Programming Assignment 5

Comparison of Optimistic and Lazy Synchronization in Sets implemented using Linked Lists

Optimistic Synchronization

Optimistic Synchronization is an improvement over single, coarse-grained, fine-grained locking. Single, coarse-grained, fine-grained locking impose a long sequence of lock acquisitions and releases.

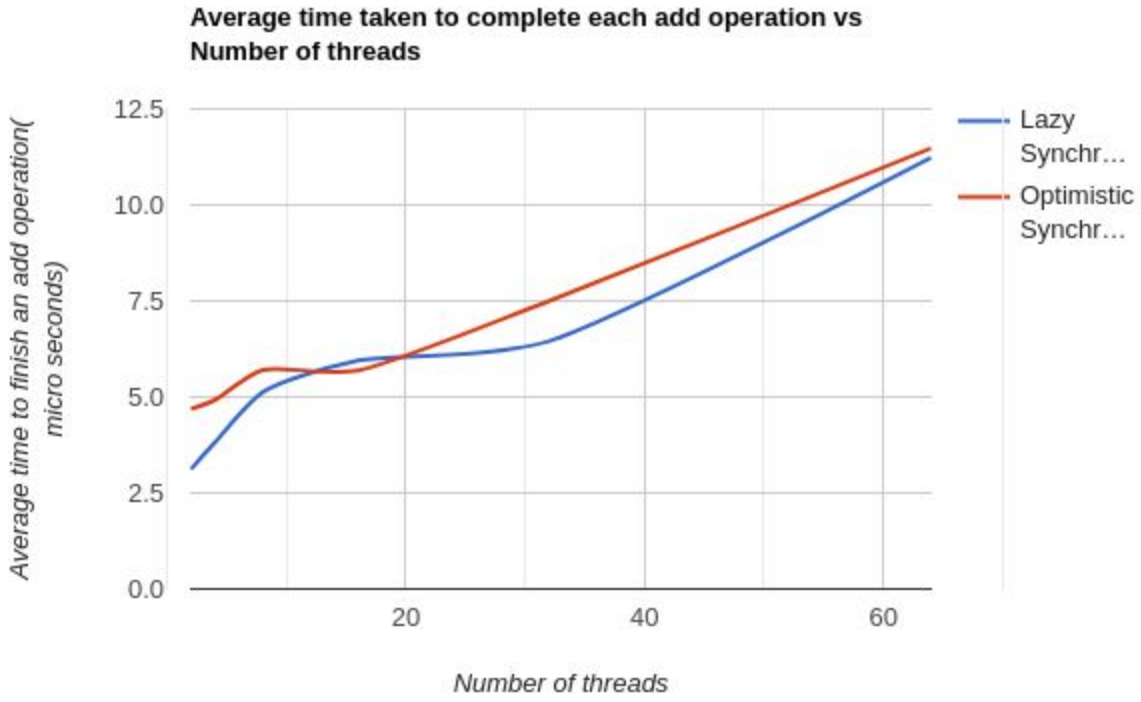
But in Optimistic Synchronization, searching is done without acquiring locks, lock the nodes found, and then confirm that the locked nodes are correct. If a synchronization conflict causes the wrong nodes to be locked, then release the locks and start over. But usually these conflicts are very rare. The OptimisticList algorithm is not starvation-free, even if all node locks are individually starvation-free.

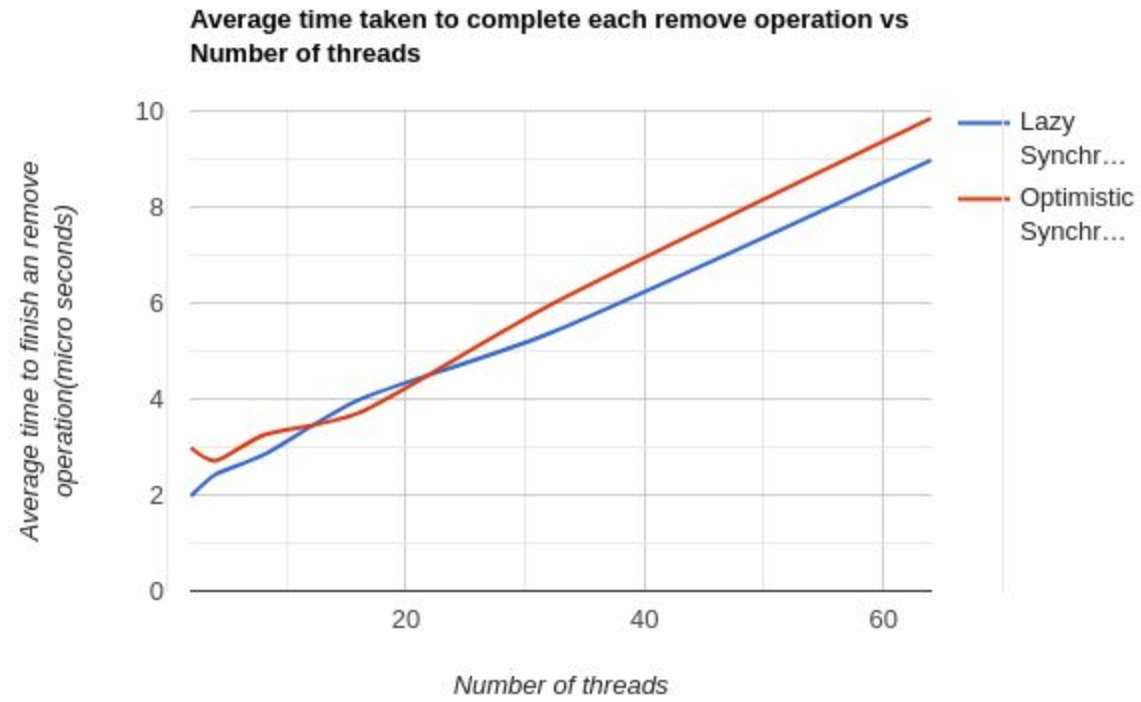
Lazy Synchronization

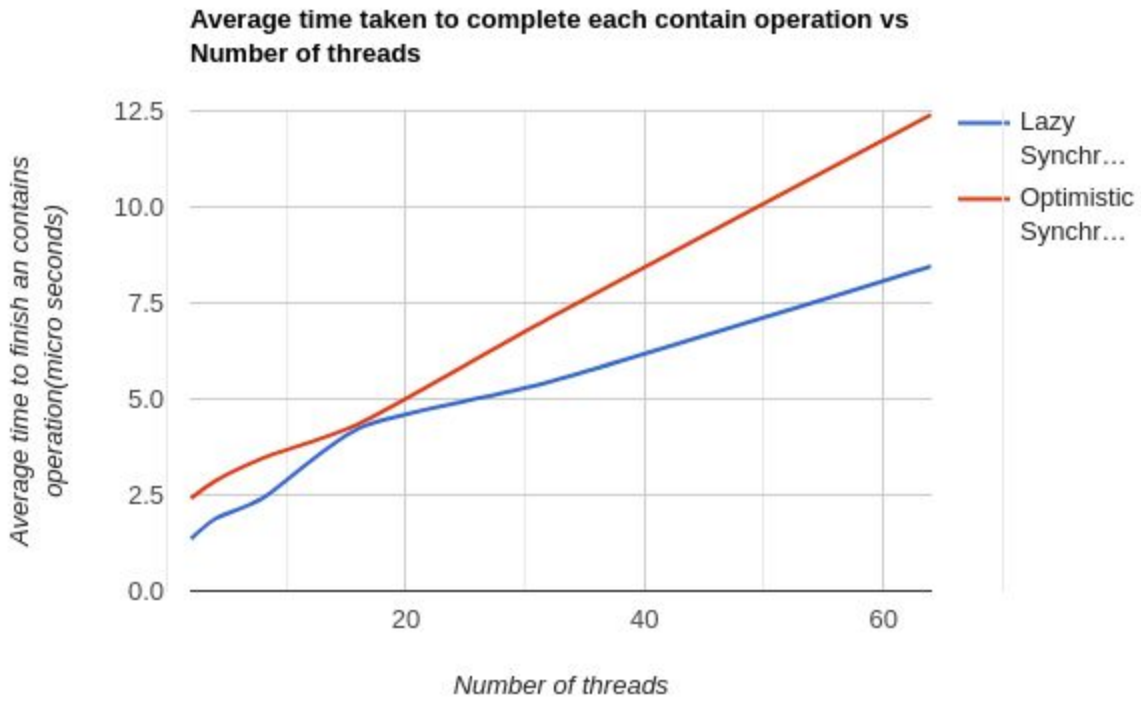
Optimistic Synchronization works best if the cost of traversing the list twice without locking is significantly less than the cost of traversing the list once with locking. One drawback of this particular algorithm is that contains() acquires locks, which is unattractive since contains() calls are likely to be much more common than calls to other methods.

In lazy synchronization, contains() method is made wait free, but add() and remove() methods are still blocking. We add to each node a Boolean marked field indicating whether that node is in the set. Now, traversals do not need to lock the target node, and there is no need to validate that the node is reachable by retraversing the whole list. Instead, the algorithm maintains the invariant that every unmarked node is reachable. If a traversing thread does not find a node, or finds it marked, then that item is not in the set. As a result, contains() needs only one wait-free traversal. To add an element to the list, add() traverses the list, locks the target's predecessor, and inserts the node. The remove() method is lazy, taking two steps: first, mark the target node, logically removing it, and second, redirect its predecessor's next field, physically removing it.

Comparison :


Average time taken to complete each add operation vs Number of threads
Average time to finish an add operation(micro seconds)
12.5
10.0
7.5
5.0
2.5
0.0
20
40
60
Number of threads
Lazy Synchr…
Optimistic Synchr…


Average time taken to complete each remove operation vs Number of threads
Average time to finish an remove operation(micro seconds)
10
8
6
4
2
0
20
40
60
Number of threads
Lazy Synchr…
Optimistic Synchr…



The contains method was faster in lazy synchronization(because for wait-free) for almost all cases but it wasn't the case for the add and remove methods. I have counted number of times each of add,remove,contains operations using add_ops, remove_ops, contains_ops and respective times as well and took average for graph.