High Performance Hardware Transactional Memory does not Equal High Performance Transaction Systems

Justin Levandoski[1] Darko Makreshanski[2] Ryan Stutsman[3]

[1,3]Microsoft Research, [2]ETH Zurich
[1]justin.levandoski@microsoft.com, [2]darkoma@inf.ethz.ch, [3]rystutsm@microsoft.com

Hardware Transactional Memory

With the emergence of hardware transactional memory (HTM) shipping in commodity CPUs (as of Intel Haswell), we now have a promising transactional memory implementation that achieves great performance and is widely available. HTM ensures atomic execution of all of the loads and stores of a critical section, thereby relieving the programmer from thinking about fine-grained concurrency. The key to HTM's performance (from what Intel has released) is that it piggybacks on existing features in CPU micro-architectures to support transactions: for instance (1) CPU caches are used to store transaction buffers and provide isolation and (2) the CPU cache coherence protocol is used to detect conflicting accesses.

So What?

Many of today's high performance transaction systems assume data lives in main-memory; systems like SQL Server Hekaton, SAP HANA, HyPeR, and MemSQL all make this assumption. Many of these systems are written from scratch to address new bottlenecks now that I/O is out of the picture. A new primary bottleneck in these systems, however, is latching (also known as "locks" outside the database community). To achieve great multi-threaded performance, many of these systems eliminate (or at least reduce) the use of latches throughout the engine. For instance, SQL Server Hekaton is completely latch-free, while MemSQL uses latch-free skiplists as its access method.

To achieve latch-freedom, these systems use atomic CPU hardware primitives such as compare-and-swap (CAS) to manipulate state within the engine. This leads to great performance, but at a cost: non-trivial latch-free systems are notoriously difficult to architect and implement. Until recently, such designs were the only way to build high performance, multi-threaded, main-memory systems. With HTM now shipping, we have what seems like a seamless method to achieve multi-threaded parallelism with much less difficulty.

Our Proposal

What is unclear is if HTM is a "cure-all" drop-in solution to building high-performance transaction systems. Recent work (from the HyPeR team) has shown that, clearly, one cannot simply wrap a database transaction within an HTM transaction and expect great performance (due to constraints in HTM). The question is then: where are current HTM implementations useful within the engine? For the past year, we have been experimentally evaluating the interplay of HTM with the workhorse behind many modern high-performance transaction engines: the crafty latch-free access methods that serve as the hot path for data manipulation and retrieval. This talk will address the following questions.

- *Does HTM obviate the need for crafty lock-free design?* The answer is no. Even wrapping a single index probe in an HTM transaction exposes unreliability. We will cover a number of experimental scenarios where this is true. The numbers we present use memory-optimized single-threaded access methods with HTM as a drop-solution for multi-threaded concurrency. For real-world data, using HTM in this manner performs poorly; in some cases performing worse than serialized performance. We also discuss how HTM is not yet suitable for implementing many fine-grained B-tree concurrency techniques such as lock-coupling.

- *How does an HTM-based design differ from a lock-free design?* We present an analysis and experimental comparison of state-of-the-art HTM-based access methods with latch-free access methods shipping in production. Latch-freedom requires extra mechanism (e.g., epoch protection for memory safety). However, a latch-free design never aborts readers (due to use of copy-on-write). This feature is advantageous, especially when reading data with a high update rate. Meanwhile, an HTM-based update-in-place approach can degrade read performance by up to 4x in such cases.

- *Given the drawbacks of HTM and that latch-free designs are relevant, is there a design paradigm that marries the two?* Using HTM as a method to perform a multi-word CAS operation (MW-CAS) helps simplify latch-free design, since it allows atomic installation of operations that span multiple arbitrary locations. We discuss design patterns simplifying latch-free design using HTM-based MW-CAS, as well as some subtleties and pitfalls we experienced in practice.

We end the talk with a wish list for features we hope to see in future HTM releases that will ultimately help us design and build high performance systems. In summary, our message is (1) HTM is not (yet) a panacea for building high-performance transaction systems; (2) Crafty latch-free system designs still matter, and will for the foreseeable future since they fundamentally differ from HTM-based concurrency schemes; and (3) Curently, HTM can help simplify high-performance system design to a *small* extent, and we will show you how.