Research Interest Statement

Haosen Wen
Computer Science Department, University of Rochester
Rochester, NY 14627

Nov. 21, 2018

Abstract

My research focuses on persistent software transactional systems, a class of applications of concurrent data structures on non-volatile byte-addressable memories (NVBM). Given past explorations on both persistent semantics for concurrent data structures (by Joseph Izraelevitz et al.) and object-based transactional memories (by Maurice Herlihy et al. and Virendra Marathe et al.), I believe I can build a software transactional system that supports composable and flexible atomicity and persistence, and has better performance and scalability than existing options.

Overview

As traditional volatile RAM approaches the limit of its capacity growth, non-volatile byte-addressable memories (NVBM) are likely to be the future of main memory, with a tempting side-feature of persistency, which suggests research into corresponding data structures and applications. In the last few years, semantics of concurrent data structures in the context of persistency [1], together with concurrent data structures based on such semantics, enabled applications to be developed with NVBMs, and software transactional systems is one class of promising candidates.

Most such systems appeared take of software transactional memory (STM) for machines with either transient memories ([2, 3, 4]) or persistent memories ([5, 6]), and tend to offer an API that allows users to specify transactional code regions, loads and stores of which will take effect atomically at commit time. Such an API inevitably requires the system to consult some globally-reachable metadata in order to resolve conflicts and maintain consistency. However, such consultation may bring significant overhead, some of which might not be necessary for consistency and/or correctness. Hence, we are planning to design a transactional system with a more flexible multi-word compare-and-swap (MCAS) API and to equip it with necessary memory flushes, fences and recovery procedure to make it applicable to a system with NVBM and transient cache [1].

Summary of past work

My work so far has focused mainly on the implementation of concurrent data structures. In my first semester, my research on potential bottlenecks of inter-thread/inter-process communication brought some fundamental understandings of how factors like context switching, cache locality and TLB misses affect the performance of concurrent programs. This work provided guidance for later research on concurrency. This is still an open research topic for me.

During my second semester, I help implemented Polytree, a general framework for easily creating non-blocking concurrent tree structures, which provides thread-safe lookups, atomic node updates, and atomic sub-tree substitutions.

When implementing and comparing tree structures for Polytree, e.g, Bonsai Trees [7], we wished for a memory management scheme that would be robust to thread stalling, easy to use, and fast. Inspired by epoch-based memory reclamation algorithms [8, 9] and some state-of-art approaches like *Hazard Eras* [10], we developed *Interval-Based Memory Reclamation* [11], which I had the pleasure of presenting at PPoPP'18.

Prior to our work, epoch-based memory management schemes were the fastest and most user-friendly ones. To ensure correctness, an operation must protect (or reserve) all blocks that are (or will be) detached

after its start time (epoch) from being reclaimed, so that it can safely access every block that it can possibly reach. Unfortunately, an unbounded number of blocks can be reserved by some stalled thread, since all blocks detached after its start epoch will be reserved. Our work solved this problem by marking a finite range of epochs "reserved" per thread; all objects whose life intervals intersect with any "reserved" ranges will be protected from reclamation, so the total number of unreclaimable blocks is bounded.

Last spring, we attempted to build a persistent STM system based on Dalí [12], a periodic persistent hash map. In order to ensure the order of memory persistency operations (that flushes transient data from cache to NVM), persistent data structures prior to Dalí tend to issue expensive `pfence`s after every such operation. Dalí, on the other hand, lowered the overhead by only issuing `pfence`s every once in a while after a whole cache flush. On newer Intel x86 machines, however, cache flushing operations such as `clflush`'es are ordered with respect to each other and fence instructions, which undermined the need of `pfences` and thus made this idea less favorable. We then decided to build a persistent software transactional system based on current STM systems.

Ongoing works and future plans

I am implementing a non-blocking object-based persistent MCAS system. Preliminary results on a non-persistent version show better throughput and similar scalability results compared to RSTM [4]. My next step, apart from wrapping up the implementation and finishing a paper about it, is to develop a persistent non-blocking memory allocator to make this system truly failure-atomic.

Pavlovic et al. [13] announced a word-based persistent multi-word CAS system with an API of similar taste of our work. Our system will be different from theirs in two major ways: first, our system will be object-based, since we believe that the semantics of MCAS fits better in object-oriented programming in the sense that one might not want to focus on the size of objects when he/she is attempting to modify them atomically; second, our system will be non-blocking, which not only prevents common problems in lock-based systems like deadlocks and preempted threads stalling the whole system, but may also significantly simplify the recovery procedure: in a lock-based system, the thread holding a lock when a crash happens is responsible to bring the concrete memory state to a valid one that has a (well-defined) abstract state during recovery, which can be difficult when other threads are still making progress; with non-blocking progress, the memory will always have a valid concrete state, which makes the recovery trivial. Our current implementation is generally inspired by DSTM [2], ASTM [3] and RSTM [4], which are non-blocking STM systems for transient memory.

References

[1] Joseph Izraelevitz, Hammurabi Mendes, and Michael L. Scott. Linearizability of persistent memory objects under a full-system-crash failure model. In *Proceedings of the 30th International Conference on Distributed Computing*, DISC '16, pages 313–327, Paris, France, September 2016.

[2] Maurice Herlihy, Victor Luchangco, Mark Moir, and William N. Scherer III. Software transactional memory for dynamic-sized data structures. In *Proceedings of the 2003 ACM Symposium on Principles of Distributed Computing*, PODC '03, pages 92–101, Boston, MA, USA, July 2003.

[3] Virendra J. Marathe, William N. Scherer, and Michael L. Scott. Adaptive software transactional memory. In *Proceedings of the 19th International Conference on Distributed Computing*, DISC'05, pages 354–368, Cracow, Poland, September 2005.

[4] Virendra J. Marathe, Michael F. Spear, Christopher Heriot, Athul Acharya, David Eisenstat, William N. Scherer III, and Michael L. Scott. Lowering the overhead of nonblocking software transactional memory. In *First ACM SIGPLAN Workshop on Transactional Computing*, TRANSACT '06, Ottowa, ON, Canada, June 2006.

[5] Haris Volos, Andres Jaan Tack, and Michael M. Swift. Mnemosyne: Lightweight persistent memory. In *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, pages 91–104, Newport Beach, California, USA, 2011.

[6] Joel Coburn, Adrian M. Caulfield, Ameen Akel, Laura M. Grupp, Rajesh K. Gupta, Ranjit Jhala, and Steven Swanson. NV-Heaps: Making persistent objects fast and safe with next-generation, non-volatile memories. In *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVI, pages 105–118, Newport Beach, California, USA, 2011.

[7] Austin T. Clements, M. Frans Kaashoek, and Nickolai Zeldovich. Scalable address spaces using RCU balanced trees. In *Proceedings of the 17th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS XVII, pages 199–210, London, England, UK, March 2012.

[8] Keir Fraser. *Practical lock-freedom.* PhD thesis, Computer Laboratory, University of Cambridge, February 2004. No. UCAM-CL-TR-579.

[9] Thomas E. Hart, Paul E. McKenney, Angela Demke Brown, and Jonathan Walpole. Performance of memory reclamation for lockless synchronization. *Journal of Parallel and Distributed Computing*, 67(12):1270–1285, December 2007.

[10] Pedro Ramalhete and Andreia Correia. Brief announcement: Hazard Eras—Non-blocking memory reclamation. In *Proceedings of the 29th ACM Symp. on Parallelism in Algorithms and Architectures*, SPAA '17, pages 367–369, Washington, DC, USA, July 2017.

[11] Haosen Wen, Joseph Izraelevitz, Wentao Cai, H. Alan Beadle, and Michael L. Scott. Interval-based memory reclamation. In *Proceedings of the 21th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming*, PPoPP '18, Vienna, Austria, February 2018.

[12] Faisal Nawab, Joseph Izraelevitz, Terence Kelly, Charles B. Morrey III, Dhruva R. Chakrabarti, and Michael L. Scott. Dalí : A periodically persistent hash map. In *Proceedings of the 31st International Conference on Distributed Computing*, DISC '17, Vienna, Austria, September 2017.

[13] Matej Pavlovic, Alex Kogan, Virendra J. Marathe, and Tim Harris. Brief announcement: Persistent multi-word compare-and-swap. In *Proceedings of the 2018 ACM Symposium on Principles of Distributed Computing*, PODC '18, pages 37–39, Egham, United Kingdom, 2018. ACM.