# CMPT 431: Distributed Systems (Fall 2023)

## Assignment 3 - Report

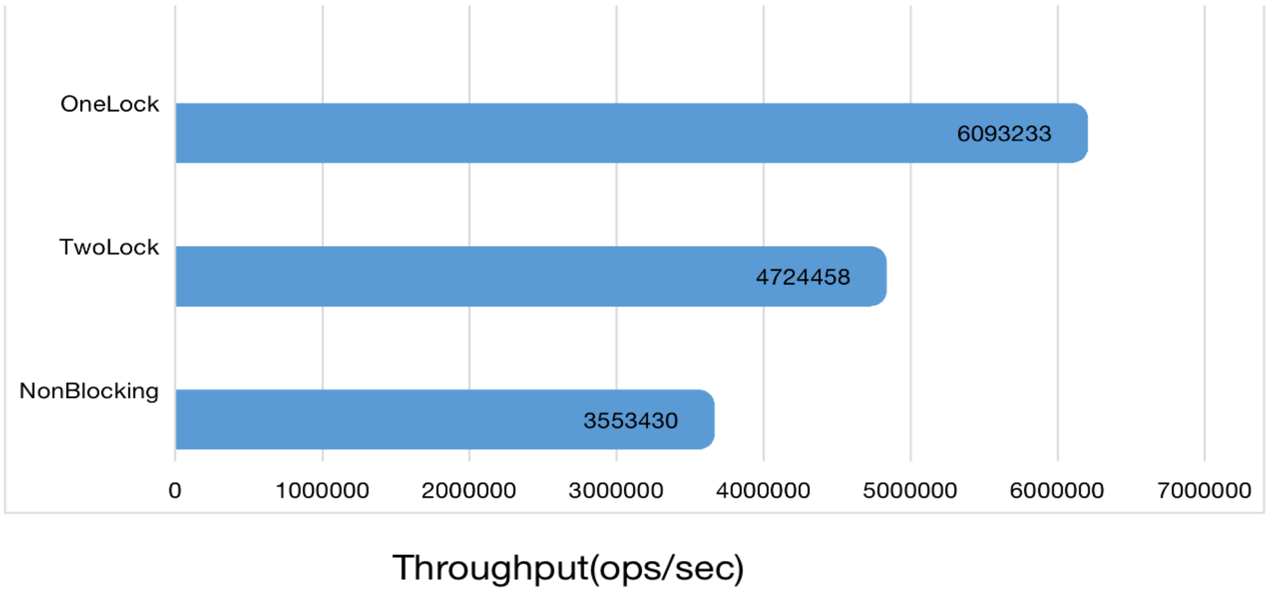
| **Name** | Lingjie Li |
|---|---|
| **SFU ID** | 301417109 |

**Instructions:**

- This report is worth 22 points.
- Answer in the space provided.
  Answers spanning beyond provided lines (11pt font) will lose points.
- All your answers must be based on the experiments conducted using the expected number of threads (as specified in the questions below) on fast nodes. Answers based on slow nodes and/or different numbers of workers than expected will result in 0 points.

---

1. [3 Points] Plot the total throughput for OneLock Queue, TwoLock Queue and Non-Blocking Queue with 4 producers and 4 consumers as a horizontal bar plot. The structure of your plot should be similar to the sample shown below: the x-axis has throughput in terms of number of operations ("**`Total throughput`**" from output) per second, and y-axis has the three queue implementations. The \`a', \`b', \`c', \`d', … on x-axis should be numbers (on linear scale).
   There may be some performance variation between runs, and hence you should take the average of 10 runs for each queue implementation. Remember to allocate one CPU per producer/consumer thread so that multiple threads do not have to fight for the same CPU (i.e., 4 producers + 4 consumers should be run using 8 CPUs).



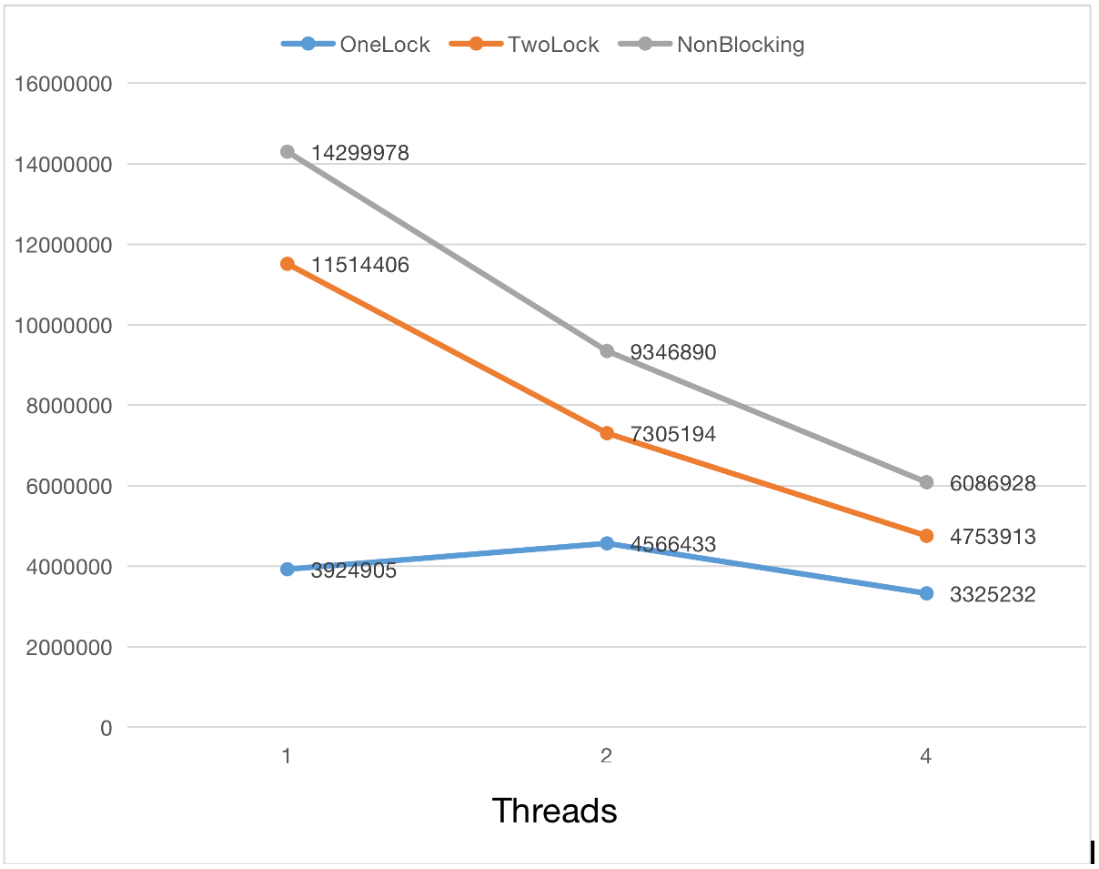
2. [2 Points] Based on your plot from question 1, why does the throughput vary as it does across different queue implementations?
A two-lock queue outperforms a one-lock queue in throughput. This is because enqueue and dequeue operations use separate locks, eliminating the need for the dequeue operation to wait for the lock released by enqueue. The absence of a lock in a non-locking queue significantly boosts throughput and concurrency.

3. [3 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.).
These lines guarantee the movement of the tail to the next node unless the tail is already pointing to the last node, known as forward tail. While this doesn't impact correctness, it does affect progress. The queue, being a lock-free implementation, allows the tail to reference either the actual last node or the penultimate node. When a thread successfully performs a CAS (Compare-and-Swap), it forwards the tail, ensuring continuity.

4. [2 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 ABA problem is mitigated when the pointer remains unique throughout a node's lifespan. However, eliminating the counter can lead to the resurgence of the ABA problem. For example, in a

situation where Thread 1 is retrieving a value and reads it for a CAS operation, Thread 2 might simultaneously use the same method to alter the initial value that Thread 1 had previously read. Later, Thread 2 restores the value to its original state. If Thread 1 was blocked or suspended during this process and then resumed execution, it may wrongly assume that the CAS operation was successful when, in reality, it was not.

5. [3 Points] Plot the total throughput for OneLock Queue, TwoLock Queue and Non-Blocking Queue with varying number of producers and consumers. The structure of your plot should be similar to the sample shown below: the x-axis has the number of producers/consumers and y-axis is the throughput in terms of number of operations ("**`Total throughput`**" from output) per second. Both axes are linear scales, and x-axis should show the following three points: 1, 2 and 4 (1 means 1 producer + 1 consumer, and similarly 4 means 4 producers + 4 consumers). There will be three lines, one for each type of queue, and each line will have three points.
   There may be some performance variation between runs, and hence you should take the average of 10 runs for each queue implementation. Remember to allocate one CPU per producer/consumer thread so that multiple threads do not have to fight for the same CPU (i.e., 4 producers + 4 consumers should be run using 8 CPUs).



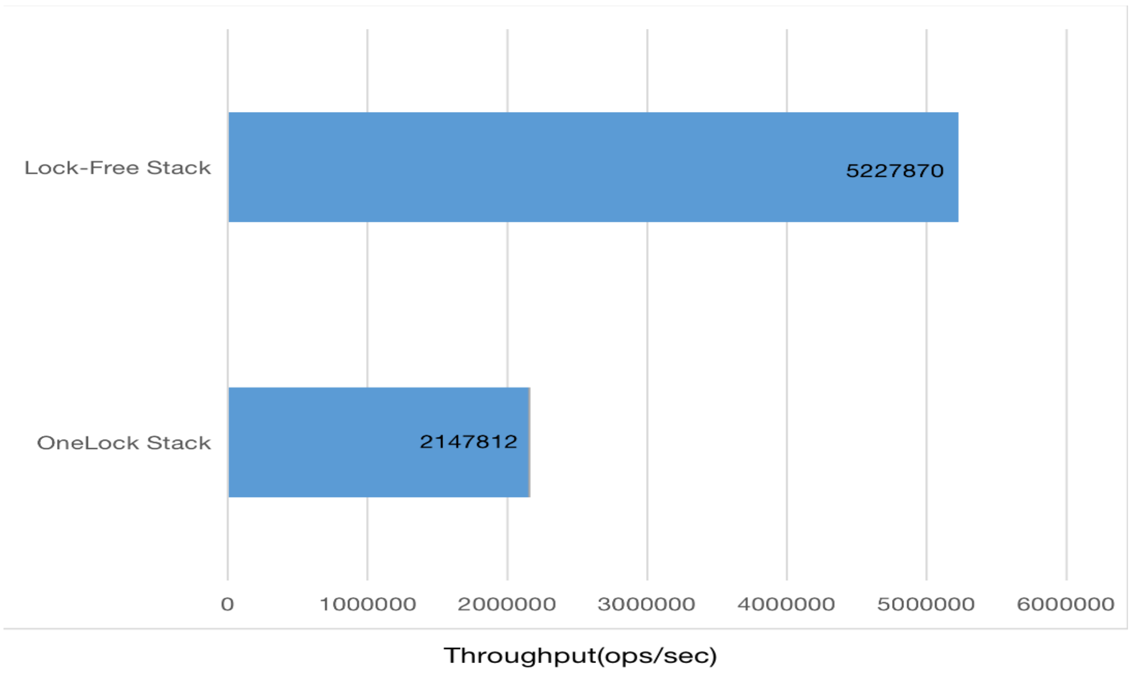
6. [3 Points] Based on your plot from question 5, why does the throughput vary as it does across different numbers of producers and consumers? If your scalability trends are different from the ones shown in slides, support your answer with appropriate measurements (e.g., latencies of operations, contention, etc.).

   Note: Adding extra timing/profiling code will affect performance. Ensure any code added to take these measurements is commented out or removed when you are measuring your final throughputs. Also ensure your measurement code is commented out in your code submission.

   As more threads are added, the synchronization overhead causes a decline in throughput. The trends observed in the plot from question 5 deviate from the optimistic scenario outlined in the slides. This deviation is attributed to contentions that arise when multiple threads interact with the queue. Even in a non-blocking queue implementation, the busy spin of the while loop necessitates numerous attempts to succeed, resulting in a reduction in throughput.

7. [2 Points] Plot the total throughput for OneLockStack and Lock-Free Stack with 4 producers and 4 consumers as a horizontal bar plot. The structure of your plot should be similar to the sample shown in question 1: the x-axis has throughput in terms of number of operations (“**`Total throughput`**” from output) per second, and y-axis has the two stack implementations. The \`a’, \`b’, \`c’, \`d’, … on x-axis should be numbers (on linear scale).

   There may be some performance variation between runs, and hence you should take the average of 10 runs for each stack implementation. Remember to allocate one CPU per producer/consumer thread so that multiple threads do not have to fight for the same CPU (i.e., 4 producers + 4 consumers should be run using 8 CPUs).



8. [1 Point] Based on your plot from question 7, why does the throughput vary as it does across different stack implementations?

   In a one-lock stack, where only a single lock is available, the time spent waiting to acquire the lock is considerably higher compared to a lock-free stack. This results in a notable decline in performance for a one-lock stack.

9. [3 Points] Can the ABA problem occur with Lock-Free Stack implementation? If yes, show the example with ABA problem. If no, why?

Yes, the ABA problem can occur with Lock-Free Stack. The ABA problem is less likely when the pointer remains unique for a node's entire existence. However, if the counter is removed, the ABA problem can reoccur. For instance, if Thread 1 reads a value for a CAS operation and Thread 2 simultaneously changes the original value, later restoring it, Thread 1, if blocked during this process, may incorrectly believe the CAS operation was successful upon resuming execution.