Thesis Proposal

Hossein Naderi

February 4, 2022

1. Importance of implementing shared data structures
    - how it helps developers
    - why lock free data structures are better than using locks?
2. What properties matter for a distributed data structure?
    - Correctness condition: Linearizability definition
    - Wait-freeness
3. Previous works on distributed queues: more detailed
    - why we are studying queus
    - p factor in time complexity of all previous algorithms
    - types of current queues in the literature
    - previous impractical universal constructions that use long words
4. Our contribution
    - polylogarithmic algorithm with respect to p and Q size of the queue
5. Explaining ideas
    - using tournament tree to agree on the linearization order
    - double refresh
    - how to compute a dequeue response

- how to be sure the number of blocks in the root is related to the size of the queue
- memory management as a goal

Shared data structures have become an essential field in distributed algorithms research. We are reaching the physical limits of how many transistors we can place on a CPU core. The industry solution to provide more computational power is to increase the number of cores of the CPU. This is why distributed algorithms have become important. It is not hard to see why multiple processes cannot update sequential data structures; for example, consider two processes trying to insert some values to a linked list simultaneously. One solution is to use locks; whenever a process wants to query on a data structure, it locks it, and others cannot use it until the lock is released. Using locks has some disadvantages; for example, one process might be slow, and holding a lock for a long time prevents other processes from progressing. Moreover, locks do not allow parallelism since only one process can make progress in one moment. Our approach is to create a lock-free data structure that developers can safely use without the disadvantages of locks.

The question that may arise is, "What properties matter for a lock-free data structure?". Since executions on a shared data structure are different from sequential ones, the correctness conditions also differ. To prove a concurrent object works perfectly, we have to show it satisfies safety and progress conditions. A safety condition tells us the data structure does not return wrong responses, and a progress condition indicates that operations eventually terminate.

The standard safety condition is called linearizability, which ensures that for any concurrent execution on a linearizable object, each operation should appear to take effect instantaneously at some moment between its invocation and response. Figure 1 is an example of execution on a linearizable queue. The arrow shows time, and rectangles show the time between invocation and termination of an operation. Since ENQ(A) and ENQ(B) are concurrent ENQ(B) may have taken effect before ENQ(A). Execution in figure 2 is not consistent since A has been enqueued before B, so it has to be dequeued first.

An algorithm is *wait-free* if each operation terminates after a finite number of steps. We call an algorithm *lock-free* if, after a sufficient number of steps, one

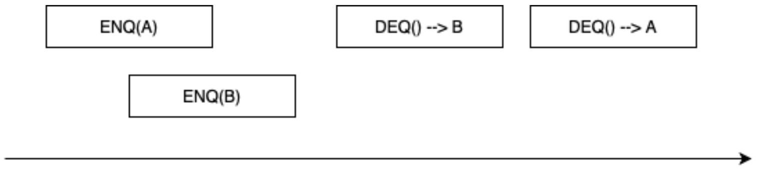

Figure 1: An example of a linearizable execution. Both could take effect first since ENQ(A), and ENQ(B) are concurrent.

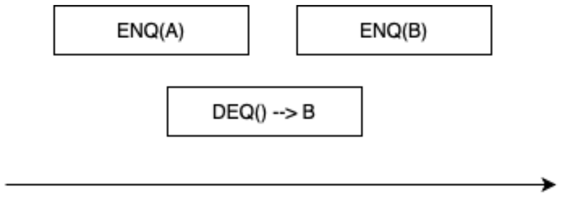

Figure 2: An example of an execution that is not linearizable. Since ENQ(A) has completed before ENQ(B) is invoked the DEQ() should return A.

operation terminates. A wait-free algorithm is also lock-free. From now on, we are going to focus on wait freedom.

Previous Work In the following paragraphs, we look at previous work on distributed queues. Michael & Scott [1] introduced a lock-free queue which we refer to as the MS-queue. It is included in the standard Java Concurrency Package. Their idea is to store the queue in a singly linked list to compute the head and tail of the queue from the head and tail of the linked list. To insert a node to the linked list, they use compare and swap operations. If p processes try to enqueue simultaneously, only one can succeed and the others have to retry. This makes the worst-case number of steps to be $\Omega(p)$ per enqueue. Similarly, dequeue can $\Omega(p)$ take steps.

Moir, Nussbaum, and Shalev [2] presented a more sophisticated queue by using the elimination idea. The elimination mechanism benefits the dual purpose of allowing operations to complete in parallel and reducing contention for the queue. An Elimination Queue consists of an MS-queue augmented with an elimination array. Elimination works by allowing opposing operations such as pushes and pops to exchange values. Their algorithm makes it possible for failed aged operations to complement. Empirical evaluation of their work is better than MS-queue, but the worst case is still the same, in case there are p concurrent enqueus their algorithm is not better than MS-queue.

Hoffman, Shalev, and Shavit [3] tried to make the MS-queue more parallel by introducing the Baskets Queue. Their idea is to allow more parallelism by treating the simultaneous enqueue operations as a basket. Since the operations in a basket are concurrent, we can order them in any way. Operations in a basket try to find their position in queue one by one by using CAS operations. The operations in different baskets can be executed in parallel. However, like the previous algorithm, if still there are p concurrent enqueue operations in a basket, the amortized complexity remains $\Omega(p)$.

Ladan-Mozes and Shavit [4] presented an Optimistic Approach to Lock-Free FIFO Queues. They use a doubly linked list and do fewer compare and swap operations than MS-queue. But as before, the worst case is when there are p concurrent enqueues which have to be enqueued one by one. The amortized worst-case complexity is still $\Omega(p)$ CASes.

Hendler et al. [5] proposed a new paradigm called flat combining. Their queue is linearizable but not lock-free. After adding an operation acquiring the lock, they maintain history in publication records and compute all active operations responses. They believe their algorithm in real-world assumptions works well.

Gidenstam, Sundell and Tsigas [6] introduced a new algorithm using a linked list of arrays. Global head and tail point to arrays and content of arrays are marked if dequeued and written by CAS operations. Their data structure is lock-free but it updates lazily. Threads have a cache view on the queue and update it if it gets old.

Kogan and Perank [7] introduced wait-free queues based on MS-queue and use Herlihy's helping technique to achieve wait-freeness. Their step complexity is $\Omega(p)$ because of the helping mechanism.

Milman et al. [8] designed BQ: A Lock-Free Queue with Batching. Their idea of batching allows a sequence of operations to be submitted as a batch for later execution. It supports a new notion introduced by the authors called Extended Medium Futures Linearizability.

Nikolaev and Ravindran [9] wCQ to be completed.

Our contribution In this work, we are trying to design a queue with $\log p^2 + Q$ steps per operation, outperforming queues built by LinkedList. Our idea is similar to the Jayanti MESD queue.

References

[1] M. M. Michael and M. L. Scott, "Simple, fast, and practical non-blocking and blocking concurrent queue algorithms," in *Proceedings of the Fifteenth Annual ACM Symposium on Principles of Distributed Computing, Philadelphia, Pennsylvania, USA, May 23-26, 1996* (J. E. Burns and Y. Moses, eds.), pp. 267–275, ACM, 1996.

[2] M. Moir, D. Nussbaum, O. Shalev, and N. Shavit, "Using elimination to implement scalable and lock-free FIFO queues," in *SPAA 2005: Proceedings of the 17th Annual ACM Symposium on Parallelism in Algorithms and Architectures, July 18-20, 2005, Las Vegas, Nevada, USA* (P. B. Gibbons and P. G. Spirakis, eds.), pp. 253–262, ACM, 2005.

[3] M. Hoffman, O. Shalev, and N. Shavit, "The baskets queue," in *Principles of Distributed Systems, 11th International Conference, OPODIS 2007, Guadeloupe, French West Indies, December 17-20, 2007. Proceedings* (E. Tovar, P. Tsigas, and H. Fouchal, eds.), vol. 4878 of *Lecture Notes in Computer Science*, pp. 401–414, Springer, 2007.

[4] E. Ladan-Mozes and N. Shavit, "An optimistic approach to lock-free FIFO queues," *Distributed Comput.*, vol. 20, no. 5, pp. 323–341, 2008.

[5] D. Hendler, I. Incze, N. Shavit, and M. Tzafrir, "Flat combining and the synchronization-parallelism tradeoff," in *SPAA 2010: Proceedings of the 22nd Annual ACM Symposium on Parallelism in Algorithms and Architectures, Thira, Santorini, Greece, June 13-15, 2010* (F. M. auf der Heide and C. A. Phillips, eds.), pp. 355–364, ACM, 2010.

[6] A. Gidenstam, H. Sundell, and P. Tsigas, "Cache-aware lock-free queues for multiple producers/consumers and weak memory consistency," in *Principles of Distributed Systems - 14th International Conference, OPODIS 2010, Tozeur, Tunisia, December 14-17, 2010. Proceedings* (C. Lu, T. Masuzawa, and M. Mosbah, eds.), vol. 6490 of *Lecture Notes in Computer Science*, pp. 302–317, Springer, 2010.

[7] A. Kogan and E. Petrank, "Wait-free queues with multiple enqueuers and dequeuers," in *Proceedings of the 16th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPOPP 2011, San Antonio, TX, USA, February 12-16, 2011* (C. Cascaval and P. Yew, eds.), pp. 223–234, ACM, 2011.

[8] G. Milman, A. Kogan, Y. Lev, V. Luchangco, and E. Petrank, "BQ: A lock-free queue with batching," in *Proceedings of the 30th on Symposium on Parallelism in Algorithms and Architectures, SPAA 2018, Vienna, Austria, July 16-18, 2018* (C. Scheideler and J. T. Fineman, eds.), pp. 99–109, ACM, 2018.

[9] R. Nikolaev and B. Ravindran, "wcq: A fast wait-free queue with bounded memory usage," *CoRR*, vol. abs/2201.02179, 2022.