Replicated Concurrency Control and Recovery

RUGVED SAI KARNATI (rsk517) , TANMAY GATLE (tvg238)

INTRODUCTION:

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

ALGORITHMS:

1. Available Copies Algorithm for replication.
2. Strict Two-phase locking for Read/Write transactions.
3. Multiversion Read Consistency for Read-Only transactions.
4. DFS for deadlock cycle detection.

SYSTEM OVERVIEW:

The system consists of 10 sites numbered 1 to 10.
Odd indexed variables are stored on one site each => site no = 1 + (index_number % 10).
Even indexed variables are stored on each site => site no = 1….10.
Each variable is initialized to 10*index_number.
Each site has an independent lock table.

INPUT/OUTPUT:

Input - In the form of a file which contains all one operation per line (or) In the form of standard input.

Operations:
dump() - It gives the committed values of all copies of all variables at all sites, sorted per site with all values per site in ascending order by the variable name.
fail(x) - Site x fails.
recover(x) - Site x recovers.
end(y) - End of transaction y. It prints whether y is committed or aborted.
begin(y) - Begins transaction y
W(y,z,v) - write operation for transaction y, variable z with value v.
R(y,z) - read operation for transaction y, variable z

Output - Output will contain the result of each operation and whether the transactions have been aborted/committed. It contains whether the operation has succeeded or failed along with the reason for failure (lock cannot be acquired).

DATA STRUCTURES

```
// Map from Transaction Name to Transaction Object
HashMap<String,Transaction> transactions;

// Transaction Waits For Graph represented as an Adjacency List
HashMap<String,Set<String>> waitsForGraph;
```

// Map of List of Operations waiting for a lock for each variable (x1, x2, ..)
HashMap<String,ArrayList<Operation>> lockWaitOperations;

// List of Operations waiting due to site failure
ArrayList<Operation> failWaitOperations;

// Map of list of Fail times for each site
HashMap<Integer,List<Integer>> SiteFailHistory;

MODULES

Main: It contains the main loop which parses through the input and calls the appropriate function in the Transaction Manager. It allows the user to provide the input data as a file (command line argument) or via standard input (no command line argument).

Transaction: It contains all the operations that it has seen till now. It shows what state it is in i.e Waiting or Aborted or Committed. It has a Transaction Name, timestamp, boolean isReadOnly and an internal lock table which contains all the locks held by the transaction.

Site: It's a simulation of a server. It maintains a state of whether the site is up or down. It has a Data Manager and a Lock Manager. Any read or write operations executed at a particular site is directed to this class. The site then calls the appropriate functions in the Lock Manager or Data Manager.

Data Manager: Each site has its own Data Manager. It manages all the variables for its site. A tuple which contains current data and committed data is stored for each variable. Each variable also holds it's commit-history and a boolean whether it is active after recovery
(a write has been committed to this variable) or not.

Lock Manager: It maintains all the locks for the variables. Each site has a lock manager which processes requests for locks. It maintains a lock table which contains the transaction or list of transactions holding a lock for each variable along with the type of lock (read or write lock).

Transaction Manager: It translates the read and write request on variables to requests on copies using the available copies algorithm or multi version depending on the type of operation. It also processes whether to commit a transaction or abort it at the end of the transaction. It's a single global transaction manager. It is also responsible for deadlock detection and resolution. When a deadlock is detected, it aborts the youngest transaction.

High Level Functions (Pseudo Functions)

main(): (Tanmay, Rugved)

The driver code which parses and input and calls appropriate functions in the Transaction manager.

Lock Manager: (Tanmay)

- getreadlock(transaction_id, variable):

It checks whether this variable is locked. If locked, it checks for the type of lock. If it is read lock it gives this lock to the transaction.
This returns whether the transaction can get the read lock for the variable.

- getwritelock(variable,transaction_id):
  This returns whether the transaction can get the write lock for the variable.
- removeSiteLocks():
  This releases all the locks at the site. It is called after a site fails.
- removeLock(variable, transaction_id):
  It removes the lock held by the transaction for the given variable.
- tobeaborted(trasnaction_map):
  It changes the status of transactions that hold locks at this site to "TO_BE_ABORTED". It is called when a site fails so that the transactions which accessed this site before failure will be aborted in the end.

Data Manager: (Rugved)

- readCommitData(variable):
  This returns the latest committed value for that variable at the site.
- writedata(variable,value):
  It writes the value to the variable in the current site. It changes the current value of the variable.
- commit(variable, time):
  It commits the data for the given variable. It copies the current value at that variable to commitVal. It also stores the commit time of the variable in it's commit history. It also sets the variable so that it can be read after site recovery.
- findData(variable, time):
  It Returns the data committed at a time closest to the given time. It uses binary search technique to find the closest smaller value.

Transaction Manager: (Rugved)

- beginROTransaction(transaction_id):
  Starts the transaction at the current timestamp and maintains that the current transaction is RO so that the reads executed by this transaction will only take place on snapshots.
- beginTransaction(transaction_id):
  Starts the transaction at the current timestamp and sets transactions status to ACTIVE.
- snapshotResult(transaction_id, variable, siteNo):
  Returns a boolean which returns true if a read is allowed. We check reads on the snapshot by applying binarysearch and checking whether the site has failed after the last commit and only then allowing the read.
- readRequest(transaction_id, variable):
  Checks whether the transaction is read only or not. If it is read only, then based on the output of snapshotResult, we allow the read. Otherwise it checks if a read lock can be acquired (for the first available site in case of even variables) and performs the read or appends the operation to either lockwait or failwait operation lists.
- writeRequest(transaction_id,variable,value):
  It checks if a write lock can be acquired and performs the write or appends the operation to either lockwaitoperations or failwaitoperations.
- release_locks(transaction_id,booleans commit):

It releases all the locks held by the given transaction. It executes all the operations waiting for the locks to be released.

- endTransaction(transaction_id):
  It executes all the waiting operations of this transaction. It does the validation for the transaction. It commits if all the sites it accessed never failed until this point or else it aborts.
- fail(site):
  It calls fail() for that site which releases all the locks held by the site and status of the corresponding transactions change to TO_BE_ABORTED and the site status to FAIL. Also the time of failure is added to the site's fail history.
- recover(site):
  It calls recover() for that site which changes the site status to RECOVER and executes all the operations which were waiting for the failed site to recover.
- check_deadlock():
  It checks for deadlock in the waits-for-graph using the dfs algorithm. It is called whenever an edge is added to the waits-for graph i.e. whenever a transaction waits for a lock.
- dump():
  It outputs the data at all the sites for all the variables. One site per line.
- executeFailWaitOperations():
  It executes all the operations that weren't executed due to site failure.

Site: (Tanmay)

- changeStatus(SiteStatus):
  It changes the state of the site to the given status.
- getStatus():
  It returns the status of the site i.e. whether it is failed or recovered.
- removeLock(variable,transaction_id,boolean commit,time):
  The locks on the variable held by the given transaction are removed. It commits the data for the variable if commit is true. It changes the status of the site from recover to active if all the variables are active after site recovery.
- initialWrite():
  It writes initial data to all the variable at the site i.e variableNo*10.

Block Diagram: