Managing Non-Volatile Memory in Database Systems

Alexander van Renen, Viktor Leis, Alfons Kemper, Thomas Neumann
Technische Universität München {renen,leis,kemper,neumann}@in.tum.de

Takushi Hashida, Kazuichi Oe, Yoshiyasu Doi, Lilian Harada, Mitsuru Sato
Fujitsu Laboratories {hashida.takushi,ooe.kazuichi,yosh-d,harada.lilian,msato}@jp.fujitsu.com

I. Introduction

Non-volatile memory (NVM), also known as Storage Class Memory (SCM), Persistent Memory (PMEM), and NVRAM, is a radically new and highly promising storage device. Technologies like PCM, STT-RAM, and ReRAM have slightly different features [1], but generally combine the byte addressability of DRAM with the persistence of storage technologies like SSD (flash). Because commercial products are not yet available, the precise characteristics, price, and capacity features of NVM have not been publicly disclosed (we resort to simulation for experiments). What is known, however, is that for the foreseeable future, NVM will be slower (and larger) than DRAM and, at the same time, much faster (but smaller) than SSD [2]. Furthermore, NVM has an asymmetric read/write latency—making writes much more expensive than reads. Given these characteristics, we consider it unlikely that NVM can replace DRAM or SSD outright.

The novel properties of NVM make it particularly relevant for database systems, but also present new architectural challenges. Neither the traditional disk-based architecture nor modern main-memory systems can fully utilize NVM without major changes to their designs. The two components most affected by NVM are logging/recovery and storage. Much of the recent research on NVM has optimized logging and recovery [3], [4], [5], [6], [7]. In this work, we instead focus on the storage/caching aspect, i.e., on dynamically deciding where data should reside (DRAM, NVM, or SSD).

Two main approaches for integrating NVM into the storage layer of a database system have been proposed. The first, suggested by Arulraj et al. [8], is to use NVM as the primary storage for relations as well as index structures and perform updates directly on NVM. This way, the byte addressability of NVM can be fully leveraged. A disadvantage is that this design can be slower than main-memory database systems, which store relations and indexes in main memory and thereby benefit from the lower latency of DRAM. Another downside of working directly on NVM is that there is no way to prevent eviction and any modification is potentially persisted. Therefore, any in-place write to NVM must leave the data structure in a correct state (similar to lock-free data structures, which are notoriously difficult) [9].

To avoid these issues, Kimura [10] proposed using a database-managed DRAM cache in front of NVM. Similar to a disk-based buffer pool, accesses are always performed on in-memory copies of fixed-size pages. However, accessing an uncached page becomes more expensive than directly accessing NVM, as an entire page must be loaded even if only a single tuple is accessed. Furthermore, neither of the two approaches supports very large data sets, as the capacity of NVM is limited compared to SSDs.

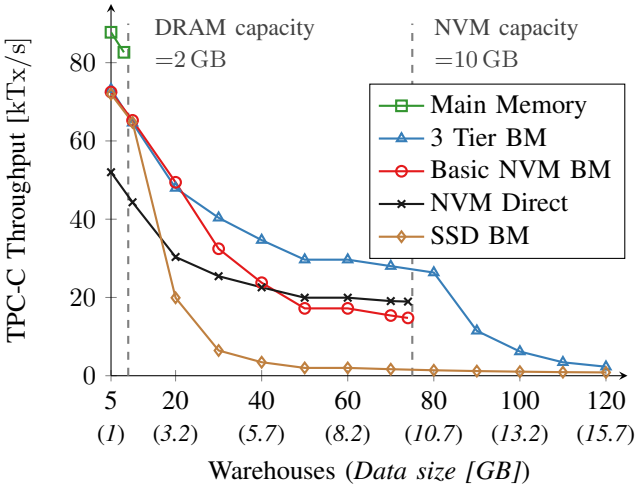

Fig. 1: **TPC-C** – Performance in TPC-C for an increasing number of warehouses. The capacity of DRAM, NVM, and SSD is set to 2 GB, 10 GB, and 50 GB, respectively.

We take a less disruptive approach and implement NVM as an additional caching layer. We thus follow Michael Stonebraker, who argued that NVM-DIMMs are ...

> *"...not fast enough to replace main memory and they are not cheap enough to replace disks, and they are not cheap enough to replace flash."* [11]

Figure 1 shows the performance characteristics and capacity restrictions of different system designs (*Buffer Manager* is abbreviated as *BM*). Besides the two NVM approaches ("Basic NVM BM", "NVM Direct"), we also show main-memory systems ("Main Memory"), and traditional SSD buffer managers ("SSD BM"). Each of these designs offers a different tradeoff in terms of performance and/or storage capacity.

In this talk, which is based on work published at SIGMOD

2018 [12], we present a novel storage engine that simultaneously supports DRAM, NVM, and flash while utilizing the byte addressability of NVM. As the "3 Tier BM" line indicates, our approach avoids performance cliffs and performs better than or close to that of specialized systems. NVM is used as an additional layer in the storage hierarchy supplementing DRAM and SSD [13], [2]. Furthermore, by supporting SSDs, it can manage very large data sets and is more economical [14] than the other approaches.

II. NVM-Optimized Database Storage

Our goal is a system that performs almost as well as a main-memory database system on smaller data sets but scales across the NVM and SSD storage hierarchy while gracefully degrading in performance. For this purpose, we design a novel DRAM-resident buffer manager that swaps cache-line-grained data objects between DRAM and NVM—thereby optimizing the bandwidth utilization by exploiting the byte addressability of NVM. Scaling beyond DRAM to SSD sizes led us to rely on traditional page-grained swapping between NVM and SSD. Between DRAM and NVM, we adaptively differentiate between full page memory allocation and mini page allocation to further optimize the DRAM utilization. This way, individual "hot" data objects that are resident on mostly "cold" pages are extracted via the cache-line-grained swapping into smaller memory frames. Only if the mini page overflows, is it transparently promoted to a full page—but it is still populated one cache-line at a time. We also devise a pointer swizzling scheme that optimizes the necessary page table indirection in order to achieve nearly the same performance as pure main-memory systems. This obviates any indirection but incur the memory wall problem once the database size exceeds DRAM capacity.

These techniques are illustrated in Figure 2. To leverage the byte-addressability of NVM, we cache NVM accesses in DRAM at cache-line granularity, which allows for the selective loading of individual hot cache lines instead of entire pages (which might contain mostly cold data). To more efficiently use the limited DRAM cache, our buffer pool transparently and adaptively uses small page sizes. At the same time, our design also uses large page sizes for staging data to SSD—thus enabling very large data sets. We use lightweight buffer management techniques to reduce the overhead of in-memory accesses. Updates are performed in main memory rather than directly on NVM, which increases endurance and hides write latency.

Our experimental evaluation is based on standard database workloads like YCSB and TPC-C. We evaluated three approaches for integrating NVM into the storage layer of a database system: One that works directly on NVM, a FOEDUS-style buffer manager based on fixed-size pages, and our novel cache-line optimized storage engine. We found that by taking the byte addressability into account, it becomes possible to outperform the other two approaches while supporting large data sets on SSD as well.

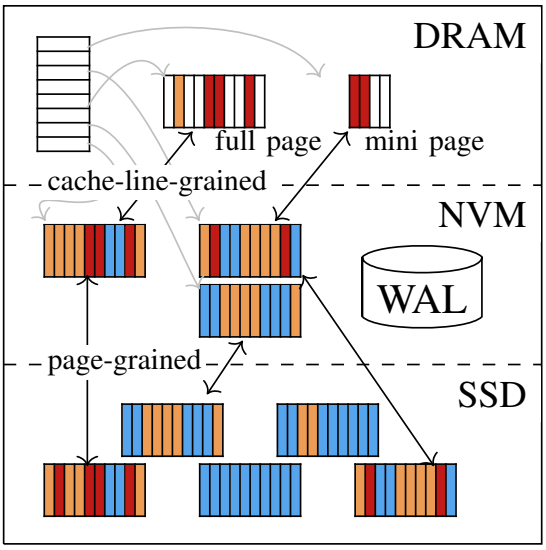

Fig. 2: Illustration of the core techniques. On SSD and NVM, data is stored on fixed-size pages (16 KB). When accessing data from NVM, cache lines may be loaded individually exploiting the byte-addressability of NVM. To save main memory, we also support "mini pages" (1 KB instead of 16 KB).

References

[1] S. Mittal and J. S. Vetter, "A survey of software techniques for using non-volatile memories for storage and main memory systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 5, pp. 1537–1550, 2016.

[2] S. Dulloor, A. Roy, Z. Zhao, N. Sundaram, N. Satish, R. Sankaran, J. Jackson, and K. Schwan, "Data tiering in heterogeneous memory systems," in *EuroSys*, 2016.

[3] J. Arulraj, M. Perron, and A. Pavlo, "Write-behind logging," *PVLDB*, vol. 10, no. 4, pp. 337–348, 2016.

[4] I. Oukid, D. Booss, W. Lehner, P. Bumbulis, and T. Willhalm, "SOFORT: a hybrid SCM-DRAM storage engine for fast data recovery," in *DaMoN*, 2014.

[5] J. Huang, K. Schwan, and M. K. Qureshi, "NVRAM-aware logging in transaction systems," *PVLDB*, vol. 8, no. 4, pp. 389–400, 2014.

[6] T. Wang and R. Johnson, "Scalable logging through emerging non-volatile memory," *PVLDB*, vol. 7, no. 10, pp. 865–876, 2014.

[7] R. Fang, H. Hsiao, B. He, C. Mohan, and Y. Wang, "High performance database logging using storage class memory," in *ICDE*, 2011, pp. 1221–1231.

[8] J. Arulraj, A. Pavlo, and S. Dulloor, "Let's talk about storage & recovery methods for non-volatile memory database systems," in *SIGMOD*, 2015, pp. 707–722.

[9] P. Götze, A. van Renen, L. Lersch, V. Leis, and I. Oukid, "Data management on non-volatile memory: A perspective," *Datenbank-Spektrum*, vol. 18, no. 3, pp. 171–182, 2018.

[10] H. Kimura, "FOEDUS: OLTP engine for a thousand cores and NVRAM," in *SIGMOD*, 2015, pp. 691–706.

[11] M. Stonebraker, "How hardware drives the shape of databases to come," https://www.nextplatform.com/2017/08/15/hardware-drives-shape-databases-come/, accessed: 2017-11-02.

[12] A. van Renen, V. Leis, A. Kemper, T. Neumann, T. Hashida, K. Oe, Y. Doi, L. Harada, and M. Sato, "Managing non-volatile memory in database systems," in *SIGMOD*, 2018, pp. 1541–1555. [Online]. Available: https://db.in.tum.de/~leis/papers/nvm.pdf

[13] P. Bonnet, "What's up with the storage hierarchy?" in *CIDR*, 2017.

[14] J. Gray and G. R. Putzolu, "The 5 minute rule for trading memory for disk accesses and the 10 byte rule for trading memory for CPU time," in *SIGMOD*, 1987, pp. 395–398.