Brief Announcement: Persistent Multi-Word Compare-and-Swap

Matej Pavlovic*
EPFL
matej.pavlovic@epfl.ch

Alex Kogan
Oracle Labs
alex.kogan@oracle.com

Virendra J. Marathe
Oracle Labs
virendra.marathe@oracle.com

Tim Harris*
tim.harris@gmail.com

ABSTRACT

This brief announcement presents a fundamental concurrent primitive for persistent memory – a persistent atomic multi-word compare-and-swap (PMCAS). We present a novel algorithm carefully crafted to ensure that atomic updates to a multitude of words modified by the PMCAS are persisted correctly. Our algorithm leverages hardware transactional memory (HTM) for concurrency control, and has a total of 3 persist barriers in its critical path. We also overview variants based on just the compare-and-swap (CAS) instruction and a hybrid of CAS and HTM.

ACM Reference Format:
Matej Pavlovic, Alex Kogan, Virendra J. Marathe, and Tim Harris*. 2018. Brief Announcement: Persistent Multi-Word Compare-and-Swap. In *PODC '18: ACM Symposium on Principles of Distributed Computing, July 23–27, 2018, Egham, United Kingdom.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3212734.3212783

1 INTRODUCTION

Emerging persistent memory (PM) technologies such as Intel and Micron's 3D XPoint [1] will offer the persistence of traditional storage technologies (NAND flash and disks), DRAM-like byte-addressability, and performance approaching that of DRAM (100-1000x faster than state-of-the-art NAND flash). The combination of these features of PM could profoundly affect the way we manage persistent data in modern applications. Traditionally, DRAM-resident data structures can be hosted in persistent memory. However, this task is complicated by the fact that the processor cache hierarchy will remain *nonpersistent* in the foreseeable future. Processor vendors, such as Intel [10], have proposed new hardware instructions to order persistence of stores to PM.

With the new instructions (flush/writeback cache lines, persist barriers), programmers are expected to rebuild applications to leverage PM. This is, however, a non-trivial endeavor. The principal challenge in building such algorithms centers around the fact that the caching layers in processor memory hierarchy are not persistent, and cache lines can be evicted (thus, persisted) in an arbitrary order. Algorithms must carefully order persistence of stores, using the aforementioned instructions, for correct recovery after failures.

Recognizing these programming challenges, several researchers and practitioners have proposed the use of various forms of transactions to access and manipulate data in persistent memory [4, 5, 16]. While transactions are a powerful abstraction to program PM, they incur significant performance overheads, which is why the field continues to be a subject of active research [12, 13]. In addition, more efficient concurrent data structure implementations can be built without transactions [6] or with a simple multi-word compare-and-swap (MCAS) primitive [7, 8, 14].

Wang et al. [17] recently proposed a lock-free persistent multi-word compare-and-swap (PMCAS) based on Harris et al's. MCAS algorithm [8]. Although an important advance, the algorithm is complex and suffers from significant performance overheads – for every PMCAS, it uses 5 CASes and 2 persist barriers per modified word in the critical path. In this brief announcement, we present new *durably linearizable* [2, 11] variants of a persistent multi-word compare-and-swap (PMCAS) primitive, where multiple words in PM can be updated atomically and in a crash-tolerant manner. Our algorithm, though blocking, requires just 1 CAS per modified word and 3 persist barriers in total in the critical path.

2 PERSISTENT MCAS ALGORITHM

We assume a conventional shared memory multicore system with deterministic threads that read and write shared memory using load, store, and CAS instructions as well as HTM transactions. We assume a fail-stop failure model, where a failure crashes all threads accessing the shared memory. The system can contain both persistent and nonpersistent (DRAM) memory. Processor buffers and caches are nonpersistent.

For simplicity, we focus on PMCAS-htm, a PMCAS variant that leverages the hardware transactional memory (HTM) feature available in some of the contemporary processors [9, 15]. In this variant, reads from memory locations being updated by a concurrent PMCAS block and wait for the PMCAS to complete. Toward the end, we briefly mention nonblocking reads, and two variants of PMCAS, one that uses the compare-and-swap (CAS) instruction instead of HTM, and another that uses both HTM and CAS.

For each PMCAS, the application creates a persistent *update structure* that holds old and new values of the target addresses. We assume that the application persistently tracks all these structures in use by potentially multiple threads (e.g. by having a persistent pool of update structures reachable from a designated "root" of a persistent region). These structures must be reachable from the recovery subsystem.

*Author was employeed at Oracle Labs during this work.

PODC '18, July 23–27, 2018, Egham, United Kingdom

ACM ISBN 978-1-4503-5795-1/18/07.
https://doi.org/10.1145/3212734.3212783