Design Document for Replicated Concurrency Control and Recovery Project

Project Requirement:

Implement a distributed database, complete with multi-version concurrency control, deadlock detection, replication, and failure recovery.

Input:

Input from a text file for testing text cases.

Implementation Modules:

Main:

- Takes the input files and parses it.
- Parses begin, beginRO, W, R, end, dump, fail, recover functions from the input file and call appropriate functions in the transaction manager.

Transaction Manager:

- Initializing sites.
- Translates read and write requests on variables to read and write requests on copies using the available copy algorithm.
- Assigns variables to be stored at the appropriate sites.
- If the Transaction Manager requests a read for transaction T and does get it due to failure then the transaction manager tries another site.
- The read-only transactions have access to the latest version of each variable before the transaction begins.
- Add transactions to a waitlist.
- Allows transactions to be retried after being waitlisted.
- Commits the values upon the end of a transaction.
- Aborts transactions which cannot access.
- Detects a deadlock and then aborts the transaction with a higher timestamp.

Transaction:

- Defines transaction id, timestamp, type, status and states of the transactions.
- Assigns appropriate values to each transaction.

Lock Manager:

- Initializes and defines read and write lock tables.
- Adds transactions into the read and write lock tables upon acquiring locks.
- Releases read and write locks.
- Raises an exception if a transaction tries to write to a variable that has already been locked.

Variable:

- There are 20 distinct variables x1... x20. The numerical portion is treated as index number.
- The odd indexed variables are present on one site each using formula 1 + index number mod 10. So, x3 and x13 are both at site 4.
- The even indexed variables are at all sites.
- Each variable xi is initialized to the value 10i. So x1 is initialized to 10, x2 to 20 and so on.

Site:

- Each site has an independent lock table.
- If that site fails, the lock table is erased.
- Initializes variables. Variable are stored at the appropriate sites using the technique mentioned above.
- Has a dump function which implements:
- dump() gives the committed values of all copies of all variables at all sites, sorted per site.
- dump(i) gives the committed values of all copies of all variables at site i.
- dump(xj) gives the committed values of all copies of variable xj at all sites.
- Sets the status of the site as failed or recovered.
- Commits and aborts transactions. Releases all locks upon commit and abort.

Algorithms:

- Available Copies Algorithm using two phase locking (using read and write locks) at each site and validation at commit time. (Read from one available site, but write to available sites).
- Deadlock Detection Protocol: Use cycle detection and abort the youngest transaction in the cycle.
- Multiversion read consistency algorithm for readonly transaction. So readonly transactions read the values of indexes that were committed at the time the transaction started.

Example script:

INPUT:

```
begin(T1)
begin(T2)
W(T1,x1,101)
W(T2,x2,202)
W(T1,x2,102)
W(T2,x1,201)
end(T1)
dump()
```

OUTPUT:
T2 should abort, T1 should not, because of kill youngest.

```
=== output of dump
x1: 101 at site 2
x2: 102 at all sites
All other variables have their initial values.
```

Steps for execution:

Terminal 1 (Server)

1) Start the 10 sites: (Ports 9090 - 9099 should be open) $ ./Start.sh

2) Start transaction manager (Port 7777 should be open) $ python TransactionManager.py

Terminal 2 (DB Client)

3) Run Main Class file (client) and provide a test file $ python MainClass.py <path-to-test-file>

Terminal 1 (Server)

4) Restart sites and transaction manager to continue to test a new test case: $ ./Restart.sh

Note: To stop sites and transaction manager $ ./Stop.sh

Diagram :

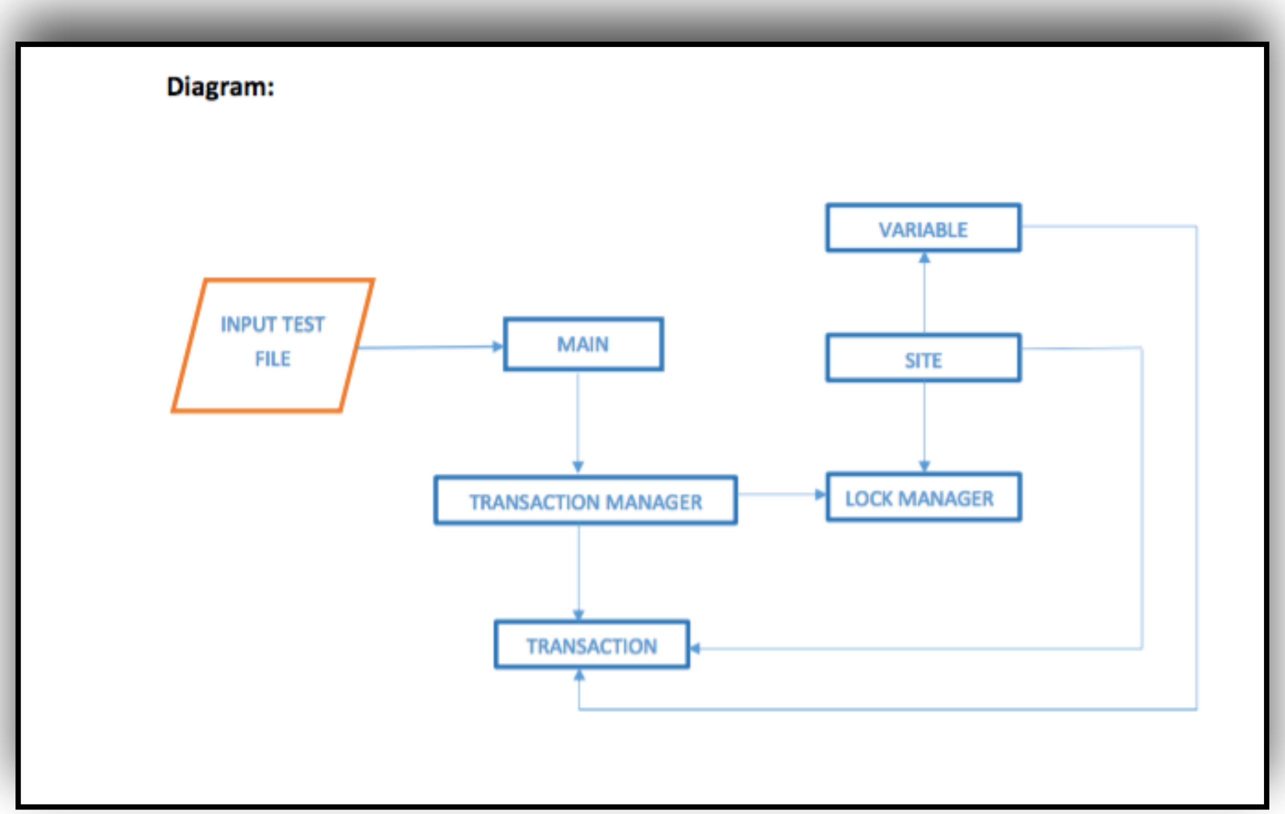
Diagram:
INPUT TEST FILE
MAIN
VARIABLE
SITE
TRANSACTION MANAGER
LOCK MANAGER
TRANSACTION