POSTER: LOFT: Lock-Free Transactional Data Structures

Avner Elizarov
Technion
Israel
avnerel@cs.technion.ac.il

Guy Golan-Gueta
VMWare research
Israel
ggolangueta@vmware.com

Erez Petrank
Technion
Israel
erez@cs.technion.ac.il

Abstract

Concurrent data structures are widely used in modern multi-core architectures, providing atomicity (linearizability) for each concurrent operation. However, it is often desirable to execute several operations on multiple data structures atomically. We present a design of a transactional framework supporting linearizable transactions of multiple operations on multiple data structures in a lock-free manner. Our design uses a helping mechanism to obtain lock-freedom, and an advanced lock-free contention management mechanism to mitigate the effects of aborting transactions. When cyclic helping conflicts are detected, the contention manager re-orders the conflicting transactions execution allowing all transactions to complete with minimal delay. To exemplify this framework we implement a transactional set using a skip-list, a transactional queue, and a transactional register. We present an evaluation of the system showing that we outperform general software transactional memory, and are competitive with lock-based transactional data structures.

Keywords Parallel Algorithms, Concurrent Data Structures, Progress Guarantees, Lock-Freedom, Transactions

1 Introduction

Concurrent computing architectures have become widespread, raising the need for efficient and scalable concurrent algorithms and data structures. Concurrent data structures are designed to utilize all available cores, achieving good performance as well as consistent behavior in the form of linearizability of operations [12]. Many implementations of concurrent data structures were proposed in recent years (e.g., [2, 8, 13]), providing an abstraction of a sequential data structure that can be accessed concurrently. However, it is often desirable to have several operations, operating on multiple data structures appear to take effect simultaneously and atomically. Linearizability of a single operation does not always suffice [1, 6, 10]. For example, moving an item from one queue to another while maintaining the invariant that other threads always see it in exactly one queue cannot be supported by a regular concurrent queue without costly synchronization.

To ensure the atomicity of such operations one can use a global lock to synchronize all accesses to the data structures, but this approach limits concurrency significantly, hindering scalability. Furthermore, the use of locks is susceptible to deadlocks, live locks, priority inversions, etc. A different approach for obtaining atomicity is *Transactional Memory* [11]. Transactional memories allow the programmer to specify a sequence of instructions that take effect atomically or not at all. Transactional Memories can be implemented using specialized hardware (HTM) or using software (STM)[10, 14]. In these implementations all reads and writes of a transaction logically appear to occur at a single instant of time (or not at all), and intermediate states are not visible to other threads. This approach is programmer-friendly, simplifying programming in a concurrent envirnoment. However, HTMs are limited in transaction size and STMs carry a performance cost [3]. In both implementations, conflicting accesses to data cause transactions to abort, and re-execute. The conflict detection mechanism and the need to re-execute transactions create an overhead that reduces performance and foils progress guarantees.

Recently, transactional data structure libraries [1, 9] were proposed to deal with some of the above disadvantages. Transactional data structures limit transactions to only execute operations on data structures, but they provide transaction semantics for concurrent data structures, and support atomic transactions containing sequences of data structure operations. Transactional data structures use mechanisms that build on the specific implementations of these data structures to reduce both the overhead and the abort rate.

In this work, we propose a framework for linearizable execution of transactions on data structures called LOFT that supports full lock-freedom. In order to be used with LOFT, a lock-free data structure has to be extended to support an adequate API that we define in this work. The proposed LOFT mechanism executes LOFT transactions, which consist of LOFT data structures operations. Transactions are always executed atomically. The LOFT engine extends and adapts

PPoPP '19, February 16–20, 2019, Washington, DC, USA
© 2019 Copyright held by the owner/author(s).
ACM ISBN 978-1-4503-6225-2/19/02.
https://doi.org/10.1145/3293883.3301491

a standard helping mechanism to help concurrently executing transactions to complete. In addition, the LOFT engine employs advanced contention management for handling cyclic conflicts. Cyclic conflicts are detected dynamically and transactions are executed in an adequate order that avoids the conflict. Next, we present an optimization for a common special case, where transactions contain a predetermined control flow, and they contain operations whose operands are known upfront. We formalize this special case and use reordering to fully avoid transaction aborts. Measurements show that this optimization (when applicable) benefits performance significantly, due to avoided aborts.

We exemplify LOFT for transactions on sets, queues, and register objects. We implemented a transactional abstract set, by extending a lock-free linked-list with the required LOFT API. We then add a skip-list to allow fast indexing into the list elements. The obtained transactional set is efficient and allows a transaction with various operations on multiple sets to be executed atomically. Next, we implemented a transactional queue, and finally we added a transactional register. We implemented and measured the LOFT sets against software transactional memory and the transactional data structures of [6]. Results show that the LOFT mechanism performs better than STMs and transactional data structures (for most scenarios). In contrast to lock-based transactional data structures and lock-based STMs, LOFT provides a lock-free progress guarantee and an advanced contention management mechanism for cyclic helping conflicts.

2 Measurements

We compared the performance of our lock-free LOFT set to the one described in TDSL [1]. We also evaluated a transaction-friendly skip list [4] running on top of a TL2 STM [5], implemented in the Synchrobench micro-benchmark suite [7]. We considered two different workloads: a $read-oriented$ workload consisting of 90% *contains* operations, and a $write-oriented$ workload consisting of 90% *add* and *remove* operations.

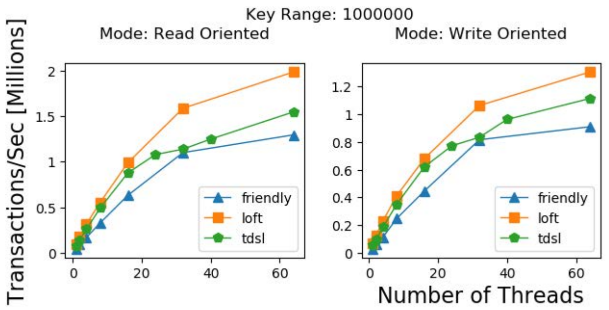

Figure 1. Throughput graphs of transactions over sets

We experimented with transactions containing operations executed on a collection of four sets. The graph in Figure 1 presents the average throughput with the workload distributions. Our algorithm outperforms the other algorithms for both the read and write oriented workloads. While all algorithms abort less, TDSL still aborts more frequently than LOFT which requires more traversals over the list and incurs smaller throughput.

References

[1] Guy Golan-Gueta Alexander Spiegelman and Idit Keidar. 2016. Transactional Data Structure Libraries. In *Proc. of the 37th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI).*

[2] Anastasia Braginsky, Nachshon Cohen, and Erez Petrank. 2016. CBPQ: High Performance Lock-Free Priority Queue. In *Euro-Par 2016, the 22nd International Conference on Parallel and Distributed Computing.*

[3] Calin Cascaval, Colin Blundell, Maged Michael, Harold W. Cain, Peng Wu, Stefanie Chiras, and Siddhartha Chatterjee. 2008. Software Transactional Memory: Why Is It Only a Research Toy? *Queue* 6, 5 (Sept. 2008), 46–58. https://doi.org/10.1145/1454456.1454466

[4] Tyler Crain, Vincent Gramoli, and Michel Raynal. 2012. A Contention-Friendly Methodology for Search Structures. (02 2012).

[5] Dave Dice, Ori Shalev, and Nir Shavit. 2006. Transactional Locking II. In *Proceedings of the 20th International Conference on Distributed Computing (DISC'06).* Springer-Verlag, Berlin, Heidelberg, 194–208. https://doi.org/10.1007/11864219_14

[6] M. Sagiv G. Golan-Gueta, G. Ramalingam and E. Yahav. 2013. Concurrent libraries with foresight. In *Proc. of the 34th ACM SIGPLAN Conf. on Programming Language Design and Implementation (PLDI).* 263–274.

[7] Vincent Gramoli. 2015. More Than You Ever Wanted to Know About Synchronization: Synchrobench, Measuring the Impact of the Synchronization on Concurrent Algorithms. *SIGPLAN Not.* 50, 8 (Jan. 2015), 1–10. https://doi.org/10.1145/2858788.2688501

[8] Timothy L. Harris. 2001. A Pragmatic Implementation of Non-blocking Linked-Lists. In *DISC'01.* http://portal.acm.org/citation.cfm?id=645958.676105

[9] Maurice Herlihy and Eric Koskinen. 2008. Transactional Boosting: A Methodology for Highly-concurrent Transactional Objects. In *Proceedings of the 13th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '08).* ACM, New York, NY, USA, 207–216. https://doi.org/10.1145/1345206.1345237

[10] Maurice Herlihy, Victor Luchangco, Mark Moir, and William N. Scherer, III. 2003. Software Transactional Memory for Dynamic-sized Data Structures. In *Proceedings of the Twenty-second Annual Symposium on Principles of Distributed Computing (PODC '03).* ACM, New York, NY, USA, 92–101. https://doi.org/10.1145/872035.872048

[11] Maurice Herlihy and J. Eliot B. Moss. 1993. Transactional Memory: Architectural Support for Lock-Free Data Structures. In *Proc. of the 20th Annual International Symposium on Computer Architecture (ISCA).* 289–300.

[12] Maurice Herlihy and Jeannette M. Wing. 1990. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Trans. Program. Lang. Syst.* 12, 3 (1990), 463–492.

[13] Maged M. Michael. 2002. High performance dynamic lock-free hash tables and list-based sets. In *Proceedings of the fourteenth annual ACM symposium on Parallel algorithms and architectures (SPAA '02).* ACM, New York, NY, USA, 73–82. https://doi.org/10.1145/564870.564881

[14] Nir Shavit and Dan Touitou. 1997. Software transactional memory. *Distributed Computing* 10, 2 (01 Feb 1997), 99–116. https://doi.org/10.1007/s004460050028