Hybrid Relaxed Concurrent Data Structures

Goals: Define a specification of hybrid relaxed data structures, devise algorithms matching the specification.

Tools: Logic, algorithmic reasoning, programming.

Prerequisites: Maturity in math and algorithms, basic knowledge of distributed computing, basic concurrent programming skills.

Many data structures, such as queues and stacks, are notorious for being *concurrency-averse*, i.e., for not permitting efficient concurrent implementations. The reason is that concurrent threads have to contend on the same elements of such data structures, which incurs considerable synchronization costs. A popular way to improve performance is to relax the semantics by allowing some operation to return elements *out of order* [1, 4, 6, 7]. For example, a *k-out-of order* queue allows items to be dequeued out of FIFO order up to k elements. One can implement a k-out-of-order queue from k independent queues: at the cost of weaker consistency guarantees, the relaxed queue offers more parallelism and, as a result, exhibits significant performance gains.

Alternatively, one can also consider relaxations by allowing inconsistent responses *under contention* [3]. For example, concurrent dequeue operations on a relaxed queue may be allowed to return the same queue element. One can implement queues and stacks using basic read and write operations, which is, in general, impossible for k-out-of-order queues.

In this project, we follow the quest for scalable but consistent concurrency by considering *hybrid* relaxation. It makes sense to expect that in *sequential* executions, when no two operations contend on the shared data, our concurrent implementation should ensure strong semantics, i.e., create an illusion of an atomic object [5]. Under contention, when $k \geq 2$ operations are concurrent, we might want to expect the object to relax (up to k) the order in which the elements can be returned.

The plan is to study this notion with different levels of contention [2], from *interval contention* to *step contention* and different relaxation approaches, and to check performance of resulting data structures experimentally.

Milestones

1. Study the recent literature on relaxed concurrent data structures, starting from [1, 3, 4, 6, 7].
2. Formally define the notion of hybrid relaxation.
3. Implement relaxed versions of a queue, a stack, and a priority queue, using read-write operations and, if needed, stronger synchronization primitives.
4. If time allows, study the performance of resulting implementations.

Contact

Petr Kuznetsov
http://www.infres.enst.fr/~kuznetso/
petr.kuznetsov@telecom-paristech.fr
INFRES, Télécom Paris

Armando Castañeda
https://www.matem.unam.mx/fsd/armando
armando.castaneda@im.unam.mx
Instituto de Matemáticas, UNAM, México

References

[1] D. Alistarh, J. Kopinsky, J. Li, and N. Shavit. The spraylist: a scalable relaxed priority queue. In A. Cohen and D. Grove, editors, *Proceedings of the 20th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, PPoPP 2015, San Francisco, CA, USA, February 7-11, 2015*, pages 11–20. ACM, 2015.

[2] H. Attiya, R. Guerraoui, D. Hendler, and P. Kuznetsov. The complexity of obstruction-free implementations. *J. ACM*, 56(4), 2009.

[3] A. Castañeda, S. Rajsbaum, and M. Raynal. Relaxed queues and stacks from read/write operations. In *OPODIS*, 2020. https://arxiv.org/pdf/2005.05427.pdf.

[4] T. A. Henzinger, C. M. Kirsch, H. Payer, A. Sezgin, and A. Sokolova. Quantitative relaxation of concurrent data structures. In R. Giacobazzi and R. Cousot, editors, *POPL*, pages 317–328. ACM, 2013.

[5] M. Herlihy and J. M. Wing. Linearizability: A correctness condition for concurrent objects. *ACM Trans. Program. Lang. Syst.*, 12(3):463–492, 1990.

[6] C. M. Kirsch, H. Payer, H. Röck, and A. Sokolova. Performance, scalability, and semantics of concurrent FIFO queues. In Y. Xiang, I. Stojmenovic, B. O. Apduhan, G. Wang, K. Nakano, and A. Y. Zomaya, editors, *ICA3PP*, volume 7439 of *Lecture Notes in Computer Science*, pages 273–287. Springer, 2012.

[7] T. Zhou, M. M. Michael, and M. F. Spear. A practical, scalable, relaxed priority queue. In *ICPP*, pages 57:1–57:10. ACM, 2019.