## Replicated Concurrency Control and Recovery - Design Document

CHEN-YUAN HO

LE WANG

### Execution

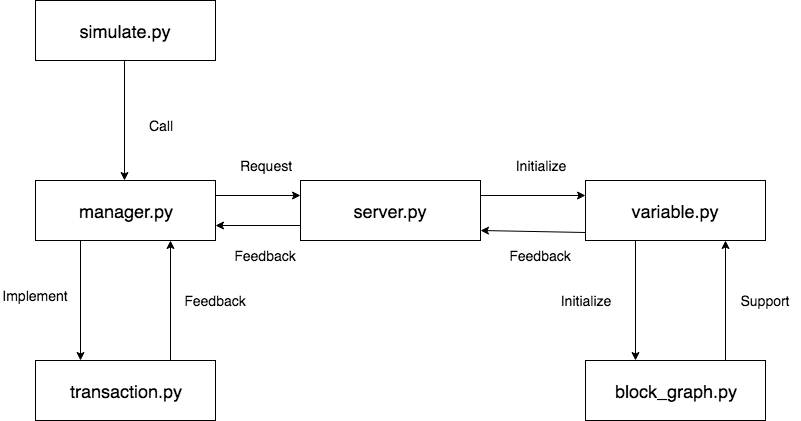
1. Requirements

Python 3.6.x

1. Command for Read from stdin and print to stdout

$ python3 simulate.py

### Flow Chart



### Components

1. simulate.py

* This is an interface provided to user to add instructions to the transaction manager(manager.py).

1. blocking\_graph.py

* The node of the blocking graph to detect deadlock.

|  |
| --- |
| **BlockingGraphElement** |
| * int id: the id of this node (describes which transaction add this node). * set prev: the edge from this node. |
| * void \_\_init\_\_: INT -> None |

1. variable.py

* This class describes the variable in each site(server).

|  |
| --- |
| **Variable** |
| * int id : INT : the id of this var * int value: the value of this var, the init value is 10 times id * dict locks: the lock table of this var, described as a hashmap(dict) in Python * bool canRead: the flag implies whether this var is readable |
| * void \_\_init\_\_ : * Called by a transaction, to add a lock to this var * Integer -> None * void addWriteLock : * Called by Variable itself, return ids of all transaction who hold a lock of this variable. * None -> None * set getPrioLock: * Called by Variable itself, return ids of all transaction who hold a lock of this variable. * None -> Set * void setValue: * Update the value of this variable * Integer -> None * void releaseLock: * Release all locks acquired by a particular transaction * Integer -> None |

1. server.py

* This class describes the server(database).

|  |
| --- |
| **Site** |
| * int id: the id of this site * dict vars: all vars hold by this site * bool online: true if this site is now up. |
| * void \_\_init\_\_ * bool exist: * Check whether this site contains a var. * Integer -> Boolean * String dump: * Return variable information of this site. * None -> String * Set fail: * Set this site down. Release all the locks and return all transactions' id who hold a lock on this site. * None -> Set * bool recover: * Recover this site. * List<Integer> -> Boolean |

1. transaction.py

* This class describes the transaction. A transaction is first initiated by user from stdin, and received instructions from transaction managerself.

|  |
| --- |
| **Transaction** |
| * int id: the id of this transaction * int timeStamp: the timeStamp when this transaction is initiated. A transaction is younger than anotherif and only if self.timeStamp > other.timeStamp * bool readOnly: describe whether this transaction is read-only * bool abort: ture if this transaction has been aborted * list<int , int , int> changes: all changes to the sites made by this transaction * list<int , int> log: all results read by this transaction * String reason: store the reason why this transaction is aborted |
| * void \_\_init\_\_ * void makeVarsCopy * Called by transaction manager when this transaction is read-onlyself to make a multi-version image of all readable vars in all online sites. * List<integer> -> None * Void acquireWriteLock * Acquire a write lock from a var * List<integer> , integer -> None * bool canWrite * To check whether this transaction can write to a particular var. * List<integer> , integer -> Boolean * Bool canRead * To check whether this transaction can read to a particular var. * List<integer> , integer -> Boolean * void write * Read a readable vars in any online site * List<integer> , integer , integer -> None * void write * Read a readable vars in any online site * List<integer> , integer -> None * void acquireReadLock * Acquire a read lock from a var * List<integer> , integer -> None * String commit * Commit all changes to target site. * Return all read results to the transaction manager. * Release all locks of this transaction. * List<integer> -> String * void setAbort * Set this transaction aborted. * List<integer> , deque<Transaction> , String -> None |

1. manager.py

* Transaction manager, which control the whole process.

|  |
| --- |
| Manager |
| * List<integer> sites: store information of all sites * dict curTrans: all transactions initiated by user from stdin. * deque: describes the wait-list for all instructions. |
| * void \_\_init\_\_ * String dumpAll * Called by user, return all info of all sites and vars. * None -> String * void run * Check whether the instruction in wait list hold all the lock required by the system, if so, run it. * None -> None * void detDeadlock * Deadlock detection function. Use the same cycle detection algorithm with topoSort. If a deadlock found, return all possible critical nodes. * None -> None * String formalize * Formalize the input from stdin. * String -> String * String dumpBySite * Return the value of the var from all sites. * Integer -> Integer * void parse * Parse the inputdate from user. * String , Integer -> None |