CS5300 Theory Assignment 4

Gautam Singh
CS21BTECH11018

1) Consider the cases of the following method calls overlapping with the add method.

   a) add: If the other thread is inserting a new element at another position, then the threads will lock different pred nodes and there will not be any issues in validations and insertions. If both threads are inserting at the same position, then one of them will acquire the lock on pred and insert their new element. The other thread will acquire the lock but fail validation, following which it will attempt to validate and attempt to insert its new element again from the start of the while loop in line 3.

   b) remove: Suppose the curr of the add method is being removed. Then, the threads will have to lock the same pred node and hence the operations will proceed sequentially, with the latter operation failing validation. If the pred of the add call is being removed, then a similar argument follows since this pred node must be locked by both calls.

   c) contains: This will not cause any issues and is linearizable with the concurrent add, even if the element being searched for is concurrently being added. In this case, both calls must lock the same pred node, making the calls linearizable.

   Thus, in all cases, we need to only lock the pred node during insertion.

2) Setting the next field of a node to be deleted to null would effectively de-link the list, destroying information about the next nodes that may not be deleted. Other concurrent calls can travel through marked nodes in the LazyList implementation, especially in the interval between logical deletion (by marking the node) and physical deletion (actually swinging the next pointer). Hence, this change will break the lazy implementation.

   This change will not work even in case of the LockFreeList since the find method performs the physical deletions of marked nodes, and hence it travels through logically deleted nodes as well.

3) a) The compareAndSetMark method does not check if curr.next is equal to succ unlike the original algorithm. This makes the list implementation incorrect. Between lines 54 and the (modified) line 55, a concurrent addition might have taken place changing the successor node of curr, thus pred.next will be incorrectly set in line 58, skipping the new nodes inserted after curr.

   b) The attemptMark method is atmoic and can spuriously fail. It checks if cur.next equals succ from line 54 and then only marks the node for deletion. This maintains the correctness of the list.

4) The minimum number of blocks to be used is $\left\lceil\frac{8000}{1024}\right\rceil = 8$. This means that there are $8 \times 1024 = 8192$ threads in the grid. The programmer should configure the kernel call to have 8 blocks and 1024 threads per block. In CUDA, this would be `kernel<<< ceil(8000/1024.0), 1024 >>> (args)` or `kernel<<< 8, 1024 >>> (args)`.