DBMS on Modern Storage Hardware

Ilia Petrov [*1], Robert Gottstein [#2], Sergej Hardock [#3]

[*] *Reutlingen University, Germany*
[1] ilia.petrov@reutlingen-university.de

[#] *Technical University Darmstadt, Germany*
[2] gottstein@dvs.tu-darmstadt.de
[3] hardock@dvs.tu-darmstadt.de

***Abstract*—In the present tutorial we perform a cross-cut analysis of database systems from the perspective of modern storage technology, namely Flash memory. We argue that neither the design of modern DBMS, nor the architecture of Flash storage technologies are aligned with each other. The result is needlessly suboptimal DBMS performance and inefficient Flash utilization as well as low Flash storage endurance and reliability.**

We showcase new DBMS approaches with improved algorithms and leaner architectures, designed to leverage the properties of modern storage technologies. We cover the area of transaction management and multi-versioning, putting a special emphasis on: (i) version organisation models and invalidation mechanisms in multi-versioning DBMS; (ii) Flash storage management and especially on append-based storage in tuple granularity; (iii) Flash-friendly buffer management; as well as (iv) improvements in the searching and indexing models. Furthermore, we present our NoFTL approach to native Flash access that integrates parts of the Flash-management functionality into the DBMS yielding significant performance increase and simplification of the I/O stack. In addition we, cover the basics of building large Flash storage for DBMS and revisit some of the RAID techniques and principles.

I. Structure and Organization of The Tutorial

We intend to present a 1.5 hour tutorial on the above topic (the material can be easily extended to 3 hours, if required by the organisers). The target audience is database researchers and practitioners with interests in data management on modern storage hardware. The proposed tutorial is original work and as such has not been presented elsewhere. The contents are outlined below and described in detail in Section III. In brackets we also point out the presenter and duration of the respective section.

- Trends and Advances in Modern Storage Hardware and Data Management (I. Petrov, 15 min.)
- Building Large Flash-based DBMS Storage (I. Petrov, 10 min.)
- Flash-Aware Buffer Management (R. Gottstein, I. Petrov, 10 min.)
- Transaction Management and Multi-Versioning (R. Gottstein, 10 min.)
- Indexing in Multi-Version DBMS (R. Gottstein, 10 min.)
- Append-Based Storage Management (R. Gottstein, 15 min.)
- DBMS on Native Flash (S. Hardock, I. Petrov, 15 min.)
- Wrap-up (I. Petrov, 5 min.)

II. Introduction

Over the recent years novel storage technologies such as Flash memories evolved and are now widely spread. The architecture and algorithms of database systems and data-intensive systems as a whole are built around the properties of spinning disk storage technologies. Many basic design and algorithmic assumptions have been made to efficiently use the strengths of disk based storage and compensate for the performance hazards. Some of those assumptions are 20-30 years old and reflect outdated hardware characteristics.

Over the last decade we witnessed several breakthroughs in I/O technologies that have important implications to basic assumptions in data management. Some of the key characteristics such technologies bring along compared to traditional spinning disk storage are: (i) read/write asymmetry; (ii) low latencies; (iii) high random performance; (iv) endurance/longevity; (v) addressability (byte, block); (vi) energy consumption. Consider the trend towards growing database page sizes, existing query execution algorithms, buffer management strategies, access paths, transaction processing techniques, even the cost functions of a query optimizer – in essence the whole DBMS architecture is designed to compensate for the access gap between memory and disk.

Jim Gray and Prashant Shenoy gave us a powerful 'collection of rules' to estimate the impact hardware development [1]. Over the next decade several important trends are expected to continue. According to Bechtolsheim et al. [2] by 2022 computer systems will have: (i) 1000 cores per chip; (ii) 64GB DRAM chips yielding more than 128TB RAM per server; (iii) high bandwidth (memory: 2.5TB/s, I/O: 250GB/s); (iv) 1TB Flash chips yielding more than 8TB per Flash drive. In addition, Non-Volatile Memories such as the Memristor [3] or Phase-Change Memories (PCM) are expected to appear. As summarized by Chen et al. [4] PCMs are persistent byte addressable (RAM-like) memories with asymmetric latency and bandwidth, denser than NAND Flash and DRAM and exhibiting endurance issues. Based on the above figures, more than 512TB of NVM in the average computer system can be expected by 2022.

III. Tutorial Outline

In the present tutorial we examine the influence of the new I/O technologies on data-intensive systems. We start with a concise summary of the technological characteristics of modern storage technologies. The main focus of this tutorial is on their influence on different aspects of data management:

(i) Firstly, we describe the intricacies of building large Flash-based storage space for database systems and how established technologies such as RAID hit performance and scalability limitations. (ii) Secondly, we take a closer look at how buffer management in current multi-versioning DBMS (MV-DBMS) is impacted by modern storage hardware, provide an overview of state-of-the-art techniques and present the FBARC approach to a write clustering buffer manager. (iii) Thirdly, we analyze the transactional model in a MV-DBMS and show that such models are optimized for legacy hardware. Furthermore, we present an approach called Snapshot Isolation Append Storage (SIAS) that introduces a novel invalidation model and version organisation paradigm, complementing the properties of new storage technologies. SIAS is a combination of multi-version concurrency control with append storage in tuple granularity, multi-version indexing, simplified buffer management and read optimisations that leverage the properties of the aforementioned combination. (iv) Fourthly, based on the observation that the Flash devices nowadays are used in a legacy mode, backwards compatible with HDDs, in the final part of the tutorial we concentrate on how DBMS can operate on native Flash storage. We describe NoFTL, an approach that enables native Flash access and integrates parts of the Flash-management functionality into the DBMS yielding significant performance increase and simplification of the I/O stack. (v) Last but not least, having considered all these in isolation, we provide a wrap up by offering an integrated view on the big picture and focus on the necessary architectural changes.

A. Trends and Advances in Modern Storage Hardware

Modern storage technologies radically change the traditional memory hierarchy (Fig. 1). Many established facts starting from the cost ratios of typical I/O operations through the units of I/O to the access gap change. The characteristics of the new I/O technologies differ significantly from the properties of hard disk drives and RAM: high performance (higher than disk but not RAM); read/write asymmetry; endurance; addressing mode; energy consumption; parallelism. Detailed information for NVM is available in [5], [4], [6], for NAND Flash in [7], [8], [9].

B. Building Large Flash-based DBMS Storage

Single server-class Flash SSDs nowadays still have relatively small volumes. Building a large Flash-based storage for DBMS, while preserving the high-performance is a non-trivial task. With traditional hardware RAID technology the controller cannot handle more than a few SSDs. It is a significant bottleneck and distorts the well known rules-of-thumb: (i) the SSD read-write asymmetry and the write behavior are amplified due to RAID; (ii) scalability issues in a RAID-based storage system appear because of inadequate controllers; (iii) fragmentation and distribution issues affect performance much more than expected. Using host based storage and software RAID address the problem and yield a scalable solution.

We will revisit some of the lessons learned so far in Section III-G. Next, we focus on multi-versioning in Flash-aware DBMS, version storage, organisation and indexing.

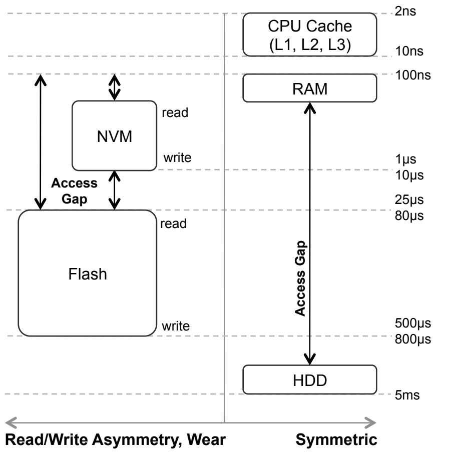

Fig. 1. Memory hierarchy with new storage technologies

C. Transactional Model

Multi-version database systems can effectively address the properties of new storage technologies. Although MV-DBMS become a dominating trend, they still contain design assumptions based on the characteristics of legacy hardware: in single-version DBMS a tuple is always updated in place, leading to suboptimal I/O patterns for Flash memories such as random writes and physical in-place updates. With multi-versioning every tuple update creates a new version of that tuple which is a separate physical entity that can be written to a new physical location (out-of place), avoiding random writes. Moreover, parallelism is leveraged, since with versions readers do not block writes.

We provide an analysis of existing version organisation and handling techniques in research prototypes and industry-strength systems. The analysis indicates that one of the issues with modern MV-DBMS is that they accomplish multi-versioning by timestamping versions with a creation and an invalidation timestamp to verify visibility to a transaction. A transaction creating a new version, changes the invalidation timestamp of the old version (predecessor) to invalidate it, which results in random writes and in-place updates. We depict the version invalidation scheme in Figure 2.

In addition, a novel *invalidation model* and version handling paradigm are presented, both of which are at the core of a newly proposed approach called Snapshot Isolation Append Storage (SIAS). Under SIAS the very presence of a successor version implicitly invalidates the predecessor. SIAS addresses tuple versions that belong to the same data item as a unique structure, whereas Snapshot Isolation (SI) addresses each tuple version individually. Conversely, traditional SI treats the visibility check for each tuple version individually as a local decision, whereas SIAS treats all tuple versions that belong to the same data item as a whole. Furthermore, SIAS unifies the concepts of: multi-versioning and multi-version concurrency control; append storage in tuple version granularity (Section III-F); multi-version indexing (Section III-D); simplified buffer management (Section III-E); and entails further optimizations

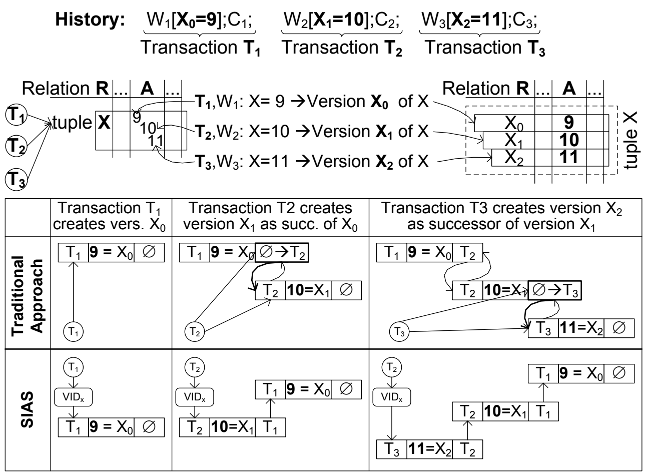
History: $W_1[X_0=9];C_1;$ $W_2[X_1=10];C_2;$ $W_3[X_2=11];C_3;$
Transaction T_1
Transaction T_2
Transaction T_3
Relation R
tuple X
T_1,W_1: X= 9 → Version X_0 of X
T_2,W_2: X=10 → Version X_1 of X
T_3,W_3: X=11 → Version X_2 of X
Transaction T_1 creates vers. X_0
Transaction T2 creates version X_1 as succ. of X_0
Transaction T3 creates version X_2 as successor of version X_1
Traditional Approach
SIAS
VID_x

Fig. 2. Invalidation in SI and SIAS

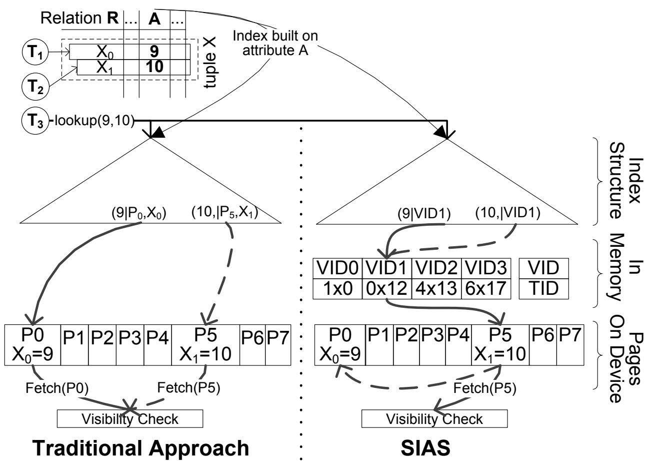

Fig. 3. Indexing in the Traditional Approach and in SIAS

to data placement. The SIAS approach alleviates invalidation related in-place updates by organising tuple versions as a simple linked list. The latest version is always known and is chained to its predecessor, demonstrated in [10] by using a bitvector to identify entrypoints. Figure 2 depicts the paradigm of the 'chained' tuple versions. Our recent SIAS approach identifies tuple versions that belong to a data item by a virtual ID (VID, Fig. 2) instead of a bitvector. A mapping table tracks the entrypoint's TID of this *version chain*, accessed by the common VID.

D. Searching and Indexing

We present an overview of how multi-version indexing is handled in existing research prototypes and industry-strength systems. The analysis shows that traditional index structures are unaware of the versioned data underneath and underlying asymmetric storage. Making index structures aware of the version organisation and invalidation model yields benefits and avoids suboptimal access patterns. In Figure 3 the traditional approach indexes two tuple versions of the same data item. Since the index structure is version-oblivious it is unaware of the fact that the two versions belong to the same data item and treats them as separate items. Hence if an index lookup request matches either one of them, both have to be fetched and subsequently checked for visibility (Fig. 3).

In a further context, we present our multi-version indexing scheme within SIAS (Fig. 3) that enables indexing of data items rather than independent physical tuple versions. Under SIAS a data item is addressed and fetched using a unique VID that is equal among all tuple versions of that data item. The most recent version (entrypoint) is always known (determined using the mapping shown in Figure 3). The records in the SIAS index store VIDs instead of the tuple IDs, as in the traditional approach. The Multi-Version Index (proposed in [11]) enables index only, in-memory version handling and visibility decisions, all of which leading to a significant reduction of the amount of I/O storage accesses and maintenance overhead. It further achieves significantly lower response times and higher transactional throughput on OLTP workloads.

E. Buffer Management

This part of the tutorial is about the buffer management in light of modern storage hardware and especially in MV-DBMS. Most buffer management strategies aim at hitrate maximization, making it the primary criterion (based on recency, frequency). New storage technologies have a significant impact on the buffer manager: due to the read/write asymmetry the cost of page eviction may be several times higher than the cost of fetching a page. Hence write-awareness and spatial locality become more important. We provide an overview of existing Flash-friendly state-of-the-art buffer management approaches. Furthermore, we show how hitrate, locality, frequency and recency are influenced by the new storage technologies and present our FBARC approach, which is an ARC based buffer management strategy designed to address I/O asymmetry on Flash devices. The SIAS approach is augmented with a simplified buffer/storage management. New tuple versions (inserts/updates) are appended to a new page at the logical end of the append storage and only if the page is completely filled or a threshold is reached (time/amount of work) it is written to stable storage.

F. Storage Management

We evaluate different approaches to append storage management (Fig. 4). Append-/Log-based Storage Managers (LbSM) for DBMS are a good match for the characteristics of new storage technologies. They alleviate random writes and immediate in-place updates, hence reducing the impact of Flash read/write asymmetry. Nevertheless they introduce mapping and granularity overhead, leading to write amplification.

We present the SIAS append storage manager that couples multi-versioning, physical out-of-place updates and append-based storage to address the read/write asymmetry of modern hardware [12]. This significantly reduces the write-overhead that is incurred by versioning (invalidation management), since pages containing invalidated versions do not have to be overwritten or remapped. Tuple versions are appended to pages and a page is written when it is completely filled, leading to a sequentialization of writes (Fig. 4: C1,C2). We discuss

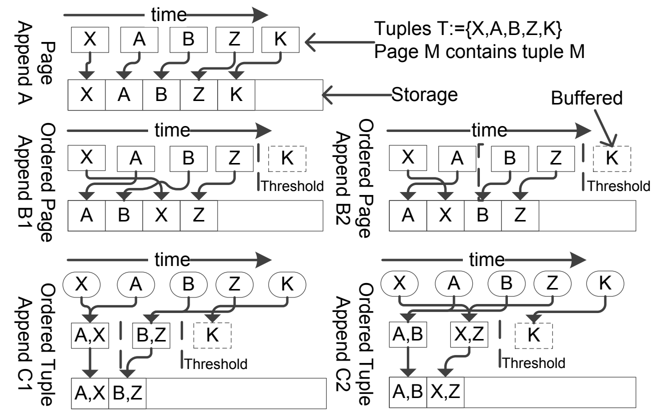

Fig. 4. Append Storage Management

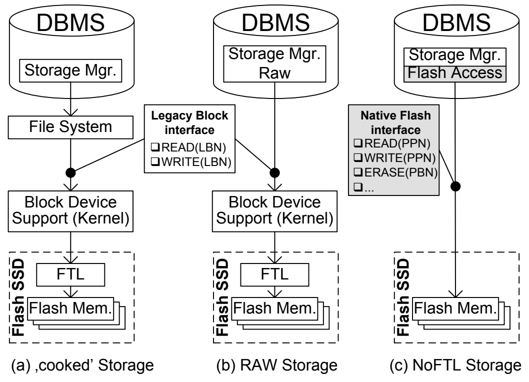

Fig. 5. DBMS storage alternatives: (a) Traditional cooked DBMS storage; (b) DBMS on RAW volumes/devices; (c) NoFTL

optimizations to append storage such as writing in sorted runs that optimize for successive read access (Fig. 4: B1,B2,C1,C2).

G. Revisiting backwards-compatible DBMS Flash Storage

Some of the lessons learnt in terms of the disadvantages of using modern Flash SSDs as DBMS storage are already introduced in Sections III-A and III-B. These include: (i) limited SSD's on-device computational and memory resources; (ii) single FTL scheme for all types of workloads; (iii) closed black-box architecture, where neither the DBMS information about data and I/O; nor (iv) the knowledge about the physical Flash layout can be utilized; (v) unpredictable performance fluctuations; (vi) legacy interfaces and protocols; (vii) functional redundancy along the I/O path; (viii) 3-5x higher costs of Flash storage.

In this context we revisit many of the assumptions stated so far and describe the NoFTL architecture [13], under which the DBMS has a complete control over the underlying Flash memory by utilizing a native Flash interface. It represents a logical continuation of the DBMS tradition (Fig. 5) to simplify the I/O path and directly control physical data placement (DBMS on RAW devices). Flash maintenance tasks, previously performed by the on-device FTL, are now integrated into different subsystems in the DBMS (storage, transaction or buffer managers). The unboxing of Flash and its native DBMS integration creates a win-win situation for the Flash management and the DBMS.

IV. Biographies of the Presenters

Ilia Petrov is a Professor at Reutlingen University, Germany since 2012. Prior to that he was a Post-Doctoral fellow with the Databases and Distributed Systems Group at the Technische Universität Darmstadt. His research focus is on data management on modern hardware. He worked on data management and Business Intelligence at SAP. He holds a Ph.D. from the University of Erlangen-Nürnberg.

Robert Gottstein is a fourth year Ph.D. student and scientific researcher at the Technische Universität Darmstadt. He is working under Prof. Alejandro Buchmann at the computer science department of Databases and Distributed Systems. His research focusses on DBMS on new storage technologies and he is actively involved in the FlashyDB DFG research project.

Sergej Hardock is a Ph.D. student at the Databases and Distributed Systems Group at the Technische Universität Darmstadt. His research interests are in database systems on modern hardware, native Flash database storage, lean Flash-aware database systems. He earned his master's degree from TU-Darmstadt in 2013.

Acknowledgment

This work was supported by the DFG project ”Flashy-DB” and the Graduate School ”Topology of Technology”.

References

[1] J. Gray and P. Shenoy, “Rules of thumb in data engineering,” in *Proc. ICDE'00*. Washington, DC: IEEE, 2000, pp. 3–10.
[2] A. von Bechtolsheim, “Technologies for data- intensive computing,” in *Keynote Presentation. HTPS 09*, Asilomar, CA, Oct. 2009.
[3] N. D. Mathur, “The fourth circuit element,” *Nature*, vol. 455, no. 7217, pp. E13–E13, 10 2008. [Online]. Available: http://dx.doi.org/10.1038/nature07437
[4] S. Chen, P. B. Gibbons, and S. Nath, “Rethinking database algorithms for phase change memory,” in *CDIR'11*, Asilomar, CA, Jan. 2011.
[5] G. Muller, T. Happ, M. Kund, G. Y. Lee, N. Nagel, and R. Sezi, “Status and outlook of emerging nonvolatile memory technologies,” *Electron Devices Meeting. IEDM Technical Digest. IEEE*, pp. 567 – 570, 2004.
[6] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, “Phase change memory architecture and the quest for scalability,” *Commun. ACM*, vol. 53, pp. 99–106, July 2010.
[7] F. Chen, D. A. Koufaty, and X. Zhang, “Understanding intrinsic characteristics and system implications of flash memory based solid state drives,” in *SIGMETRICS '09*, 2009, pp. 181–192.
[8] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. Manasse, and R. Panigrahy, “Design tradeoffs for SSD performance,” in *USENIX'08*, 2008, pp. 57–70.
[9] T.-S. Chung, D.-J. Park, S. Park, D.-H. Lee, S.-W. Lee, and H.-J. Song, “A survey of flash translation layer,” *J. Syst. Archit.*, vol. 55, pp. 332–343, May 2009.
[10] R. Gottstein, T. Peter, I. Petrov, and A. P. Buchmann, “SIAS-V in Action: Snapshot Isolation Append Storage - Vectors on Flash,” in *EDBT'14*.
[11] R. Gottstein, R. Goyal, S. Hardock, I. Petrov, and A. Buchmann, “MV-IDX: Indexing in Multi-Version Databases,” in *IDEAS'14*, pp. 142–148.
[12] R. Gottstein, I. Petrov, and A. Buchmann, “Append storage in multi-version databases on Flash,” in *BNCOD*. Springer Berlin Heidelberg, 2013, pp. 62–76.
[13] S. Hardock, I. Petrov, R. Gottstein, and A. Buchmann, “Noftl: Database systems on ftl-less flash storage,” *Proc. VLDB Endow.*, vol. 6, no. 12, 2013.