DSA I Lab 10: Concurrency

Christine Baca [cab8xd]

[1] University of Virginia, Charlottesville VA 22904, USA

Abstract. The lab investigated implementation and runtimes of concurrent queues and threading. The experiment uses a linked list-based queue that utilizes locks and conditions to allow blocking and allows concurrent programming. By comparing runtimes, the lab tests the efficiency of concurrency between sequential and varying multithread runs of the queue's enqueue and dequeue methods. The lab concluded that running the queue using two threads proved the most efficient, and adding more thread or using a sequential queue within the main program proved slower likely due to the added steps or processes that the two-thread implementation condenses when utilizing the enqueue and dequeue methods.

Keywords: Concurrency, threads, blocking, Java, sequence, enqueue, dequeue

Concurrency allows program processes to run in parallel and improve the efficiency of larger program structures through the process or threading. Each thread represents a component process of overarching program. For concurrent queues, threading allows enqueue and dequeue to run in parallel. When the same variable is used by multiple threads, the variable becomes a shared resource and blocking and conditions must be added to the data structure to accommodate possible errors and deadlocks that may come from the overlapping use of variables between threads.

1.1 Predictions

The lab predicts that concurrent queues with two threads will have a faster runtime than a sequential queue and if the number of threads are greater than two, the computer will start to lag due to the number of processes running or that the threads will overwhelm and slow down the dequeue blocking process.

1.2 Test Code for Threads = 3 Concurrent Queue Trial

```
//Testing Code Sample:
                System.out.println("   THREADS = 3");
                System.out.println("   RUNTIME:");
                System.out.println("-----------------------------------");
                th3.start();
                th4.start();
                th5.start();
                try {
                        th3.join();
                        th4.join();
                        th5.join();
                } catch (Exception e) {
                        e.printStackTrace();
                }
                System.out.println(Math.max(Math.max(ct3.time, ct4.time),
ct5.time) + " ms");
```

2 Experiment:

The procedure for the experiment runs through numerous trials of their respective algorithms and data sets, recording the runtime of the enqueue and dequeue operations. The average and standard deviation of the trials will be calculated and recorded. The data will be analyzed based on the trial's runtimes, averages, number of threads implemented, and standard deviations. The experiment table lists the number of trials, runs, and additional information regarding the experimental procedure.

2.1 Experiment Table:

	Experiment
No. of Trials	10
Number of Elements enqueued and dequeued	1,000,000
Number of enqueue and dequeue runs	10,00,000 each

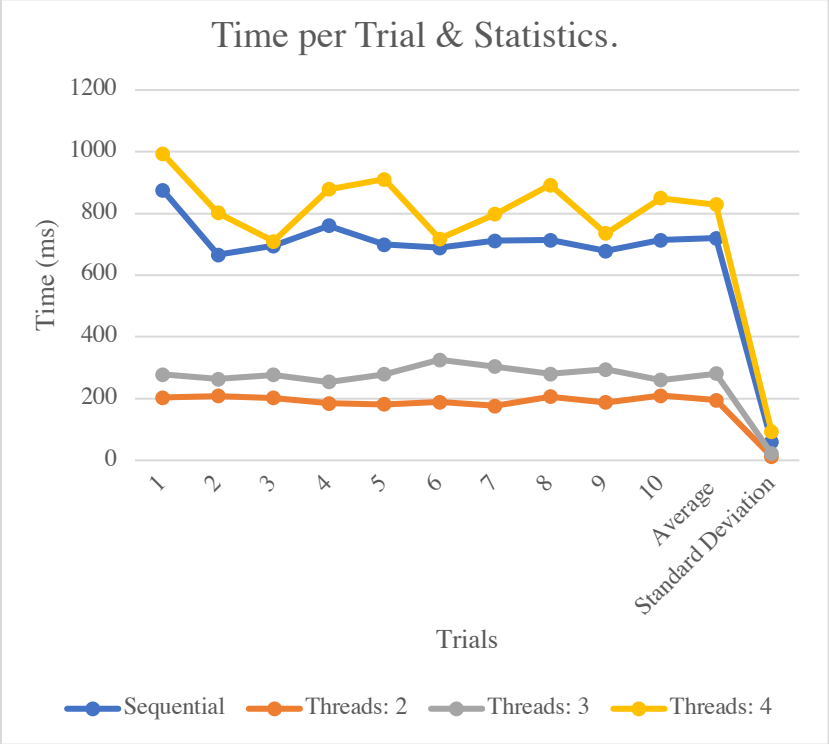
3 Results

3.1 Table Results: Trial per Runtime (milliseconds)

	Sequential	*Threads: 2*	*Threads: 3*	*Threads: 4*
1	876	203	278	994
2	666	208	263	802

3	695	202	277	710
4	760	185	254	879
5	699	182	279	911
6	689	189	326	718
7	712	176	304	798
8	714	206	280	892
9	678	188	295	735
10	714	210	260	850
Average	720.3	194.9	281.6	828.9
Standard Deviation	60.3	12.2	21.7	93.0

3.2 Graph Results



4 Conclusion

The experiment compared the efficiency of sequential and several multithreaded concurrent queues. Concurrent queues run the enqueue and dequeue processes in parallel while the sequential queue runs enqueue and dequeue in order. The data results confirm the prediction that the concurrent queue runs faster than the sequential queue most of the time. The trial using two threads held the lowest average runtime and standard deviation while the trial testing four threads held the highest average runtime and standard deviation. As the number of threads in the concurrent queue increased, the runtime became slower. Errors include possible mistakes in code that could skew the results. Further experiments could the test the reasoning for this; whether the slowing runtimes is due to the threshold that a certain number of threads and processes in parallel may overwhelm computer's processors and slow runtime or if the slower runtimes relate to the number of threads and the number of the methods implemented; as the queue only utilized two methods, using more threads than methods may have proved counterintuitive to the algorithm or data structure used.