Parallel and Distributed Computing
DD2443 - Pardis24
Exercises for Lecture 6

Name: Casper Kristiansson

September 12, 2024

Exercise 1

Question

Explain why the fine-grained list's add() method is linearizable.

Answer

The fine-grained list add() method is linearizable due to the reason that it ensures that each operation it påerformis is atomic. It makes sure that at each step in the method, the list is maintained in the correct structure all the time.

It does this by making sure that the specific different nodes in the operation require locks one by one and making sure that no other thread will be modifying the current node or any other action that might not make the method atomic. The important takeaway is that the locks are acquired in a way that prevents both a deadlock but also the correct locks by locking the correct nodes in the correct order.

Exercise 2

Question

For each of the following modifications of the sorted linked list algorithms, explain why the respective algorithm is still linearizable, or give a counterexample showing it is not.

a. In the optimistic algorithm, the contains() method locks two nodes before deciding whether a key is present. Suppose, instead, it locks no nodes, returning true if it observes the value, and false otherwise.

b. In the lazy algorithm, the contains() method executes without inspecting the locks, but it inspects the mark bit; it returns false if a node is marked for removal. Suppose, instead, the contains() does not inspect the mark bit of the nodes, and returns true even for nodes that may be marked.

Answer

Optimistic Algorithm

In the optimistic algorithm if the contains() method is modified so that the locks don't lock any nodes it returns true if the observes the value and false otherwise. Doing this type of implementation will lead to it not being linearizable. This happens because if another thread is inserting/removing a new node while the contains() method is acting it would lead to the result being inconsistent. For example, if the contains() method reads a specific node directly after it is removed. Because it doesn't lock it, it would lead to still returning True while in that situation it was False.

Lazy Algorithm

For the lazy algorithm where the contains() is modified so that it doesn't inspect the mark bit of the nodes and returns true even for nodes that may be marked. Doing this will lead to the algorithm not being linearizable anymore. The logic that we might mark nodes for deletion means that logically we can see those specific nodes as are removed. Return true for a marked node would mean that the contains method might return true for a node that no longer logically is a part of the list. This can create inconsistency in the list, and therefore the operation will not be able to maintain consistency in the list.

Exercise 3

Question

The add() method of the lock-free algorithm never finds a marked node with the same key. Can one modify the algorithm so that it will simply insert its new added object into the existing marked node with the same key if such a node exists in the list, thus saving the need to insert a new node?

Answer

Modifying the lock-free algorithm so that it will simply insert its newly added object into the existing marked node with the same key is not possible due to it compromising the principles of the lock-free data structure. A marked node should be seen as logically removed from the list meaning inserting it into would create inconsistency in it. When using multiple threads on a list a marked node could be in the process of being fully removed which could lead to a race condition happening if a new object would be inserted. The lock-free algorithm relies on the actions that are atomic and the removal and insertion of nodes and consistent.