Assignment 2 Report - CS5300
Comparing Different Queue Implementations

Thupten Dukpa - ES20BTECH11029

CLQueue

The class variables for this implementation are head and tail of integer datatype, an items vector of T datatype, where T is a user-defined type, and a lock from the mutex library.

The constructor takes in a parameter capacity, initializes head and tail to 0 and allocates space for 'capacity' number of elements for items vector.

In enq(), firstly, a lock is acquired to ensure mutual exclusion. It checks whether the queue is full by comparing the difference between tail and head to the size of the items. If the condition is true, it throws a "Full Exception". If not, then there is space in the queue and it adds the element x at the tail index (taking care of wrap-around) and increases the value of tail by 1. The lock is then released.

In deq(), a lock is acquired to ensure mutual exclusion. It checks whether the queue is empty by comparing tail and head. If they are equal, it throws an "Empty Exception". If the queue is not empty, it retrieves the element at the head index (again, taking care of wrap-around) and increments the head. Finally, the lock is released, and the removed element x is returned.

If exceptions are thrown, they are caught but not handled explicitly so the program can continue on.

NLQueue

The class variables for this implementation are a vector of atomic optionals of T datatype called items, an atomic integer tail and a constant integer CAPACITY which has been initialized to INT_MAX by default.

The enq() method, uses atomic fetch_add on tail to obtain an index at which to insert the element and also increments tail by 1. It stores the element x which is taken as an argument at the i^{th} position in the items vector using the store() method.

The deq() method operates in a loop and checks each element in the items vector for values that are non-empty (i.e., not equal to nullopt). It uses the exchange() method to atomically retrieve and clear the value in the items vector, ensuring that only one thread succeeds in obtaining each element. If a non-empty element is found, it is returned. There is a possibility that this loop will run forever when the number of dequeue operations is more than the number of enqueue operations.

QTester Implementation

The algorithms have been tested in the CLQ1-ES20BTECH11029.cpp and NLQ1-ES20BTECH11029.cpp files itself.

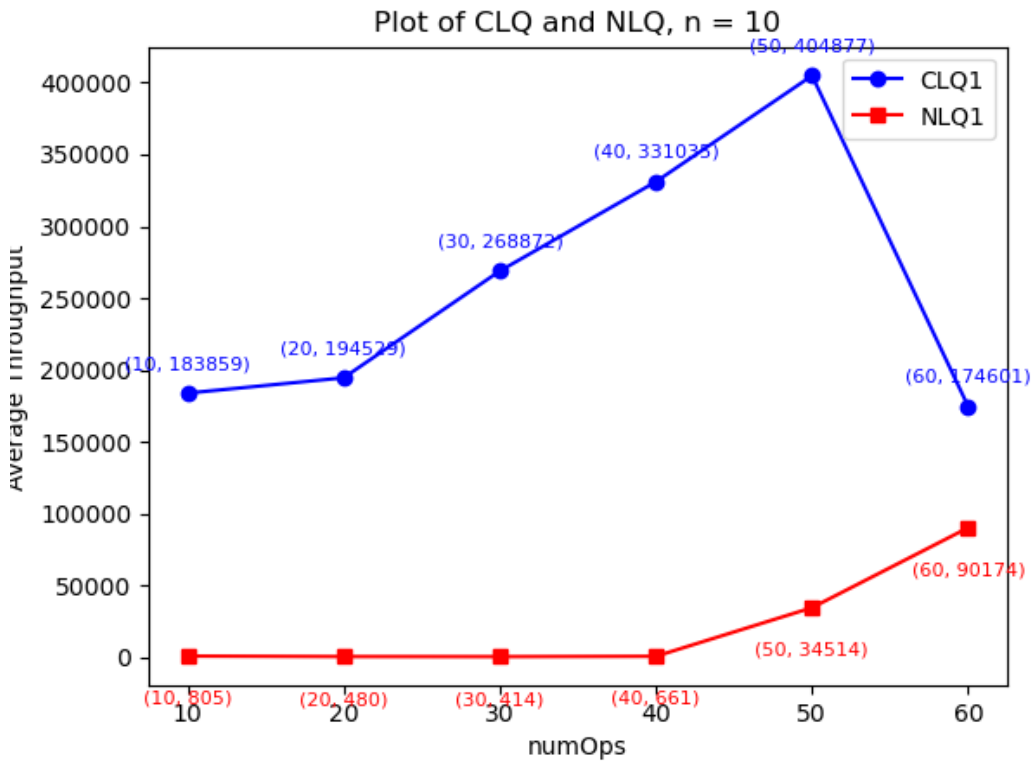
The input variables, the array to store time taken by each thread, the time taken for enqueue and dequeue operations and number of each of these operations are stored as global variables. In the main() function, the inputs are read and an object of class CLQueue or NLQueue are created. Also, n number of threads and the array thrTimes are initialized. A for loop creates n threads each of which executes the testThread() function. It waits for all threads to finish before joining them and finally calls the computeStats() function.

The testThread() method takes the object reference and thread_number as arguments. It uses the random library to generate two types of values from the uniform distribution from [0, 1] and exponential distribution of parameter λ. A for loop iterates for numOps number of times performing either enqueue or dequeue operation randomly and measures the time taken using the chrono library. It also makes the thread sleep for an exponential amount of time(in millisec).

The computeStats() method calculates the average time taken for enqueue and dequeue and also the average time and outputs them in a text file.

The graphs obtained after testing are displayed in the next pages.

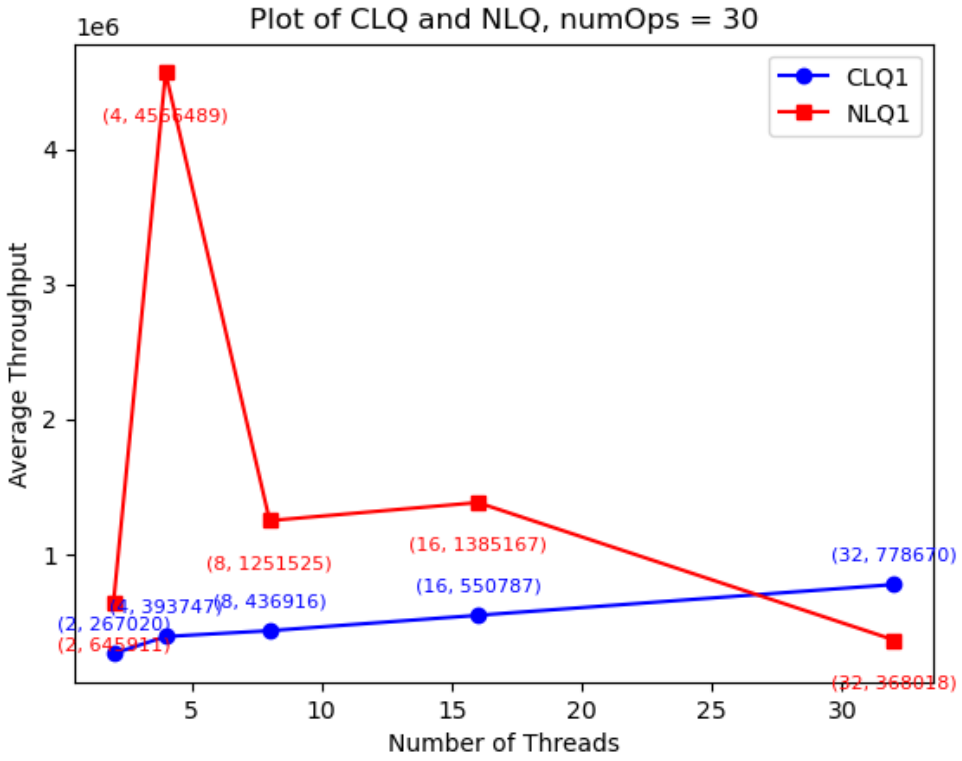
Throughput(in operations per second) vs numOps, $n = 16$, $rndLt = 0.5$, $\lambda = 5$



numOps	throughput_CLQ	throughput_NLQ
10	183859	805
20	194529	480
30	268872	414
40	331035	661
50	404877	34514
60	174601	90174

The Throughput was calculated as the reciprocal of the average time taken which is given as output and was averaged over 5 runs each time.

Throughput(in operations per second) vs n, $numOps = 30$**,** $rndLt = 0.5$**,** $\lambda = 5$



numThreads	throughput_CLQ	throughput_NLQ
2	267020	645911
4	393747	4566489
8	436916	1251525
16	550787	1385167
32	778670	368018