Data-Intensive Systems on Evolving Memory Hierarchies

Ilia Petrov, Daniel Bausch, Robert Gottstein, Alejandro Buchmann

DVS, Technische Universität Darmstadt, Germany
{petrov, bausch, gottstein, buchmann}@dvs.tu-darmstadt.de

Abstract: New memory technologies with radically different properties are appearing. A substantial hardware and architectural redesign is required if they are to be used energy and performance efficient in a high-performance data-intensive system. In this paper we explore different alternatives and claim that the changing memory hierarchy and the challange for their energy efficient usage at the software layer disruptively impact the architecture and algorithms in any data-intensive system; direct control of the raw storage media is required yielding different hardware interfaces and software layers; data placement and versioning are emerging as new research fields promising most efficient usage in terms of performance and energy consumption.

1 Introduction

New storage technologies with radically different properties are appearing. These change not only the established principles of building storage systems, but also impact above all the established balance in computer systems as well as many assumptions in the architecture of data-intensive systems. To maximize performance, database architecture and data

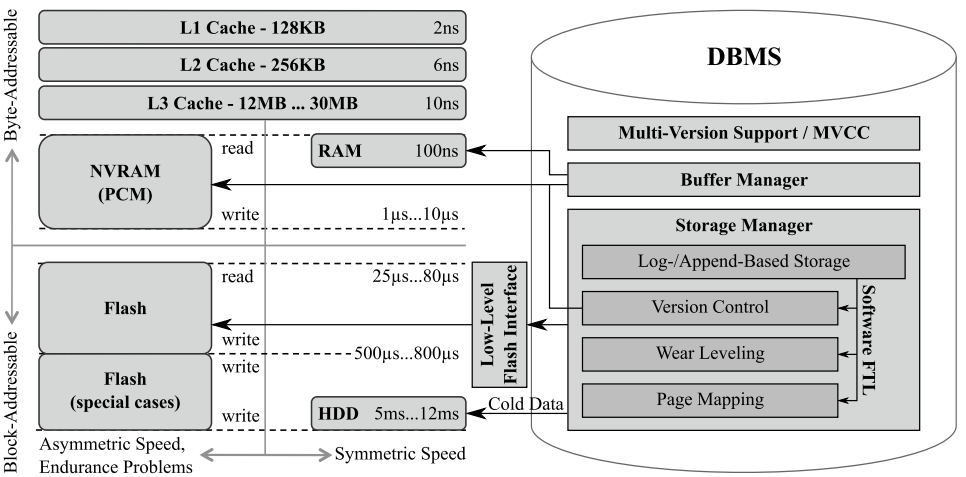

Figure 1: Changing Memory Hierarchy and Proposed Usage in Database Management Systems

Table 1: Media Properties (based on [CGN11])

	DRAM	NVM/PCM	NAND Flash	HDD
Page Size	64B	64B	4KB	512B
Read Latency (Page)	70–100ns	80–100ns	50µs	5ms
Write Latency (Page)	70–100ns	1µs	500µs	5ms
Write Bandwidth	1GB/s (die)	100MB/s (die)	40MB/s (die)	170MB/s (drive)
Erase Latency			1.5ms	
Density	1x	4x	4x	
Endurance	∞	10^8 cycles	10^5 cycles	MTTF: 1.6×10^6h
R/W Energy [J/GB]	0.8/1.2	1/6	1.5/17	65/65
Idle Power [mW/GB]	100	1	1	10

processing algorithms are designed and engineered to efficiently use hardware and leverage its properties. This is especially true for the storage subsystem since most database systems are I/O bound. Growing demand for e.g. cloud-based services creates new requirements for data-intensive systems, which often solely focus on performance and neglect software-driven energy efficiency (architecture and algorithms), while hardware energy efficiency (e.g. cooling and efficient power supplies) is already applied. The need for a software redesign caused by upcoming new technology offers the opportunity to address performance and software-driven energy efficiency at the same time.

Consider the following example: present database systems assume a memory hierarchy comprising CPU cache, main memory and disk with a significant access gap between memory and rotational disks. To estimate I/O requirements, they merely count random and sequential I/O operations ignoring the size and type (read or write). Furthermore, writes and updates are performed in-place (re-write). Such a model is valid since the I/O behavior of the "traditional" memory hierarchy is symmetric, i.e. reads and writes are equally fast and consume the same amount of energy. New technologies (e.g. Non-Volatile Memories) yield new levels in the memory hierarchy (Figure 1) and narrow the access gap. The latency spectrum evolves to an operation-dependent continuum. These also have different addressing modes and are asymmetric, i.e. reads are (significantly) faster and need less energy than writes. An in-place update of a 4KB page would cost up to 2ms (59.6µJ) on the average Flash SSD, while an out-of-place update would cost approx. 500µs (9.4µJ). Writing 4KB on a single PCM chip would cost 1ms and on a DIMM module 100µs [ACM+11]. Therefore algorithmic properties and assumptions, data organization, transaction management and in essence the whole database architecture have to be rethought.

In the present paper, we discuss multiple aspects of a reorganized database architecture, targeting new storage technologies. It is based on the following assumptions:

(i) *The memory hierarchy changes* significantly: new levels are added; these have different technological characteristics (Table 1). Compared to traditional technologies (RAM, HDD) their properties are: asymmetric (read vs. write) performance and energy consumption; endurance issues; different addressability modes; high internal parallelism [CGN11].

Data Placement emerges as a new research problem as will be explained in Section 5.

(ii) In order to translate the efficient use of new storage technologies into performance gains, data-intensive systems would need *direct control of the raw storage media* thus impacting current hardware interfaces and software layers. Hardware interfaces (e.g. block I/O, SATA) will evolve from providing backward compatibility to offering high performance, low latency and direct exposure (PCIe, NVMHCI). Non-Volatile Memories (NVM) will have DRAM packaging and be used accordingly; not as a part of an attached storage system. Software layers (OS, Drivers, Storage, Buffer and Cache management) will merge and some will disappear. The remaining layers should efficiently exploit their knowledge about data management, placement and energy behaviour. These impact both existing hardware interfaces (e.g. block I/O, SATA2/3, FTL), as well as the software layers of data-intensive system's architecture (OS, Drivers, Storage, Buffer and Cache management).

(iii) A corollary of the above assumption is that some traditional OS functions will be *integrated* into data-intensive software. Such an approach can be observed in large commercial database solutions.

(iv) As discussed in [EHJ+10], new I/O technologies are best utilized as *high-performance host-based storage* for data-intensive systems. This promotes shared nothing architectures enabling scalability and parallelism as well as direct control.

In the rest of the paper we present a detailed discussion of the following aspects, taking a database perspective: (a) evolution of Flash storage management in data-intensive systems; (b) utilization of Non-Volatile Memories (NVM) in computer and data management systems.

We claim that, to optimally utilize the properties of Flash memories, new hardware interfaces (PCIe) need to be employed; a departure from existing block I/O techniques needs to take place; the FTL (Flash Translation Layer) concept and functionality needs to be rethought. All these aim at increasing parallelism, decreasing access latencies by enabling direct access to the raw medium. We assume an FTL separation where the page mapping and wear-leveling functionality are part of the storage manager of the database system (or file system driver). Commercial database systems can operate natively on raw devices, hence, the proposal is a continuation of an already existing concept. The hardware FTL can perform read/write cache control and low level garbage collection. We claim that performance and endurance issues can be addressed by careful data placement, log-/append-based storage management, and the concept of versioning in database systems. It can also be used to address endurance issues on NVM. The buffer manager of a database system should address asymmetry, not only the access gap (especially with the advent of NVM). We propose a simple staging model, where data can be staged down the memory hierarchy respecting the different properties of each level and the type of data. In Section 5, we claim that directly controlled NVM can be used in a number of ways for typical database tasks, e.g. logging, buffer and version management.

In Section 3 we will present a new hardware/software architecture (visualized in Figure 1) with an added NVM layer and direct control of the Flash storage. Section 4 further explains why databases systems do not need a general purpose FTL and might perform better

with a simpler low-level interface. In Section 5 we discuss the problem of efficient data placement. Our proposals should lead to a system that is able to deliver more transactions per unit of time while using less energy per transaction.

2 Energy and Performance

Data-Intensive Systems are usually energy efficient at high peaks of load when all hardware is busy, they are also efficient on idle when the hardware is able to enter a deep sleep state. Anyhow, they are not efficient when experiencing weak load and the hardware cannot enter a sleep state or when the load changes from idle to non-idle, because turned-off hardware has to be turned on again. The key to performance and energy efficiency is the direct usage of the available resources while using only the smallest effective amount. Approaches considering energy-proportional hardware systems and dynamic powering of server cluster nodes have been made in [THS11] and [SH11]. FTL processes (Section 4) hide complexity but introduce uncertainty in energy usage; hence, merging or avoiding them to gain better performance also leads to a more efficient energy utilization. The optimal usage pattern of different storage media (RAM, NVM, Flash, disk) differs. Therefore a DBMS needs to be aware of those differences when placing data (see Section 5).

3 Architecture

Figure 1 also provides an overview of the target architecture's building blocks. Versioning is a critical component of the proposed architecture. In such an environment, multiple decoupled versions of a data item exist: every update produces a new version (out of place), not a direct modification; read operations use stable versions thus reads are never blocked. Such a decoupling leverages important hardware and software properties. On the software side there are implications on the transaction model, indexing and data organization. More importantly, however, data placement of different versions is facilitated. On the hardware side, the full potential of the asymmetric performance of new I/O technologies can be utilized. Modern CPUs support a cheap creation of memory snapshots. Because of the wear problems both Flash and NVMs have to support out-of-place updates which fits versioning very well. FTL algorithms for Flash devices natively operate on several versions of the data (see also Section 4). Such functionality should be integrated within the DB storage. New versions of database items are co-located and placed together in separate storage blocks, which can be logically appended to the end of database storage, which is realized as a circular log. This matches well the properties of Flash (and NVM) [SAJA09]. Random writes are converted to sequential writes, wear-leveling is facilitated and garbage collection is automated and simplified, on-the-fly free space management is supported by copy-on-write operations, all I/O operations can be implemented in atomic manner by using the hardware/software structure of the FTL (Section 4). A database system can even control caching parameters of the hardware depending on the current workload [SXX+09].

4 Flash Translation Layer

Most of today's Flash based SSDs are meant as drop-in HDD replacements. Therefore, they emulate a block-oriented interface and use the same SATA connectors to ensure compatibility. A *Flash Translation Layer* (FTL) implements these interfaces and hides Flash specifics by performing: (i) translation between *Logical Block Addresses* (LBA) and physical page locations on the Flash chips; (ii) wear-leveling; (iii) background garbage collection. Different mapping approaches exist: page-mapping, block-mapping, and hybrid-mapping [CPP+09]. Processes are encapsulated as a black box, simplyfing the access but inevitably leading to energy-consuming busy states where the hardware may be active although the software does not issue an access.

We advocate a new generation of Flash SSDs that allows direct access to the Flash media. Such SSDs would unveil their full hardware parallelism to applications delivering high performance and would allow non-redundant wear and free-space management. Only the low-level Flash operations *read*, *write*, and *erase* would be carried out by hardware or firmware logic. Higher level tasks like mapping, garbage collection, wear-leveling, and scheduling would be performed by software running on the main CPU. This software could be a driver or even the data-intensive application itself. We opt for the latter which is an extrapolation of the continued efforts of commercial database systems (e.g. Informix [IBM01]) to work with raw storage. A DBMS could use its knowledge about data structures and expected future access patterns to optimize performance and the energy consumption of the system.

The most similar existing approach comes with Fusion-io's PCIe Flash memory cards. Their driver emulates a continuous virtual memory segment *(Virtualized Flash Storage – VFS)* to higher software layers, while the *VFS* layer performs mapping, wear-leveling, and garbage collection. This is used, for example, by the *Direct File System (DFS)* [JBFL10] which provides a high performance file system on top of *VFS*. Nevertheless, no direct control from data-intensive applications is possible, e.g. an application running on top of DFS/VFS is not able to control the timing of the garbage collection algorithms or specify exact physical locations for write accesses to optimize its data organization.

5 Data Placement

The new memory hierarchy opens a rich set of possibilities for placing different data on different layers. The suitable hierarchy layer can be selected based on freshness, update frequency, response time, transactional consistency, etc. We identify data placement as a new problem. The present approaches focus on access gap minimization and cache efficiency and solve only a small subset of the new problems.

As mentioned in the sections above, NVMs are exposed to wear and the DBMS' Storage Manager (full control of the devices activities) has to guarantee that memory cells are evenly worn. The combination of versioning and out-of-place updates using a cyclic storage manager implicitly gives the functionality of wear-leveling. As soon as the log space is recycled, a naïve garbage collector process transparently merges recent versions

with new versions and frees space that is occupied by invisible versions (collect and compact on demand). The refinement of this techniques is still required. It depends on the DBMS schema whether very old (invisible) versions are maintained, garbage collected or archieved (e.g. on HDD). The introjection of knowledge about the placement of a previous version of a page enables the creation of before- and after-images at the storage manager level and therefore simplifies logging and recovery. This reduces write overhead and the amount of written data, which improves energy efficiency – the less data there is to write, the less energy is needed. Before and after images are implicitly contained within the database and do not have to be logged separately.

Shadow Paging is a concurrency control algorithm based upon indirect mapping and out-of-place updates. Advantages of Shadow Paging are simplified logging (minimized or completely eliminated) and out-of-place updates. Shadow Paging is not used on HDDs since it destroys clustering and therefore creates random read patterns, which are slow on HDDs but fast on NVMs. In order to extend Shadow Paging to support the new storage architecture and multi-versioning, we propose the following (not exclusive list of) points: (i) On an update of a page, a new page is created out of place (no overwrite). (ii) At commit of a transaction, the new page is set most recent (after image) and the previous version is maintained (before image). (iii) The granularity of the update has changed and locking of whole segments is not necessary any more. (iv) Updates are versioned (e.g. by timestamps). These points avoid direct overwrites, extensive logging (no forced flushes) and circumvents the need for the on-device garbage collection, wear-leveling and translation layer, furthermore they can be combined with the cyclic log storage. Reading transactions do not have to be blocked according to protocols that are based on timestamp ordering (e.g. Snapshot Isolation). This is especially profitable on NVMs. Using Snapshot Isolation, the unit of versioning are tuples while shadow paging uses pages. Further research on the applicability of Shadow Paging on supporting tuple-versioning is required.

A staging of versions, where hot data is kept on NVM and cold data is staged to SSD is also valid to assume. Single tuple versions can be persisted to NVM according to the concurrency control mechanism (reconsider *steal – no force* paradigm). As they are non-volatile, they can be grouped, packed and sequentially written. Sequentialization of writes implies a trade-off in DRAM usage, more DRAM enables a better sequentialization but DRAM also consumes more energy than NVMs. NVM is able to handle read accesses nearly as fast as DRAM which makes it valid to use DRAM primarily as a write cache for NVMs.

Reading and writing cost varies between different storage technologies. Therefore, the eviction strategy of the buffer manager needs to account for those device properties in its priority calculations. When, for example, a read from NVM is cheaper than a write, the eviction strategy should prefer to choose a clean page for eviction. NVMs support smaller I/O granularities than Flash memories; thus we suggest the use of *multi granularity pages* which have different sizes, depending on which media they are persisted. It is also possible to create differential updates on NVM and full updates on Flash memories. Versions created by a transaction are packed and persisted to NVM first. This complements the concepts of write-ahead logging and recovery.

6 Conclusions

The advent of new fast non-volatile memories like *Phase Change Memory* adds another layer to the storage hierarchy. The access gap between volatile memory (RAM) and stable storage shrinks. Assuming direct control over storage media, database management systems will be more capable to choose optimal destinations for certain data fragments than general-purpose abstraction layers.

This impacts DBMS' design in a variety of places: (i) Buffer and storage manager must choose on which medium changed data is stored while they are required to prevent the memory cells of the destination from being worn out by skewed write distributions. In addition to the speed of the memories, they need to respect asymmetric read/write latencies. (ii) Data contained in NVM can be excluded from read buffering. (iii) Logging can be simplified by techniques guaranteeing atomic modifications. (iv) Lower access latencies allow to reconsider *Shadow Paging* for concurrency control. (v) Versioning becomes a central concept.

Acknowledgements This work was supported by the DFG (Deutsche Forschungsgemeinschaft) project "Flashy-DB".

References

[ACM+11] A. Akel, A.M. Caulfield, T.I. Mollov, R.K. Gupta, and S. Swanson. Onyx: a protoype phase change memory storage array. *Proc. HotStorage*, page 74, 2011.

[CGN11] Shimin Chen, Phillip B. Gibbons, and Suman Nath. Rethinking Database Algorithms for Phase Change Memory. In *Proc. CIDR*, pages 21–31, 2011.

[CPP+09] Tae-Sun Chung, Dong-Joo Park, Sangwon Park, Dong-Ho Lee, Sang-Won Lee, and Ha-Joo Song. A survey of Flash Translation Layer. *J. Syst. Archit.*, 55:332–343, May 2009.

[EHJ+10] Evangelos Eleftheriou, Robert Haas, Jens Jelitto, Mark A. Lantz, and Haralampos Pozidis. Trends in Storage Technologies. *IEEE Data Eng. Bull.*, 33(4):4–13, 2010.

[IBM01] IBM. *Administrator's Guide for IBM Informix Dynamic Server, Version 9.3*, chapter 1, page 9. IBM, 2001.

[JBFL10] William K. Josephson, Lars Ailo Bongo, David Flynn, and Kai Li. DFS: A File System for Virtualized Flash Storage. In *Proc. FAST*, pages 85–100, 2010.

[SAJA09] Radu Stoica, Manos Athanassoulis, Ryan Johnson, and Anastasia Ailamaki. Evaluating and repairing write performance on flash devices. In *Proc. DaMoN*, pages 9–14, 2009.

[SH11] Daniel Schall and Volker Hudlet. WattDB: an energy-proportional cluster of wimpy nodes. In *Proceedings of the 2011 international conference on Management of data*, SIGMOD '11, pages 1229–1232, New York, NY, USA, 2011. ACM.

[SXX+09] Ji-Yong Shin, Zeng-Lin Xia, Ning-Yi Xu, Rui Gao, Xiong-Fei Cai, Seungryoul Maeng, and Feng-Hsiung Hsu. FTL design exploration in reconfigurable high-performance SSD for server applications. In *Proc. ICS*, pages 338–349. ACM, 2009.

[THS11] Yi Ou Theo Härder, Volker Hudlet and Daniel Schall. Energy Efficiency is not Enough, Energy Proportionality is Needed! In *Proc. DSFAA*, pages 226–239, 2011.