Concurrent Queue Performance Analysis

Project Members: Srikanth Kannan
Giridara Varma Elumalai

Abstract

The purpose of the experimentation is to implement and compare operations like enque, deque and is-empty in a concurrent queue using the following three approaches

a. lock-based synchronization
b. lock-free synchronization

Methodology

The implementation is done in java. An interface IConcurrentQueue is defined with following four methods.

a. enque
b. deque
c. isEmpty

The interface is implemented by following classes

a. LockBasedQueue.java
b. LockFreeQueue.java

ApplicationThread.java will call above methods defined by IConcurrentQueue interface based on given probability distribution which is passed as parameter.

Performance Testing Setup

The main class to run the performance testing is ApplicationRunner.java expects four optional arguments. Following are the information about the arguments.

a. Argument 0 – Refers to maximum number of threads that is used in the application. By default maximum number of threads is set as 8. For instance if this value is set as 8, then application will run for given probability distribution (argument 2) for 2 threads, then do the same for 4 threads, followed by 6 and 8 threads.
b. Argument 1 – Refers to maximum number of operations. If value is not passed, then by default 1000000 is used.
c. Argument 2 – Refers to relative distribution of various operations. It is given as comma separated values. First value in comma separated refers to enque, followed by deque and then is-empty operation. For example, if the intention is to test enque-Dominated work load, then input can be given as 80,15,5. If the intention is to test mixed domination work load, then input can be given as 50,50,0.
d. Argument 3 – Refers to initial capacity of the queue.

Our experimentation is done for maximum number of threads as 32(argument 0). Application will run both the locking approaches with number of threads as 2, then 4, followed by 6,8,10,12,16,18,20,22,24, 26,28,20 and 32. Maximum number of operations (argument 1) is set as 1000000 in all runs. Since there was a lot of fluctuation in the run times each experiment was run for three times and their average is plotted.

Performance testing is done four times each on the following configuration respectively

a. 50% Enque-50% Deque-0% isEmpty (Initially Empty)
b. 40% Enque-40% Deque-20% isEmpty (Initially Empty)
c. 50% Enque-50% Deque-0% isEmpty (Initial Capacity 1000)

d. 40% Enque-40% Deque-20% isEmpty (Initial Capacity: 1000)

Below is the information about the machine in which experiment was performed

RAM Size:	32 GB
Number of Cores:	16
Operating System:	Cent OS
Programming Language:	Java

Correctness Testing Strategy

I. Manual Testing

ApplicationThread.java is added with static set of operations and it was run for different work load configurations with 2 to 8 threads over key space of ten. The output of every operation is logged in log file and validity of outputs are checked manually.

Sample set of static operations

```
getConcurrentQueue().isEmpty();
getConcurrentQueue().enque(10);
getConcurrentQueue().enque(20);
getConcurrentQueue().enque(30);
getConcurrentQueue().enque(40);
getConcurrentQueue().enque(50);
getConcurrentQueue().deque();
getConcurrentQueue().deque();
getConcurrentQueue().isEmpty();
```

Sample Output:

Printing Queue:
True
30 40 50
False

II. Automated Testing

The correctness of the concurrent queue is verified by enqueing items with multiple threads and dequeing using single thread and dequed items are analyzed for correctness. In case of testing, the data structure holds object of type Key (which has both key and thread id of the thread which enqued the key) while in case of performance analysis, the data structure holds only key. Following are the classes implemented for testing strategy.

a. TestRunner.java
b. TestThread.java

While dequeing, an array of counter of size equal to number of threads involved in enqueing is created. On each deque operation, the respective element in counter array is incremented based on the thread id of dequed object. If the counter value is not equal to key of the dequed object, testing fails.

Automated Testing Setup

The main class to run the automated testing is TestRunner.java which expects three optional arguments. Following are the information about the arguments.

a. Argument 0 - Refers to maximum number of threads that is used in the application. If not value is passed, 8 is considered as default number of threads.
b. Argument 1 – Refers to maximum number of operations. If value is not passed, then by default 1000000 is used. This value determines number of items to be enqued by each thread.

Sample Output:

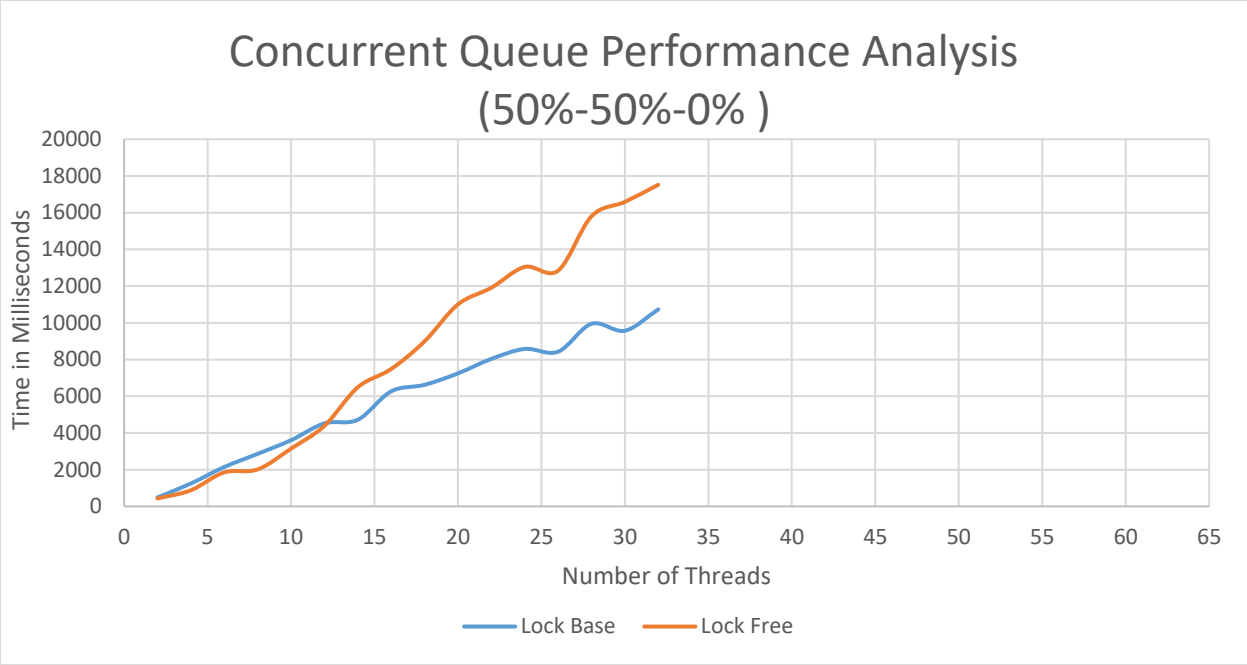
```
ThreadCount: 2, LockBasedQueue - true
ThreadCount: 2, LockFreeQueue - true
ThreadCount: 4, LockBasedQueue - true
ThreadCount: 4, LockFreeQueue - true
ThreadCount: 6, LockBasedQueue - true
ThreadCount: 6, LockFreeQueue - true
ThreadCount: 8, LockBasedQueue - true
ThreadCount: 8, LockFreeQueue - true
```

Test Result

For all the below tests, the number of operations is fixed as 1000000.

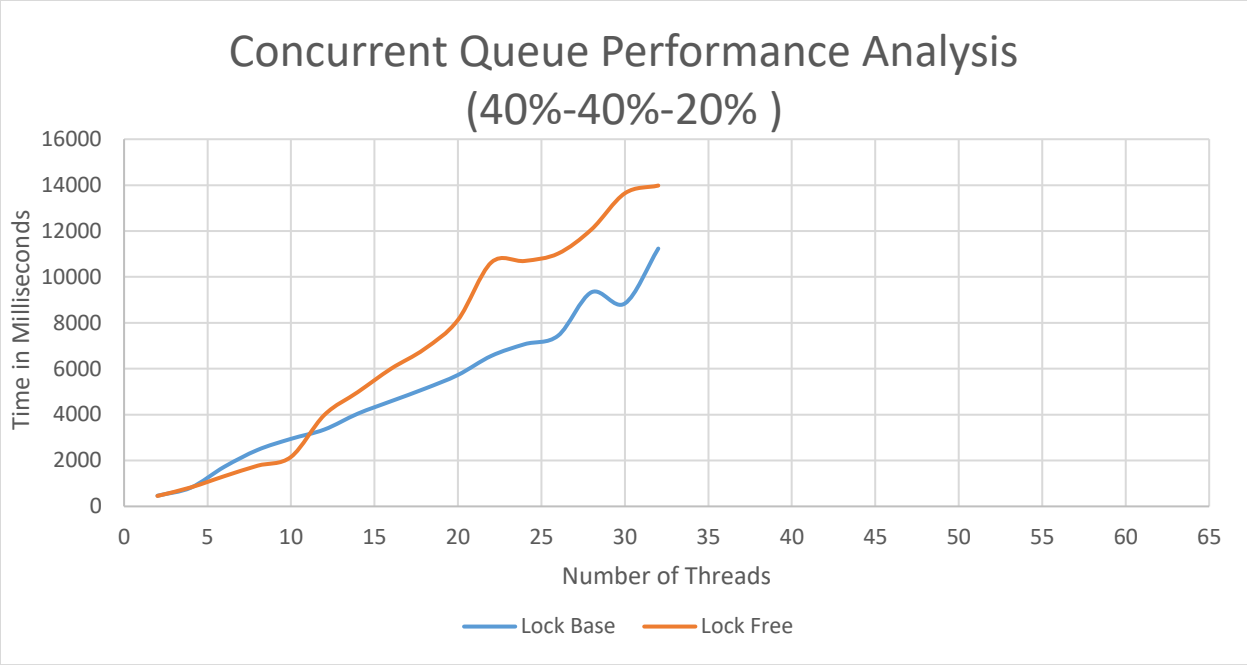
I. 50% Enque-50% Deque-0% isEmpty (Initially Empty):

Number of Threads	Lock Based(ms)	Lock Free(ms)
2	488	432
4	1246	879
6	2145	1852
8	2861	2013
10	3605	3145
12	4523	4387
14	4720	6487
16	6266	7484
18	6622	8991
20	7246	11002
22	8034	11911
24	8578	13045
26	8426	12839
28	9949	15803
30	9570	16587
32	10731	17519



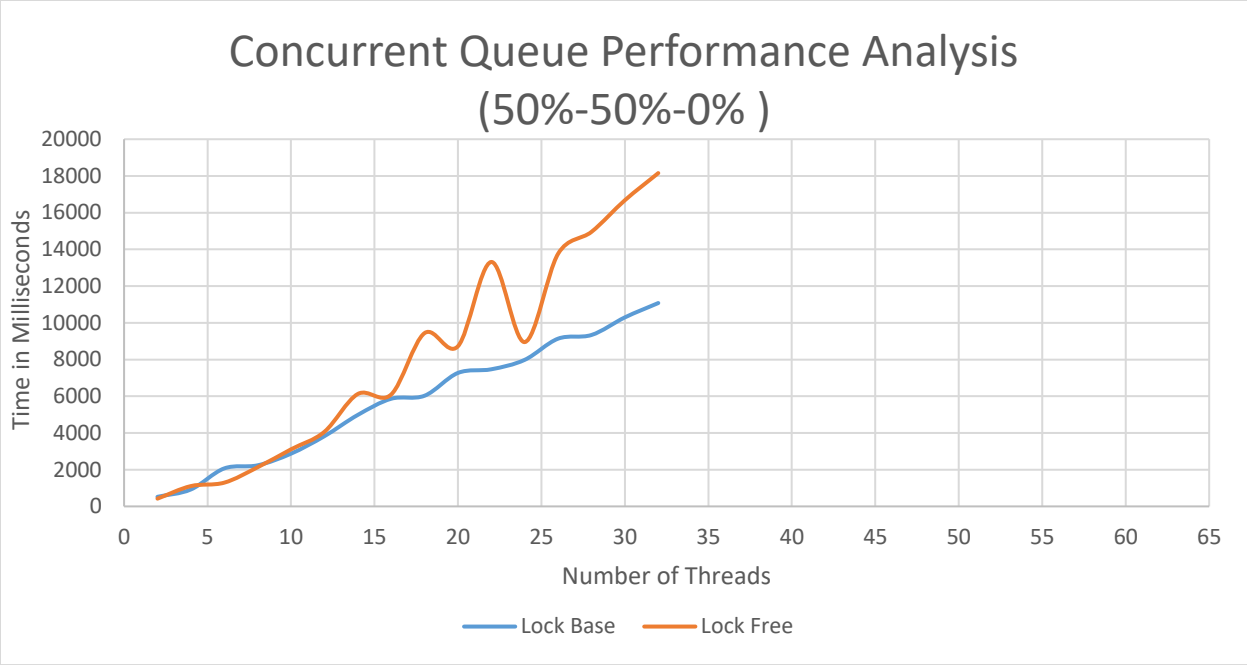
II. 40% Enque-40% Deque-20% isEmpty (Initially Empty):

Number of Threads	Lock Based(ms)	Lock Free(ms)
2	465	456
4	816	828
6	1724	1315
8	2461	1773
10	2944	2157
12	3350	3986
14	4041	4980
16	4585	6003
18	5126	6862
20	5728	8125
22	6559	10632
24	7069	10696
26	7440	11017
28	9332	12070
30	8843	13645
32	11236	13981



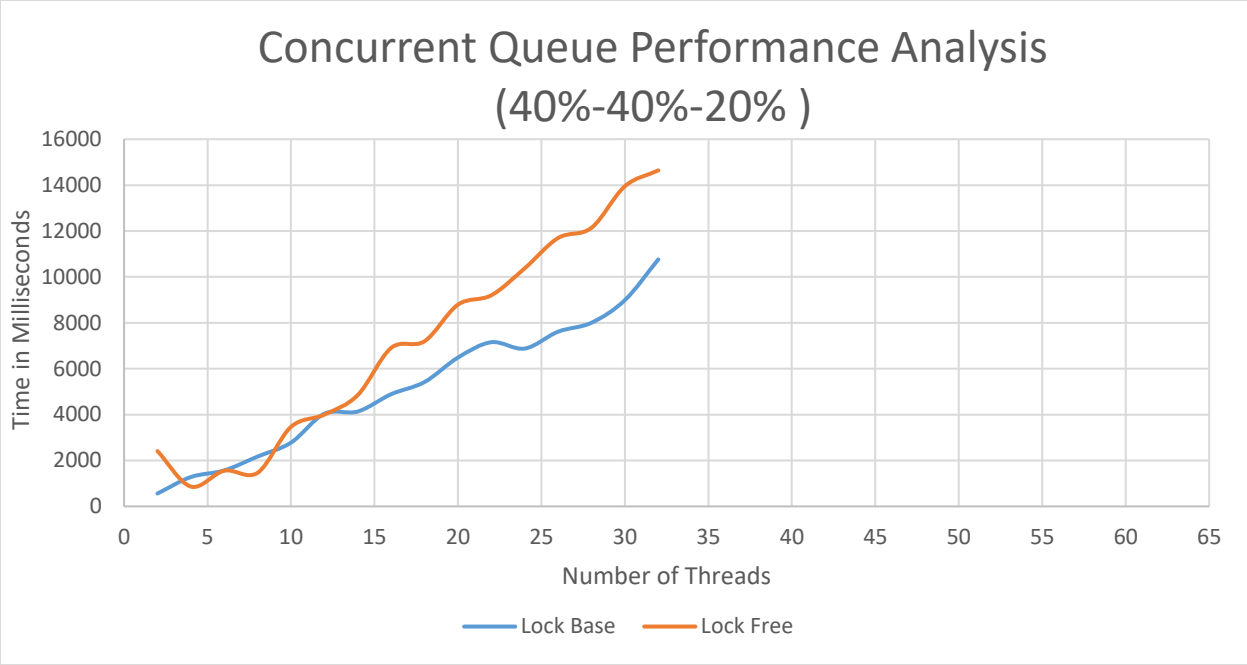
III. 50% Enque-50% Deque-0% isEmpty (Initial Capacity 1000):

Number of Threads	Lock Based(ms)	Lock Free(ms)
2	522	422
4	922	1106
6	2069	1293
8	2237	2132
10	2873	3109
12	3824	4063
14	4982	6122
16	5864	6095
18	6028	9439
20	7270	8723
22	7469	13315
24	7984	8951
26	9136	13762
28	9337	14955
30	10296	16677
32	11078	18156



IV. 40% Enque-40% Deque-20% isEmpty (Initial Capacity: 1000):

Number of Threads	Lock Based(ms)	Lock Free(ms)
2	561	2411
4	1278	864
6	1567	1557
8	2173	1464
10	2779	3466
12	4040	4000
14	4136	4853
16	4890	6907
18	5420	7197
20	6486	8792
22	7155	9200
24	6878	10376
26	7614	11696
28	8003	12140
30	8989	13951
32	10762	14642



Observation

From all the readings above, it is observed that when the thread count is less than 12, lock free algorithm performs better than lock based algorithm. But, as the thread count increases beyond thread count 12, lock based algorithm out performs lock free algorithm.

Conclusion

From the observation it can be inferred that lock based queue is outperforming lock free queue as the contention increases, contrary to the general expectation that lock free data structures outperform their lock based counterparts. This could be due to the following reasons,

1. In real world scenarios where there are other processes and high load on the system, with lock based queue if the thread which holds the lock gets interrupted by the OS and goes to sleep, then other threads in contention will all get delayed. But in our experimental setup with no extra load by external processes, chances of interruption on the thread holding the lock is less.
2. Another possible reason is that lock free queue executes a lot of hardware dependent atomic instructions (AtomicReference), which could cause extra overhead depending on how these instructions are implemented by the hardware, thereby affecting the performance of lock free queue.