CMPT 431: Distributed Systems (Fall 2019)
Assignment 3 - Report

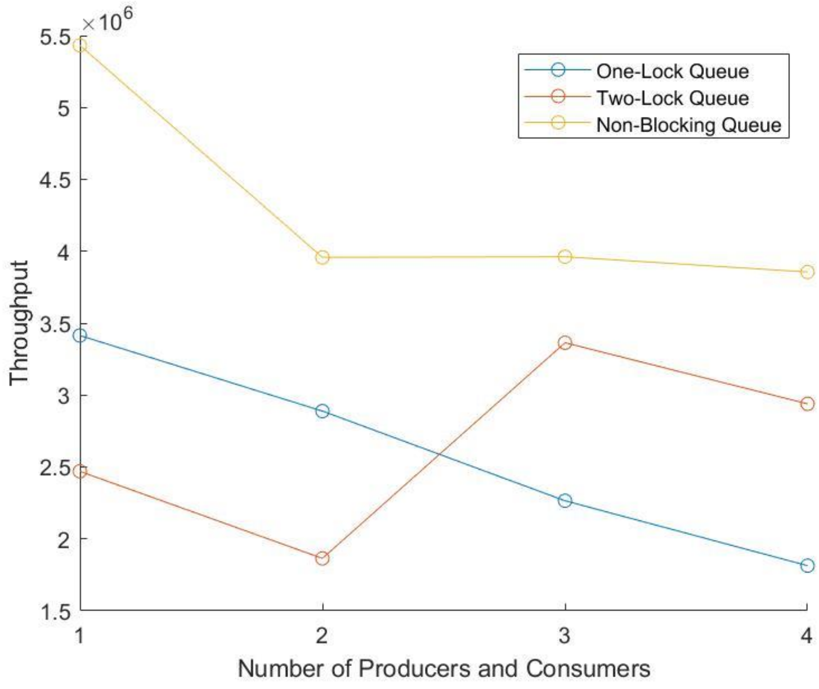
Name	Greyson Wang
SFU ID	301249759

Instructions:

- This report is worth 20 points.
- Answer in the space provided.
  Answers spanning beyond provided lines (11pt font) will lose points.

---

1. [5 Points] Plot the throughput per thread for OneLock Queue, TwoLock Queue and Non-Blocking queue. The structure of your plot should be similar to ones shown in class where x-axis has number of producers (P) and consumers (C) between 1 and 4 (i.e., 1P1C, 2P2C, 3P3C, 4P4C), and y-axis has throughput in terms of number of operations (“`Total throughput`” from output) per second. Your plot should have three lines (one for each queue).



2. [4 Points] Based on your plot from question 1, why does the throughput for each queue implementation vary as it does when producers and consumers increase? Why does the throughput vary as it does across different implementations?

   As the number of threads increase, the single lock in one-lock queue becomes a sequential bottleneck and throughput decreases. The two-lock queue has the same sequential bottleneck with its single enqueue and dequeue lock, but since enqueue and dequeue can happen at the same time, it has better performance. The non-blocking queue has the best performance since no locks are needed, but performance drops as threads need to help each other update tail before proceeding.

3. [7 Points] In Non-Blocking Queue pseudocode, there are two lines labeled as ELabel and DLabel. Why are those two lines present in the algorithm? Will the correctness and/or progress guarantees change if they are removed? If yes, which ones and why? If no, why? Support your argument using formal terms taught in class (e.g., linearizable, sequentially consistent, lock-free, wait-free, etc.).

   The line at Elabel and Dlabel helps update tail to the last node if the tail has not yet been updated to the last node. If Dlabel and Elabel are removed, the correctness of dequeue and enqueue will not change, but dequeue and enqueue will no longer be non-blocking, because they must wait until the current enqueue has finished updating tail.

4. [4 Points] In Non-Blocking Queue pseudocode, what is the purpose of the counter that is co-located with the next pointer? What can happen if the counter is removed? Please provide an example to support your argument.

   The counter prevents the ABA problem when the queue reuses recycled nodes that are put in a local pool. We can have a situation where thread A tries to dequeue and node a is the sentinel, followed by node b. Before thread A calls CAS to update head to b, node b and the following node is dequeued, then eventually, node a is reused by an enqueue and becomes the sentinel node again. Thread A then wakes up and calls CAS, which will succeed (and set head to the deleted node b) if the counter is not present, since the counter will inform thread A that node a has been removed and reused.