Advanced Database Class Document and Flow Discussion

Team Members:

Sree Gowri Addepalli (sga297)

Sree Lakshmi Addepalli (sla410)

Replicated Concurrency control and Recovery in distributed databases:

We try to build a database system that helps us simulate multiple transaction processing at one attempt and thereby help us build features of multiversion concurrency control, deadlock detection, replication and failure recovery.

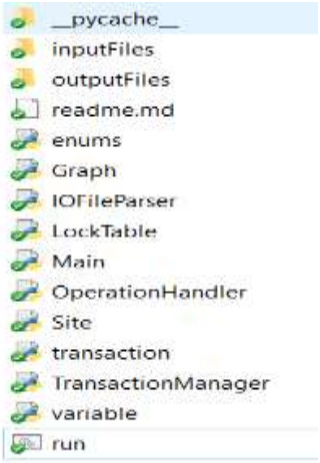
Following are some features of this project:

1. The system has 10 sites and 20 variables with even numbered replicable variables present at every site and odd number non replicated variable present at a single site.
2. We have used available copies algorithm and two-phase strict locking.
3. Failed sites are no meant for performing transactions.
4. A recovered site will not allow a variable to read until a write happens.
5. Deadlock detection is resolved using by building wait for graph
6. All the variables are updated only if the transaction fully commits.
7. Read only transaction have values of the initial system as they are read only and any transaction that alters the state of the system doesn't affect them.

In this python project several classes and enums have been used:

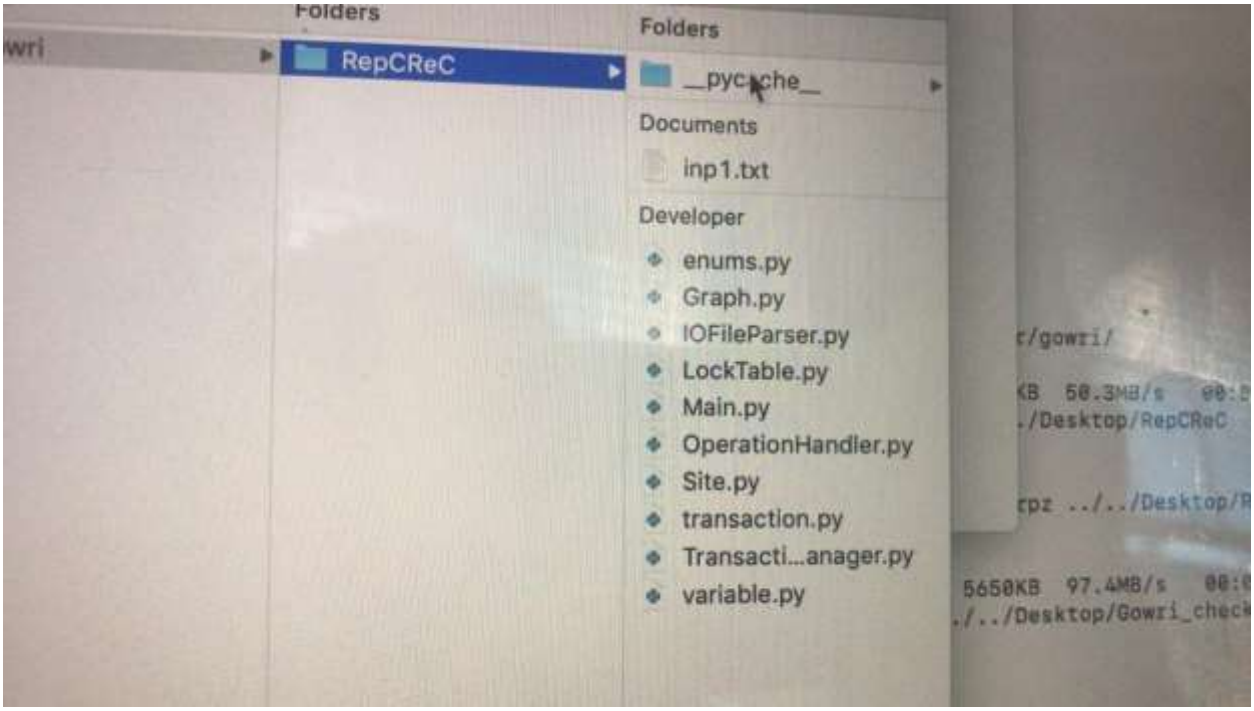
Without Reprozip and running on windows

File Structure:



The file structure consists input files folders where you can put in all your testcases and output files where the output of these test files are there. There is a run.bat file which you can run on windows machine to generate the output of all the testcases at once.

With Reprozip and running on linux



Below are the classes used:

1. Main (Class):

This is the starting part of the class which takes the file Input path, filename from command like arguments to start reading the commands of the file.

2. IOFileParser (Class):

Responsible for reading each line in the file of operations. Every line is sent to transaction manager for each line to be processed as an instruction.

Variables:

- ListofInstructions – stores each instruction in the file to be processed by the transaction manage
- tm - Transaction Manager object.

Methods:

- fileRead(): reads the file.
- executeOperation (): for each instruction in the file execute the operations.

3. Enums.py:

This is a python file containing all the enums used in the project.

- **SiteHealthSignal** - Enum for telling the health signal of a site as UP, DOWN and RECOVER
- **TransactionOperationType** - Enum for telling the type of transaction operations as Read and Write.
- **LockType**: - Enum for telling the type of Lock as Read and Write.
- **TransactionType**: Enum for telling the type of Transaction as ReadOnly and ReadWrite
- **TransactionHealthSignal** - Enum for telling the health signal of any particular transaction as Active, Committed, Failed, Aborted, PartiallyCommitted, Waiting, Deadlocked.

4. LockTable (Class):

Represents a lock and holds information of type of lock and Transaction that holds it. Takes type of the lock, R or W variable number with which this lock is associated and transaction that holds this lock as inputs.

Variables:

Type – The type of Lock from LockType Enum.

TransactionNumber – The associated transaction number with this lock.

VariableNumber - The associated transaction number with this lock.

lastSetTime- The time associated when the lock is taken.

Methods:

getTransactionNumber (): Gets transaction number of this lock.

getVariableNumber(): Gets variable number of this lock.

getType(): Returns type of lock

getTime(): Returns the time set in nanoseconds when the lock was set.

5. Graph (Class):

Represents the transactions in graph data structure.

Variables:

Graph - The dictionary mapping with vertex and its neighbours..

Methods:

- **addTransactionEdge():** Add an edge to the adjacency list with vertexes u,v.
- **getNeighboursOfATrans():** Get the neighbours of a Transaction of vertex u.
- **removeNeighbourOfATrans():** Remove the neighbours of a Transaction with edge u,v.

- **addTransactionVertex():** Add a vertex to the graph.
- **removeTransactionVertex():** Remove the nodes of a vertex(Transaction) and its corresponding neighbours.
- **detectCycleUtil():** Detect a cycle in a graph.
- **dfs():** depth first search for finding visited vertex.

6. Variable (Class):

Variable represents each variable(data) stored on the sites and takes the variable number and value to be set as parameters.

Variables:

Index – The variable number.

value – The value associated with the variable.

timeVersionsValues – The dictionary of variable value with time in nanoseconds in key and variable value

lastSetTime- The time associated when the variable is taken.

Methods:

getVariableValue(): Gets variable value of this variable.

setVariableValue(): Sets the variable value of this value.

getLatestCommittedValue(): Get the latest committed value of the variable below the given input time.

getVariableIndex() – returns the variable index.

replicateVariable() – returns a deep copy of the variable.

8. **OperationHandler(Class):**

Variables:

- **OperationType –** The type of operation from TransactionOperationType.
- **transactionNum –** The associated transaction number with this operation.
- **timestamp-** The associated time with the operation Handler.
- **variableNumber-** The variable number with the operation.
- **variableValue –** The variable value with the variable with the operation.

Methods:

- getOperationType(): Get the operation type of this operation.

- getTransactionNum(): Gets transaction number of this operation.
- getTimestamp(): The associated time with the operation.
- getVariableNumber(): returns the variable index
- getVariableValue() – get the variable value associated with the operation.

8. Transaction (Class):

Transaction objects holds information of the transaction by taking transactionNumber, timestamp, and transaction Type.

Variables:

- TransactionType – The type of transaction from TransactionType from BeginRO and Begin.
- transactionNum – The associated transaction number with this transaction.
- timestamp- The associated with the transaction.
- operations – list of operations associated with this operation.

Methods:

- getTransactionType (): Get the transaction type of this transaction.
- getTransactionNumber (): Gets transaction number of this Transaction.
- getTimeStamp (): The associated time with the transaction.
- getOperations(): Return get the list of operations associated with this Transaction.
- addOperation() – Add operation to the list of Operations.
- removeOperations() - Remove all the operations related to the set of Transactions

9. Site (Class): Site instance that stores data

Variables:

- **siteIndex -** The site number.
- **SiteHealthSignal -** The site status from the enum class.
- **variableList -** The variable list is variable id and variable.
- **LockList -** The lock list is variable id and locks on that.
- **transactionList -** The transactionID and the the list of operations to be committed.

Methods:

- getSiteSignalHealth(): return site status.
- getLockTable(): get the list of locks on a certain variable.
- recoverSite(): recover the current site.
- failSite(): fail the current site and abort transactions on this site..
- dump(): Get the value of every variable at every site.
- dumpNum(): Get the value of particular given variable at every site.
- addOperationList(): This method takes in a transaction number and the Operation.
- dropLock(): drop the lock on the given variable with a specific transaction.
- addLock() - add the lock on the given variable with a specific transaction.
- isCommit() - commit a given operation of a transaction.
- removeTransaction() - Remove all the locks held on this site for a corresponding transaction and remove corresponding transaction.

10. TransactionManager (Class): Responsible for processing transactions. Detects deadlock and resolves it.

Variable

variableSiteMap - Map of variables with the list of site they are in

transactionMap - Map of Transaction numbers to set of site numbers.

pendOperations - list of pending operations waiting to be completed.

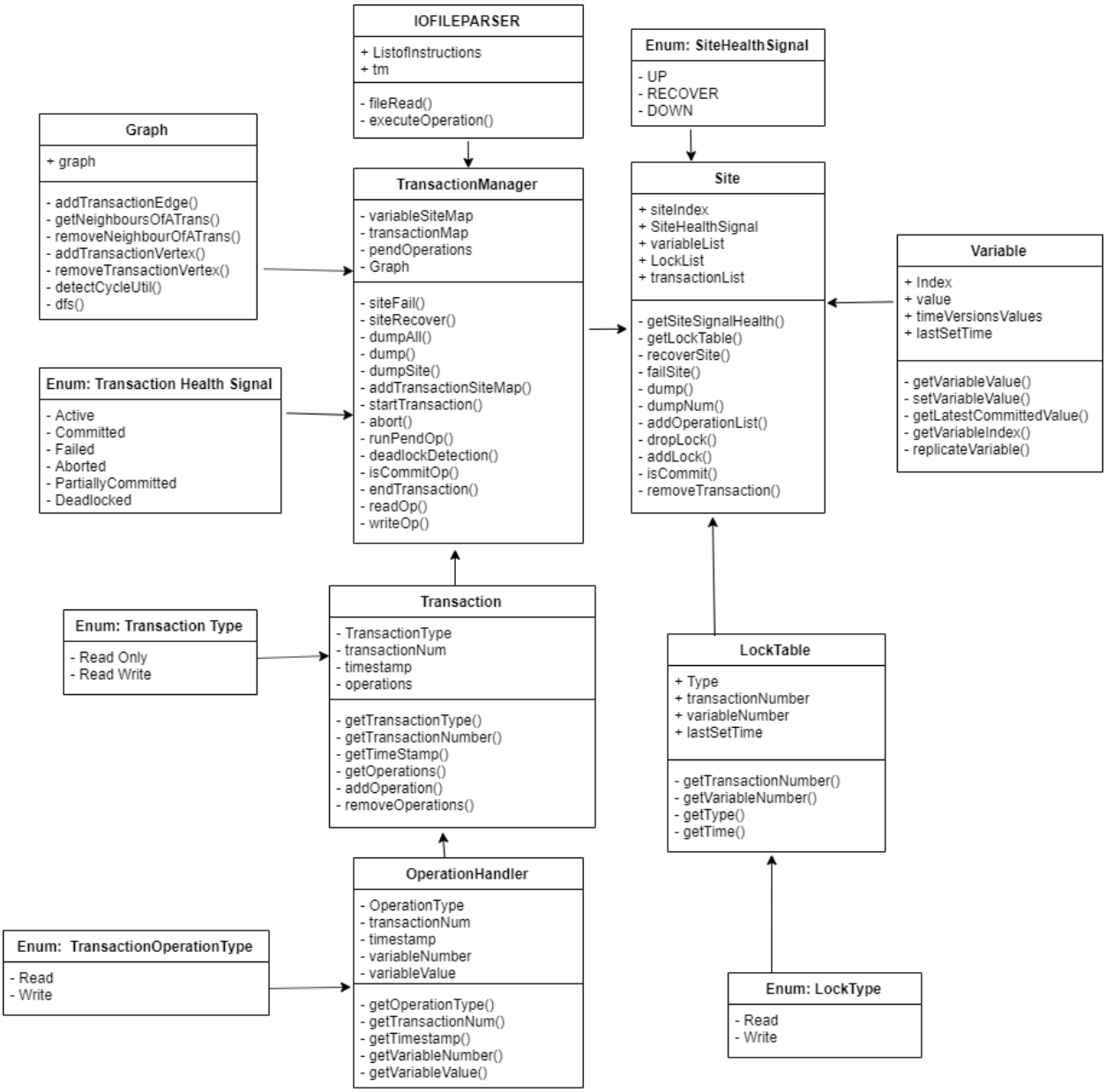
Graph - the graph object for representing transactions.

Methods:

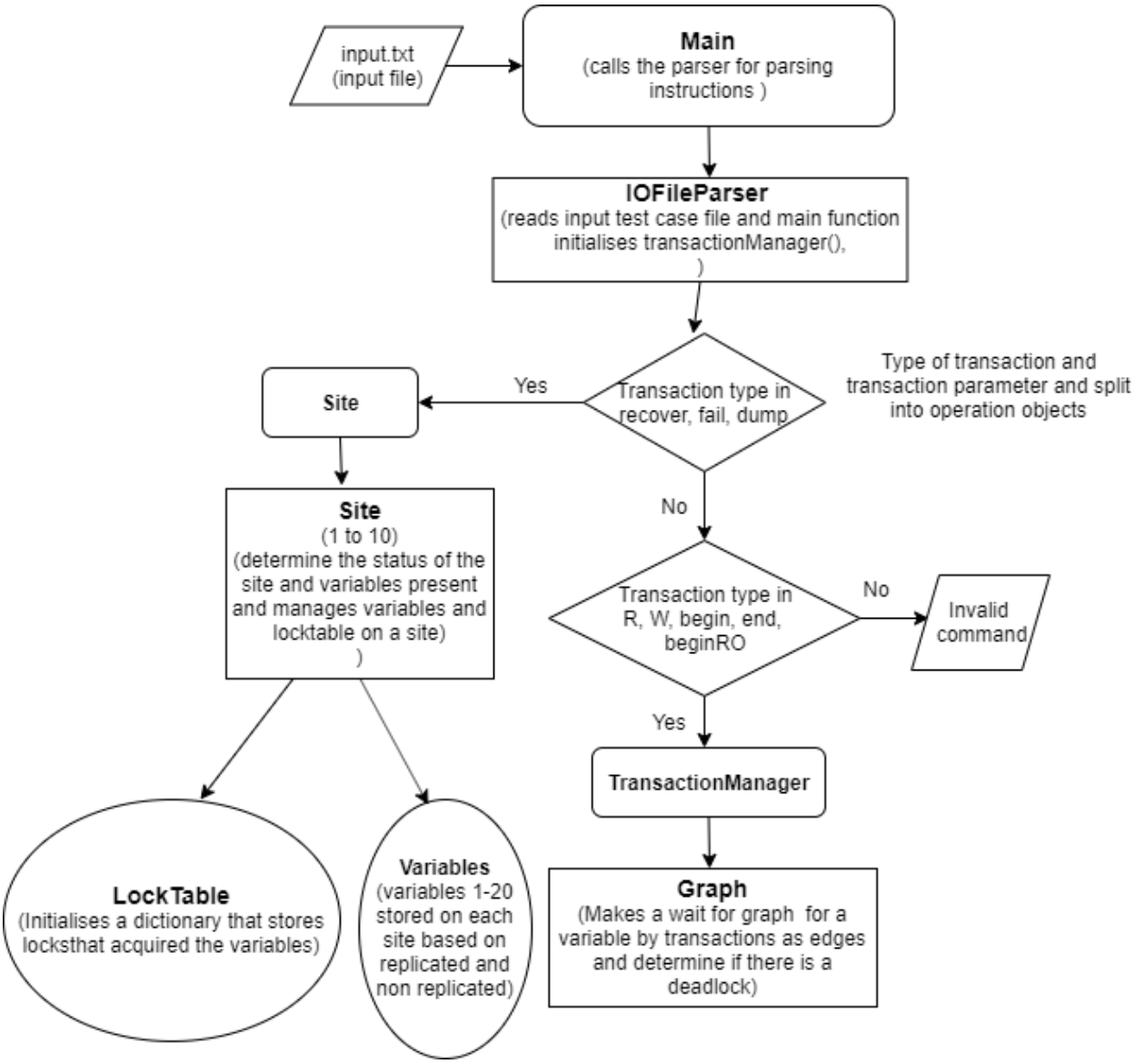
- siteFail(): Fails a corresponding site with a particular site number.
- siteRecover(): Recover a particular site and wait for read operations.
- dumpAll() : Print the value of variables at every site.
- dump(): Print the value of the respective variable at every site.
- dumpSite() - print the value of variable at every site.
- addTransactionSiteMap() - add particular transaction with it corresponds site number.
- startTransaction() - starts the corresponding transaction based on it being ReadOnly or ReadWrite.
- abort() - abort the corresponding transaction number.
- runPendOp() - execute the corresponding operations in the pending operation list.
- deadlockDetection() - detect deadlock in the graph.
- isCommitOp() - whether the transaction has committed or not.
- endTransaction() - end the transaction based on the end operation
- readOp() - the command to read a particular variable of a transaction.
- writeOp() - the command to write a particular variable of a transaction with a particular value.

CLASS DIAGRAM:

Below is the class diagram for the project.



PROJECT FLOW:



Transaction flow diagram

For any set of transactions at a time,

1) The instructions of the transactions are read and processed by the transaction manager, which is the main centralized control for delegating instructions as seen in the above class diagram.
2) It is a distributed event driven object-oriented system that when a set of instruction takes place, it gets controlled by the states of various objects that maintain states which are represented by enums like lock type, transaction type, operation type, site status.
3) There are various levels of how the transaction flows and the variables that decide the flow are:

- Transaction Type - Read Only or Read Write
- Site Status - UP, DOWN or RECOVER
- Variable type - replicated or non-replicated
- Dependency parsing - Wait for edge for transactions processing the same resource or variable results in a deadlock or not and thereby giving access to the resource using lock.

4) Based on the kind of operation instruction given to the transaction manager, it delegates works to the site which here independently processes every variable at every site and gaining locks there, maintained as a lock table at every site.
5) Here, the site acts as a data manager as it is responsible for performing the same instruction by the transaction manager and updating variable value based on resources and locks on that site.
6) So, if a certain transaction operation cannot be executed a certain point of time, it goes into pending state and gets back when it has access to locks in the transaction manager.
7) If a site fails, transaction at that site gets removed and the transaction gets aborted.
8) Timestamp is maintained for every lock, transaction and operation along with variable updates so as to distinguish and help give resources based on a first come first serve rule.
9) Deadlock detection here is done using Depth first search for graph traversal on the wait for graph which is created as along a transaction proceeds ahead and many transactions are vying for the same resource which is detected after every transaction edge is added.
10) Pending operations are run when a transaction ends which is when it gets decided whether the operation would be committed or not based on resource availability. The operations abort and all the lock tables, transactions for a side related to it are dropped and so are the edges on the graph.