

Replicated Concurrency Control and Recovery

Design Document

Xinyi Liu

Ming Xu

December 7, 2019

1 Introduction

This project is to implement a tiny distributed database, complete with multiversion concurrency control, deadlock detection, replication, and failure recovery. The program reads input instructions from a file or standard input, and outputs the results on screen.

2 Design

2.1 Overview

This project is written in Java. The program consists of several data models (`Transaction`, `Operation`, `Variable`, `Lock`), business logic managers (`TransactionManager`, `DataManager`, `LockManager`), and a runner (`Database`).

2.2 Data Models

Below are the data models we have to maintain this distributed database.

```
public class Transaction {
    private int id;
    private int timestamp;
    private TransactionType type;           // Enum, can be READ_ONLY or READ_WRITE
    private boolean isBlocked;
    private boolean isAborted;
    private Set<Integer> accessedSites;
}

public class Operation {
    private int timestamp;
    private int transactionId;
    private int variableId;
    private OperationType type;           // Enum, can be READ or WRITE
    private int value;
}

public class Variable {
    private int id;
    private int valueToCommit;
    private int transactionIdToCommit;
```

```

    private int lastCommittedValue;
    private Map<Integer, Integer> committedValues;
                                // <timestamp, committedValue>
    private boolean isReadable;
}

public class Lock {
    private int transactionId;
    private int variableId;
    private LockType type;      // Enum, can be READ_LOCK or WRITE_LOCK
}

```

Constructors, builders, setters and getters are omitted in this design document.

2.3 Business Logic

TransactionManager handles all transactions including read or write on variables and events of different sites. It is able to detect deadlocks.

```

public class TransactionManager {
    private Map<Integer, DataManager> sites;
                                // <siteId, dataManager>
    private Map<Integer, Transaction> transactions;
                                // <transactionId, transaction>
    private List<Operation> waitingOperations;
    private Map<Integer, Set<Integer>> waitsForGraph;
                                // <transactionId, Set<transactionId>>

    public void begin(int transactionId, int timestamp);
    public void beginRO(int transactionId, int timestamp);
    public void end(int transactionId, int timestamp);
    public int read(int transactionId, int variableIndex, int timestamp);
    public void write(int transactionId, int variableIndex, int value, int timestamp);
    public void dump();
    public void fail(int siteId);
    public void recover(int siteId);
}

```

DataManager provides the storage of variables and manages their locks for a certain site.

```

public class DataManager {
    private int id;
    private boolean isActive;
    private Map<Integer, Variable> variables;
                                // <variableId, variable>
    private Map<Integer, LockManager> lockManagers;
                                // <variableId, lockManager>

    public int getId();
    public boolean isActive();
    public boolean containsVariable(int variableId);
    public boolean canRead(TransactionType transactionType, Operation operation);
    public void read(TransactionType transactionType, int timestamp, Operation
        operation);
    public boolean canWrite(TransactionType transactionType, Operation operation);
}

```

```

    public void write(TransactionType transactionType, Operation operation);
    public List<Integer> getLockHolders(int variableId);
    public void abort(int transactionId);
    public void commit(int timestamp, int transactionId);
    public void dump();
    public void fail();
    public void recover();
}

```

Each LockManager maintains locks for a single variable on a certain site.

```

public class LockManager {
    private int siteId;
    private int variableIndex;
    private List<Lock> locks;

    public int getVariableId();
    public boolean canAcquireLock(OperationType operationType, int transactionId);
    public Lock lock(OperationType operationType, int transactionId, int variableId);
    public void unlock(int transactionId);
    public void unlockAll();
    public boolean isWriteLockedBy(int transactionId);
    public List<Integer> getLockHolders();
}

```

Note that private methods in implementation are omitted in this design document.

2.4 Runner

A main class is required to initialize TransactionManager, read and parse the input, and perform right actions.

```

public class Database {
    public static void main(String[] args);
}

```