CMPT 431: Distributed Systems (Fall 2023)

Assignment 3 - Report

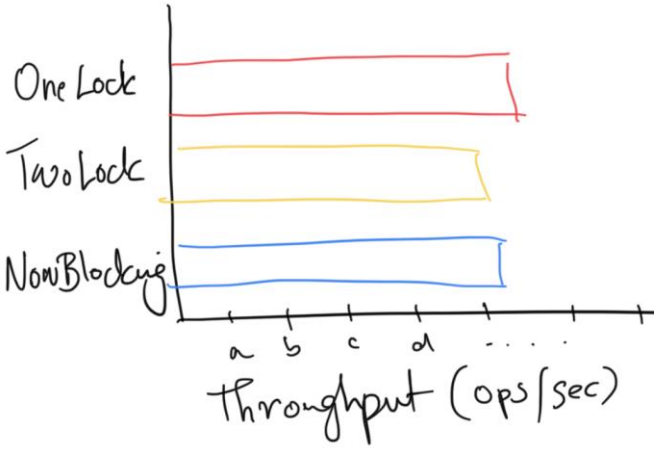
Name	MANAV MEGHPARA
SFU ID	301401593

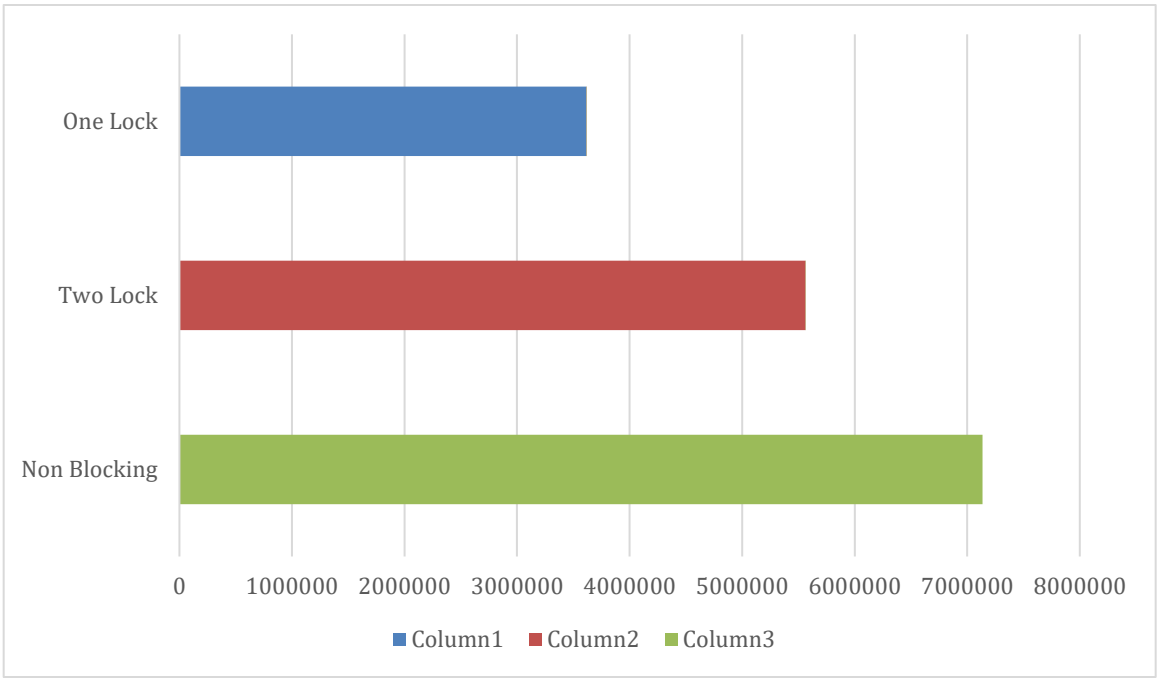
Instructions:

- 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?

   The two lock based approach has low throughput compared to non-blocking because of the elimination of lock, many threads can access the enqueue and dequeue functions. The two lock is better than one lock as it allows two threads to work concurrently for enq and deq while one-lock only allows either of those functions at a time which is basically sequential.

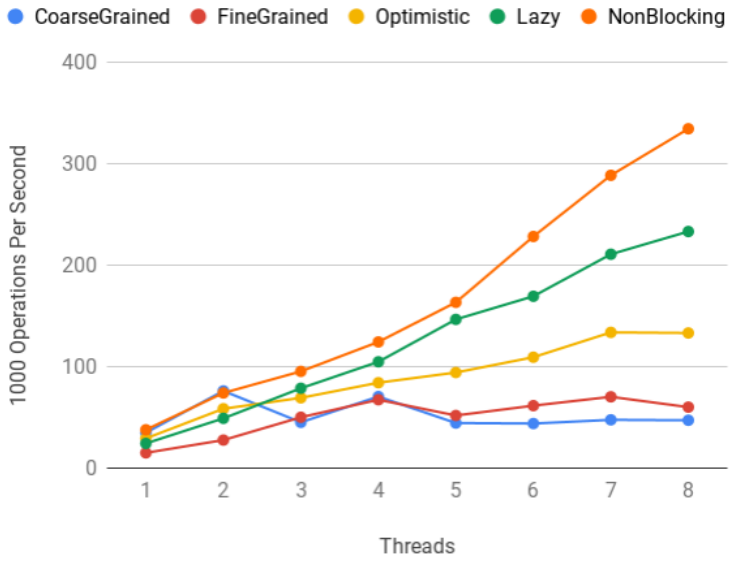
3. [3 Points] In Non-Blocking Queue pseudocode, there are two lines labeled as ELabel and DLabel. Why are those two lines present in the algorithm? Will the correctness and/or progress guarantees change if they are removed? If yes, which ones and why? If no, why? Support your argument using formal terms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

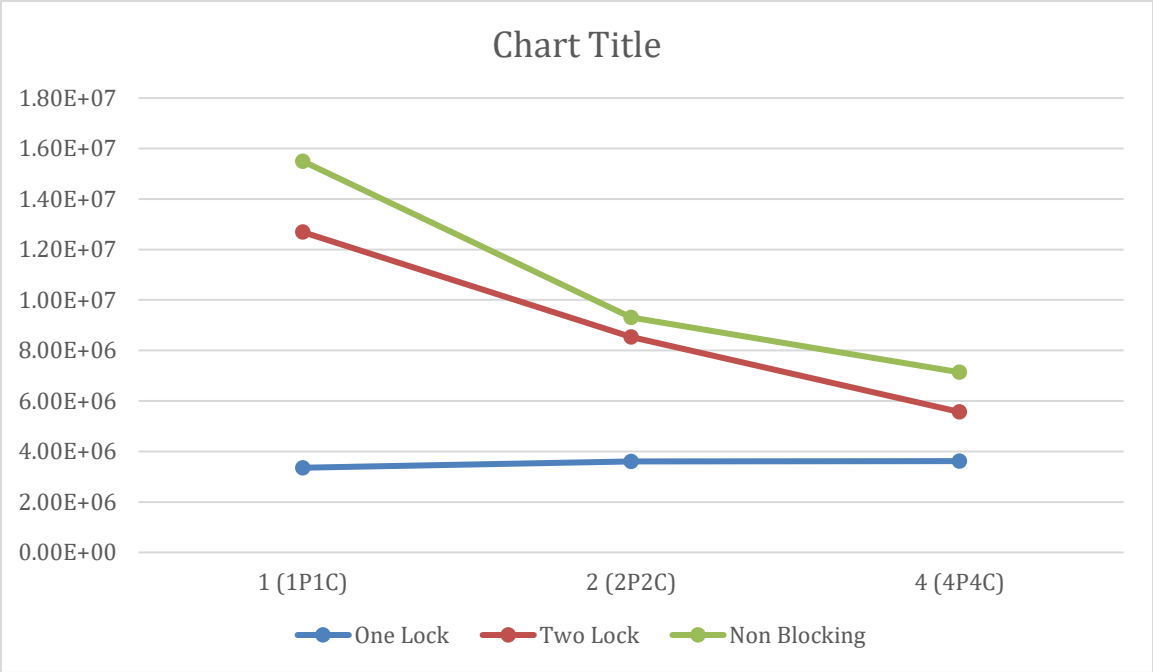
   Both the labels help forwarding the tail to aid with concurrent addition/ removal. If we remove Elabel, then most of the enqueue will be blocked and will depend on dequeue to forward tail. Similarly, if we remove Dlabel, dequeue will be blocked and depend on enqueue. Either of this will not change correctness and progress gurantees. If we remove both labels, none of those will progress as the tail won't move, hence, cannot guarantee linearization, correctness and progress.

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.

Counter helps resolve the ABA problem. If counter is removed, head can point to a freed memory location. Example – head->A->B->C, here thread 1 removes A holding first A and next B values and sleeps before performing CAS. Now, thread 2 removes A, B and enqueue A back, then deq C, then Q= head->A. thread 1 wakes and CAS success and now head points to a freed location which is B.

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).



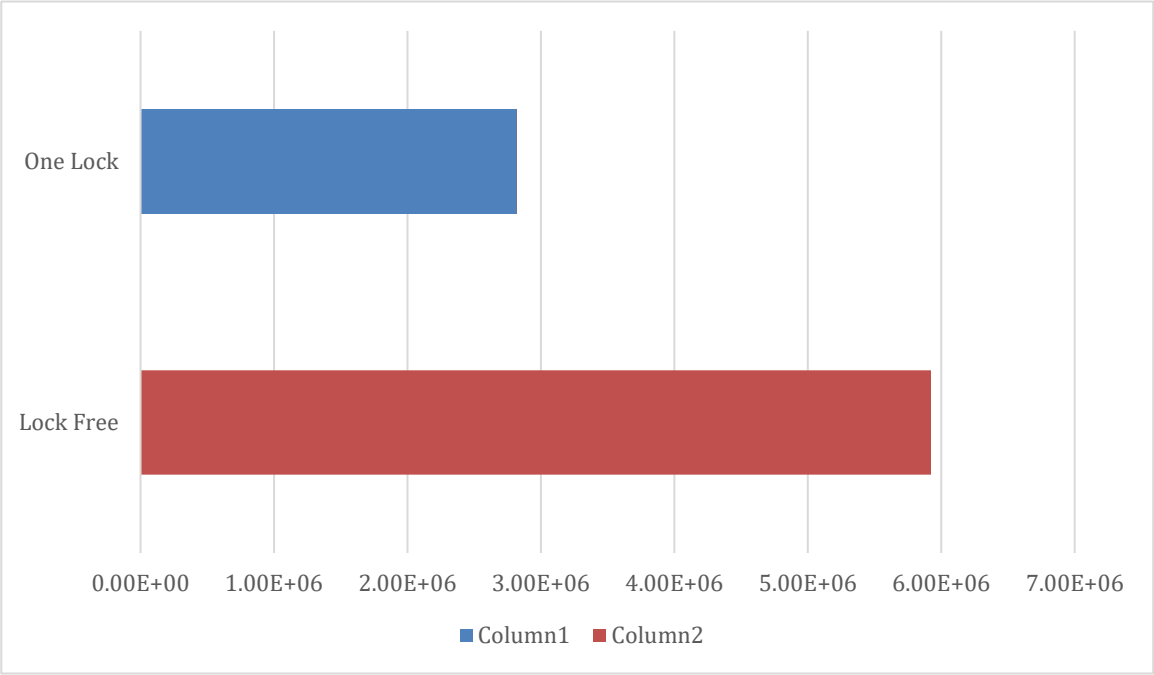


6. [3 Points] Based on your plot from question 5, why does the throughput vary as it does across different numbers of producers and consumers? If your scalability trends are different from the ones shown in slides, support your answer with appropriate measurements (e.g., latencies of operations, contention,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

One-lock remains same across all cases as it doesn't affect with increasing threads. Two-lock has significant throughput as it allows two concurrent operations, then it decreases with more threads due to more wait times. Non-blocking has the highest throughput among all. But it decreases with more threads because of increased contention on head and tail. This trend is different than slides as there was an extra wait free contains operation which resulted in high 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?

Lock Free stack has double throughput compared to one lock. The main reason is the elimination of lock. In one-lock, only one thread can do push/pop which is basically sequential. Whereas in lock free many threads perform concurrent push and pop resulting in high throughput.

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, ABA can occur in lock-free stack. One example would be, assume stack= top->A->B->C, thread 1 starts pop with old_top = A, new_top = B and sleeps before CAS. Now Thread 2 pops A and B, now stack = top->C. Thread 2 now pushes A again which makes stack = top->A->C. Now, thread 1 wakes up and try CAS which is successful as old_top is A again making B a new top which is a freed memory.