Fall 2021 Advanced Database Systems Project Design

Student1: MuCheng Luo (ml6389)

Student2: Jingwei Ye (jy3555)

Project: Replicated Concurrency Control and Recovery

**Summary:**

* There will be three main class: Transaction, DataManager and TransactionManager
* TransactionManager will read inputs from .txt file and create corresponding Transaction Objects.
* Then TransactionManager will deal with different operation types, check through locks and put operations into active queue or wait queue. There will also be deadlock check if there is new wait edge generated.
* Then Data Manager will lock corresponding variable or get the replicate data
* If there are operations commit/abort, TransactionManager will deal with those operations

**Transaction**

Fields:

* Int ID: transaction ID
* Int Timestamp: the begin time of a transaction
* ENUM Status: reflects the status of the current transaction (blocked, aborted, active)
* Bool ReadOnly: Boolean flag, indicates if a transaction is read only
* Int LockTime: the time this transaction acquires a lock

Methods:

* Transaction (int beginTime, int readOnly)

Constructor. Initialize a transaction

**DataManager:**

Fields:

* Int DM\_ID: Data Manager ID
* VariableArr: int[20], stores the values for all variables(x1, x2..x20)
* TransactionLockMap: Map<Integer, Integer>, a map in which the key is transaction id and the value is variable index.
* LockList:int[20], stores if the variable got locked.( the value of this array will be -1, which stands for read locked, 0, which means no lock, 1, which means write locked )
* SiteStatus: enum{Down, Up}, shows if the site is up( recovered ) or down( failed )

Methods:

* DataManager(int ID): constructor, init a site with an site\_id
* Void SetLock(int transactionID, int variable, int lockType): set lock for a variable due to a transaction. The lockType will be -1, stands for read lock and 1, stands for write lock.
* Void Fail(): fail the site
* Void Recover(): recover the site
* Void Dump(): print the values of the variables

**TransactionManager:**

* List[int] activeQ: stores the active transactions
* List[int] WaitingQ: stores the transactions that are waiting for the release of some lock

Methods:

* TransactionManger: constructor.
* Void readInputs(): parse and process the inputs stored in the input text file
* int Read(int transactionID, int variable): a read operation performed by a transaction with transactionID, return the read value
* Void Readonly(Int transactionID, int variable): read only performed by a transaction with transactionID, should use multiversion read consistency.
* Int readSnapShot(int transaction, int variable): read a variable from any sites, should return the variable value read
* Void Write(int transactionID, int variable, int value): a modify operation performed by transaction with transaction ID. Set the variable to value.
* Boolean acquireLock(int transactionID, int variable, int lockType): transaction with transaction ID acquire a lock in variable. LockType should be either 1 (read-lock) or -1 (write-lock). Return True if acquire is successful, False otherwise.
* Void addEdge(int T1, int T2): add a wait-for-edge from transaction T1 to transaction T2
* Boolean checkDeadLock(): check if there exists a cycle (deadlock) in the transaction graph
* Void abort(int transactionID): abort a transaction with transactionID
* Void commit(): commit appropriate transaction(s) in activeQ
* Void dump(): print the values of the variables
* Void begin(int transactionID): begin a transaction with transactionID
* Void end(int transactionID): end a transaction with transactionID