Darren Levy and Viswanath Kammula
Due Date: December 2, 2014
Design Document for Replicated Concurrency Control and Recovery Project
Variables:

- An Outputter singleton object to place messages into a file.
- Tick variable to count for time (initialized to 0).
- Transaction objects get created with an ID and contain a state (ready, waiting, no-ready-site and aborted) and a boolean for readonly. They also contain a list of Instructions. Instructions are also objects containing whether it is read or write, an index, a value if necessary and a start time.
- Site objects each contain an array of size 21 to store values for an index, initialized to 10. (Index 0 won't be used in the array of indexes since indexes are numbered starting at 1.) There is another boolean array of size 21 representing each index to keep track of which indexes can be read (in case of Site failure). Only the indexes that the Site can write to are written to, ie odd indexes are stored at Site with ID (index number % 10) + 1 as stated in the syllabus. Each Site object also has a state (ready or failed) and contains two lock tables. One is a write lock table array. The transaction holding the lock is stored at the index that it holds. The read lock table is a list of lists since more than one transaction can hold a read lock. Again, each list of transactions lives at the index of the outer list where it holds the lock. For example, transactions at the third item in the read lock table hold a read lock on index 3. Lastly, there is a linked hash set of waiting transactions that take over locks as they are freed.
- A Transaction Manager singleton object receives the parsed commands and determines where to read and write values. Also, it has a list of all the transactions. And an array holding each site (again, 0-10 where 0 is not used).

Algorithms:

- Available Copies Algorithm (ready from one available site, but write to available sites)
- Wait-die avoidance protocol with optimization suggested in syllabus: abort younger transaction immediately, if they conflict with older transaction's lock. If older wait.
- Multiversion read consistency algorithm for read-only transaction. So read-only transactions read the values of indexes that were committed at the time the transaction started.

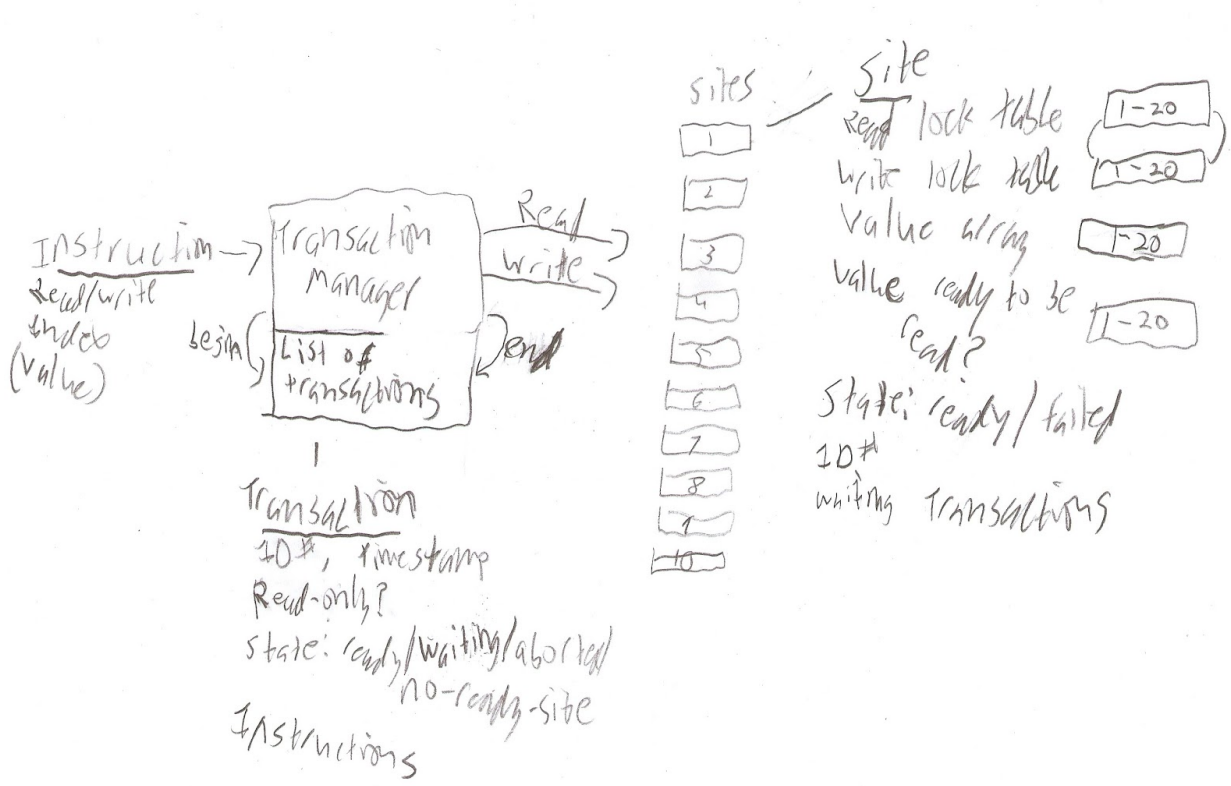
Example script 1:
begin(T1)
begin(T2)
beginRO(T3)
W(T1,x6,5) (T1 locks index 6 at all sites)
R(T3,x6) (T3 reads 10)
R(T2,x6) (T2 aborts because it is younger than T1.)
end(T1) (T1 commits 5 to index 6 at all sites)
end(T3)

Example script 2:

begin(T1)
W(T1,x3,100) (T1 locks index 3 at site 4)
fail(4) (T1 aborts)

Example script 3:
begin(T1)
begin(T2)
R(T1,x3) (T1 ready 10)
fail(2) (Site 2 fails)
W(T2,x3,20) (T2 aborts because it is younger than T1)
W(T1,x1,40) (T1 waits for Site 2 to recover)
recover(2) (T1 can now lock index 1 at Site 2)
end(T1) (T1 commits 40 to index 1 at Site 2)

Diagram:



Psuedo-code:

```
Transaction Manager(command) {
  if (command == Read) read(T,v);
  else if(command == Write) write(T,v,x);
  else if(command == Fail) fail(x);
  else if(command == Recover) recover(x);
  else if(command == End) end(x); // commit write values to sites
}
```

```
read(T,v) {
  If (T is Readonly )        //MultiVersion Read
    Read from the snapshot of the Database
  else {
    If( v is write Locked by T') {
      fetch the Transaction with lock T'
      if(T is younger than T') Kill T and exit
      else wait
  }
  else {
    Lock V with T
    read V
  }
}
```

```
write(T,v,x) {
  if( v is Locked by T') {
    fetch the Transaction with lock T'
    if(T is younger than T') Kill T and exit
    else wait
  }
  else {
    Lock v with T
    v=x //when T ends
  }
}
```

```
fail(x) {
  Fetch all T which Read or Write on x
  if(T is not committed) abort T
}
```

```
recover(x) {
  Recover site x,
  Allow Writes on X
  and Reads (except replicated non-written
data)
}
```

Major functions, their inputs, outputs and side effects (see Javadoc/code in repo for more):

Class DatabaseApp

public static void main(java.lang.String[] args)

This is the main method of the application. It takes in a path to a script file as args[0]. It then parses it and sends each instruction to the transaction manager.

Author: Darren and Viswanath

Class Site

public void promoteWaitingTransactions()
Performs instructions and acquires locks for waiting transactions.
Author: Darren

public java.lang.Integer readValueAtIndex(int index, Transaction transaction)
Reads value and acquires appropriate lock.
Author: Darren and Viswanath
Parameters:
index - the index to read from
transaction - the transaction that wants to read
Returns:
the value read

public void writeValueAtIndex(int index, int value)
When a value is written to a site its index is ready to be read again. This takes place at the end of the transaction when the values can be committed.
Author: Darren and Viswanath
Parameters:
index - the index to write
value - the value to write

public void recover()
When a site recovers its state is set to ready and its waiting transactions get promoted, if they can.
Author: Darren and Viswanath

Class Transaction Manager
public void intake(int tID,boolean readOnly, int timestamp)
This intake method is called when a transaction begins. If it is a read-only transaction, then a snapshot for that transaction is saved., if possible
Author: Darren and Viswanath
Parameters:
tID - the id of the transaction
readOnly - true if the transaction is read-only, false otherwise
timestamp - the time the transaction begins

public void intake(Instruction instruction, int tID)
This intake method is called when an instruction is either read or write
Author: Darren and Viswanath
Parameters:
instruction - the instruction with what to do
tID - the id of the transaction

private void abort(Transaction transaction, java.lang.String reason)
Sets the transaction's state to aborted and removes any locks it held on all sites.
Author: Darren

private void readonlySnap(Transaction transaction)
Creates a snapshot of the database for the given transaction.
Author: Darren and Viswanath
Parameters:
transaction - the read-only transaction

private void performWrite(Instruction instruction, Transaction transaction)
Performs the locks required to prepare the site(s) for a write. If the transaction cannot obtain the locks, it will wait or abort.
Author: Darren and Viswanath
Parameters:
instruction - the write instruction
transaction - the transaction that gave the instruction

private void performRead(Instruction instruction, Transaction transaction)
Reads the value for the instruction if it can. It also obtains the locks for the given index. If it can't get the lock, it will either wait or abort.
Author: Darren and Viswanath
Parameters:
instruction - the read instruction
transaction - the transaction that sent the instruction

public void endTransaction(int tID)
Called when a transaction ends, all its writes are committed and so values at sites are updated.
Author: Darren
Parameters:
tID - the id of the transaction to end.

public void dump()
Called when dump() instruction sent, this method dumps each site's indexes and corresponding values.
Author: Darren

public void shorterDump()
Called when dump(s) instruction sent, this method dumps the committed values at each site.
Author: Darren and Viswanath