Sequential Linked Ordered Set

Robert Hernandez, *Student, UCF,* and Alaric O'Connor, *Student, UCF*

***Abstract*—We have implemented a sequential version of an ordered linked list with no duplicate entries. We describe its creation, and show that it does not perform especially well with multiple threads, as expected. Nothing new or creative was ventured.**

***Index Terms*—IEEE, IEEEtran, journal, LaTeX, paper, linked, list, multi-threading.**

I. Introduction

THERE have been many implementations of linked lists, sequential or otherwise. Ours is atypical in that it does not permit duplicates, and it keeps itself sorted from least to greatest at all times. This was not our decision; rather, our mission was to create a sequential version of such a structure. No special effort was made to ensure that it was particularly efficient. Function was the primary concern. There were no great challenges faced, nor risks taken.

February 10, 2016

A. Wait-Free Linked Lists

Shahar Timnat, Anastasia Braginsky, Alex Kogan, and Erez Petrank wrote a paper, titled "Wait-Free Linked Lists," in which they described their creation of the titular data structure, intended for multi-thread usage. As described in that paper, the linked list does not allow duplicate entries, and automatically keeps the list sorted from least to greatest. They claimed that this was "the traditional practice," for concurrent linked-list data structures, but we do not have sufficient knowledge to be certain of whether or not this is the case. Regardless, we were tasked to create a sequential version of this structure, described below.

B. Our Linked List

Java was chosen as our language due to mutual familiarity with it, as well as its extensive documentation. We created a "SequentialWaitFreeLinkedList" class, which contains a Node class. The latter can contain a reference to an object as its "item" value, and a reference to a node as its "Next" value, while the former has functions to insert or delete nodes with a particular "item" value, to find out whether there is a node with a particular "item" value, and to output a string representing the whole list. We are aware that this Linked List is in fact not Wait-Free at all.

As per the paper, the data structure that we implemented does not allow the user to determine where insertions will be made; instead, they are made wherever necessary to maintain a strict least-to-greatest ordering. The insertions, as per the rule disallowing duplicate entries, are also not guaranteed to succeed.

The wait-free linked list as described in the paper has elements which would be useless for our sequential version, so we did not include them. For example, our "next" pointers do not have a bit to signify logical deletion of the node. We also did not have a state array for each thread. Our deletion operation is not split up as it is in the paper, either. Although efficiency and speed of the algorithms did not factor into the process at all, there were no deliberately useless additions made for the sake of maintaining similarity to the Wait-Free Linked List.

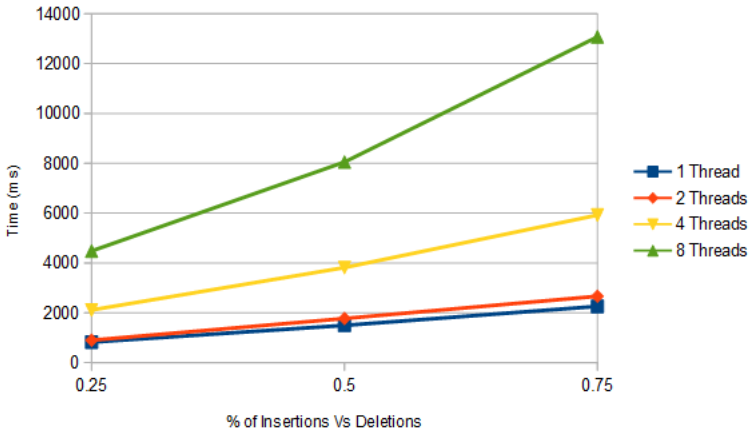
We created tests which were not intended to measure the performance of the Linked List, but merely to determine that it is in fact functional. Tests were performed on two computers. Our classes do function as intended.

C. The Process Of Creation

It is worth noting that we elected to use GitHub as a method of sharing code, with the intention of collaborating on a proper implementation of the wait-free linked list, in the future; it was almost unnecessary for this project, due to its simplicity. Initially, the basic linked list, without the special additional features, was written. The restrictive functions were added later.

D. Performance

Though our linked list is intended for single-threaded usage, we tested it, for later comparison against an implementation of the multi-thread version, using 1, 2, 4, and 8 threads, with a single global lock.



Different distributions of insertions and deletions were performed, shown by the x axis. During each test, 500000 operations were performed by all of the threads combined. As one can see, when additional threads are added, performance degrades greatly. Deletions appear to be much faster than insertions, judging by the trends of the lines.

II. Conclusion

We implemented a sequential version of an ordered set in the Java programming language and tested its performance using different numbers of threads to access it. It was not performant when additional threads were added. It is our hope and expectation that implementing the Wait Free Linked List on which it is based will achieve far superior results.

Acknowledgment

The authors would like to thank Dr. Damian Dechev, for giving us this assignment. If it were not for him, this would never have taken place.