Programming Assignment 3

Due date: Thursday, April 13th by 1:59 PM

Design and Implementation of Transactional Linked List:
Using the method proposed in Deli Zhang's paper "Lock-free Transactions without Rollbacks for Linked Data Structures", transform the lock-free linked list from Assignment 2 into a transactional linked list.

Evaluation:
Evaluate your approach by executing performance tests of your transactional list. In your benchmark tests, vary the number of threads from 1 to 32. Each thread should execute a certain number of transactions (e.g. 100,000), and these transactions should perform operations on keys within a certain key range (e.g. 10,000). The key range is the maximum number that the list can contain. Produce graphs where you map the total number of committed transactions on the y-axis and the number of threads on the x-axis. It is preferred to produce the graphs in logarithmic scale.

Optionally, you may produce 3 different graphs representing different ratios of the invocation of insert, delete, and find. For example, your different ratios could be:

- Mixed: 33% insert, 33% delete, 34% find
- Write-dominated: 50% insert, 50% delete, 0% find
- Read-dominated: 15% insert, 5% delete, 80% find

Each graph should have at least 4 lines representing different transaction sizes. A transaction's size is equal to the number of operations in the transaction. For example, your different transaction sizes could be 1, 2, 4, and 8.

You may use Zhang's paper as a reference for the format of your graphs. However, note that there was an error in the gathering of the results for this paper, so your results should not look exactly the same as the results in the paper.

Experimental details:
In your experiments, make sure to pre-populate the list with nodes in the following way. Before the actual experiment begins, have one thread execute a number of transactions equal to the key range. Each of these transactions should execute a single Insert operation on a random key. After this pre-population step, the list should be about 60% full of nodes.

To avoid "polluting" your results with the overhead of memory management (the standard malloc and free don't scale well), you should have each thread pre-allocate its own supply of nodes, which it can keep on a private list when they are not in the list. Finally, to avoid pollution from calls to the pseudo-random (the standard rand isn't even thread-safe), you should pregenerate enough random integers to drive the choice between insert, delete, and find for the entire test run.

Deliverables:
Provide a brief summary of your approach and an informal statement reasoning about the correctness and efficiency of your design.

Use either C or C++ for this assignment and provide a ReadMe file with instructions explaining how to compile and run your program.