Assignment #4

Building on Assignment #3:

Instead of using TL algorithm we will use (Transactional Locking II) TL2 which uses a Global Clock.

You must use:

- The versioned lock code (using one integer and atomic CAS) in versioned lock.cpp
  - This file has some macro representing the functions described in the lecture. You have the option to use them or rewrite them.
- Global Clock is a shared counter which is incremented using atomic instruction `__sync_fetch_and_add(&global_clock, 1)`
- The random function in rand_r_32.h

BONUSES:

1. Implement Read-Only Transactions Optimization (+2)
2. Implement Timestamp Extension (+2)

Like Assignment #3:

We have some bank accounts (represented by a number) initialized with $1,000, implement a program to do 100,000 transactions of 10 transfers between 20 randomly chosen accounts (in each transaction). The amount of money transferred is $50 and no negative accounts' balances are allowed.

You need to run the program using 1, 2, 4 threads. The number of transfers should be split on the number of threads. For example, 25,000 transfers per thread when we are running 4 threads.

The total amount of money before and after the transfers must be the same. You program should print the sum before and after executing all transfers.

Implement the program using Software Transactional Memory (STM). You should build a special purpose STM for this problem.

Your STM algorithm will follow Transactional Locking II.

Run your program with 4 different configurations (each with 1, 2, 4 threads):

1. 1,000,000 accounts
2. 1,000 accounts
3. 1,000,000 accounts with disjoint access (which means each thread access a unique set of the accounts. E.g., with two threads, one will access the first 500,000 accounts and the other will access the following 500,000 accounts)
4. 1,000 accounts with disjoint access

Measure the total execution time for each case and prepare a plot with 12 bars (3 threads settings * 4 configuration)

Notes:

You are free to use word-based or object-based implementation.

You must use barriers to ensure all threads started at the same time. The skeleton file from assignment #2 can be used to start (it has barriers and time calculations).

Deliverables:

1. Program implementation (C/C++ file(s))
2. Bars Graph + Tabular Results Data (any format, but I suggest Excel or Google Sheets for simplicity)

Dead line:

Feb 28th, 2018 at 12:00 PM