on
        v: the value to write
    Returns:
        If read successfully, return (True, []);
        otherwise, return (false, list of transaction blocking this write)

    Check if locks can be acquired on all active sites. If true lock the variable, then do write on each site.
```

CommitOnAllSites(t):

```
    Parameters:
        t: transaction name

    Commit on all active sites.
```

AbortOnAllSites(t):

```
    Parameters:
        t: transaction name

    Abort on all active sites.
```

Transaction Management

Class Transaction

```
The object represents a transaction.
```

Variable

Name: transaction name
startTime: transaction start time
readOnly: a boolean showing whether the transaction is a read-only transaction
Abort: a boolean showing whether the transaction should be aborted

Class WaitlistManager

```
Response for maintaining the waitlist and deadlock detection.
```

Major Functions

add_to_waitList(t, op, args, blockedBy)

```
Add operation (Read or Write) and arguments to the waitlist, storing information about which transaction requesting this operation and which transactions are blocking this operation for wait-for graph.
```

DeadLock_Detection()

```
Detecting deadlock, if deadlock detected, return the youngest transaction along the path.
```

Class TransactionManager

```
TM object. Responsible for creating Transactions based on requests and sending them to the DM.
```

Variable

DataManager DM # initialize with the number of sites
Dictionary transactions

Key: transaction name
Value: transaction object

Waitlist: an WaitList Object

Major Functions

StartTransaction(t):
```
Parameters:
t: transaction name

Create a transaction object, initialize it with the transaction name and current time as the start time.
```

StartROTransaction(t):
```
Parameters:
t: transaction name

Create a transaction object, initialize it with the transaction name, current time as the start time and set the readOnly boolean to True.
```

EndTransation(t):
```
Parameters:
t: transaction name

Check if this transaction need to abort. (abort boolean) Do commit or abort
```

Abort(t):
```
Parameters:
t: transaction name

Call DM to abort this transaction. Remove transaction from the transaction dictionary. Remove any operations of this transaction from the waitlist.
```

Commit(t):
```
Parameters:
t: transaction name

Call DM to commit this transaction. Remove transaction from the transaction dictionary.
```

```

**Read(t,x):**
```
    Parameters:
        t: transaction object
        x: name of the variable to write on

    1) If t is a read-only transaction, call the DM to process the read-only request.
    2) If t is not a read-only transaction, call the DM to process the read request.

    If read failed, add this operation to the waitlist or abort.
```

**Write(t, x, v):**
```
    Parameters:
        t: transaction name
        x: name of the variable to write on
        v: the value to write

    Call DM to process the write request. If write failed, add this operation to the waitlist.
```

# Main()

```
    Main function that reads the input and uses TM to process the requests and corresponding executions. It also maintains the Waitlist etc.
```

Major Functions

ProcessInput(file):
```
    Read input from text file, and return a list of operations
```

Run(executions, dataMgr, transMgr)

```
    Execute operation one by one, at the beginning of each tick, will try to execute the operation from the waitlist. If last operation was added to the waiting list, at the very beginning of the tick, it will do deadlock detection. If there is a deadlock, abort the youngest transaction until there are no more deadlocks.
```
```