Technical Perspective: Scaling Dynamic Hash Tables on Real Persistent Memory

Kenneth A. Ross
Columbia University
kar@cs.columbia.edu

Byte-addressable persistent memory was considered in the data management community as long ago as 1986. Thatte saw the advantages for programmability in unifying the abstractions of byte-addressable RAM with persistence [2]. Thatte's context was object-oriented databases containing a variety data structures that would be awkward to transform into the block-oriented abstractions provided by typical secondary storage. Thatte's proposed physical instantiation of persistent memory was a disk-backed device, although it is unclear whether such a device was ever built. Thatte recognized the importance of recovery to the overall scheme.

Fast forward to 2017 when Intel released its Optane nonvolatile memory. Among the key benefits of Optane is byte-addressibility, making it a strong candidate for use as a persistent memory. There are many different aspects of Optane memory that might influence how data structures behave when implemented on Optane rather than on RAM. Traditional measures such as latency and throughput (which may differ for reads and writes) and capacity are important. Particular use cases may depend on other measures, such as the supported concurrency level and the speed of atomic operations. Some details of the device, such as the internal block size (analogous to the cache line size for RAM or the block size for a secondary storage device) may affect performance without being fundamental to the technology. Given that all of these measures are quite different from the corresponding parameters for RAM and for SSDs, the ideal data structure configurations for such devices may look quite different from those used before.

In this context, Lu et al. [1] have tackled the question of how best to design a persistent dynamic hash table using persistent memory, evaluating their design on the Optane platform. Persistent dynamic hash tables have broad applicability for storing data and indices in data management and other applications, and therefore constitute a workload worth optimizing. Prior work in this area included designs that were proposed before widespread availability of Optane devices. As Lu et al. [1] demonstrate experimentally, there were several "gotchas" in those implementations that caused poor performance on Optane because they accessed the device in a manner that turned out to be inefficient. Informed by the Optane device, Lu et al. [1] have engineered a highly performant implementation of persistent hash tables.

What makes this work stand out is the way many competing demands of the data structure are addressed in a balanced way. Unlike traditional hash tables, load factors are kept high by allowing multiple possible destinations for a key. Variable-length keys are supported. Failed searches are optimized by maintaining metadata that can short-circuit such look-ups. High concurrency is achieved through careful implementation of optimistic synchronization primitives. Fast recoverability is based on a timestamp-based scheme that allows the system to be available in constant time, with recovery work leading to somewhat degraded performance for a short period after recovery. The implementation is open-source, allowing others to build on these insights.

While the achievements of Lu et al. [1] are impressive, their paper highlights several issues for future research. It is not easy to program efficient data structures for devices like Optane, despite the byte-addressable interface. Complex issues such as leaks of persistent memory are more serious than leaks in RAM. Some of the choices made by Lu et al. [1] depend on knowing the internal block size of the device, something that may be different in future devices, and may vary across devices. Some kind of self-tuning might be needed to make the right choices based on measured performance rather than inside knowledge of such parameters.

The future role of persistent memory in database management systems remains open. Simply switching the underlying memory type to Optane without changing the data structures is not efficient [3]. For inner-loop data structures like hash tables, it may be worth the effort to write specialized code so that the advantages of persistent memory translate into tangible benefits for database users.

1. REFERENCES

[1] B. Lu, X. Hao, T. Wang, and E. Lo. Dash: Scalable hashing on persistent memory. *Proc. VLDB Endow.*, 13(8):1147–1161, 2020.

[2] S. M. Thatte. Persistent memory: A storage architecture for object-oriented database systems. In *Proceedings on the 1986 International Workshop on Object-Oriented Database Systems*, pages 148–159. IEEE Computer Society Press, 1986.

[3] Y. Wu, K. Park, R. Sen, B. Kroth, and J. Do. Lessons learned from the early performance evaluation of Intel Optane DC persistent memory in DBMS. In *Proceedings of the 16th International Workshop on Data Management on New Hardware*, DaMoN '20, 2020.