NoFTL for Real: Databases on Real Native Flash Storage

Sergey Hardock #1, Ilia Petrov #2, Robert Gottstein #1, Alejandro Buchmann #1
#1 *Databases and Distributed Systems Group TU-Darmstadt,* #2 *Data Management Lab, Reutlingen University*
#1`{hardock | gottstein | buchmann}@dvs.tu-darmstadt.de`, #2`ilia.petrov@reutlingen-university.de`

ABSTRACT

Flash SSDs are omnipresent as database storage. HDD replacement is seamless since Flash SSDs implement the same legacy hardware and software interfaces to enable backward compatibility. Yet, the price paid is high as backward compatibility masks the native behaviour, incurs significant complexity and decreases I/O performance, making it non-robust and unpredictable. Flash SSDs are black-boxes. Although DBMS have ample mechanisms to control hardware directly and utilize the performance potential of Flash memory, the legacy interfaces and black-box architecture of Flash devices prevent them from doing so.

In this paper we demonstrate 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. NoFTL is implemented on real hardware based on the OpenSSD research platform. The contributions of this paper include: (i) a description of the NoFTL native Flash storage architecture; (ii) its integration in Shore-MT and (iii) performance evaluation of NoFTL on a real Flash SSD and on an on-line data-driven Flash emulator under TPC-B,C,E and H workloads. The performance evaluation results indicate an improvement of at least 2.4x on real hardware over conventional Flash storage; as well as better utilisation of native Flash parallelism.

1. INTRODUCTION

Many basic database architectural principles and algorithms have been designed around the properties of HDD. Flash memories provide a set of different I/O characteristics and promise to speedup the critical I/O path. We argue that the design of the storage architecture is not well suited for new kinds of memory in terms of both software and hardware. Flash devices still support the same block level interface as HDDs, which ensures backwards compatibility and eases replacement, but is also a major source of unpredictability and non-robustness.

The low-level block interface compatibility is realized by the *Flash Translation Layer (FTL)* that is executed inside the storage device on top of limited hardware. The FTL creates a black-box around the Flash memory, masking its performance characteristics and emulating a HDD-like behaviour [5, 3]. The FTL yields: (i) significant overhead; (ii) unpredictable and state-dependent performance due to background processes [5, 4]; (iii) inability to optimize the DBMS I/O behaviour to new kinds of storage[4, 10]; and last but not least, uncures (iv) high costs of Flash SSDs.

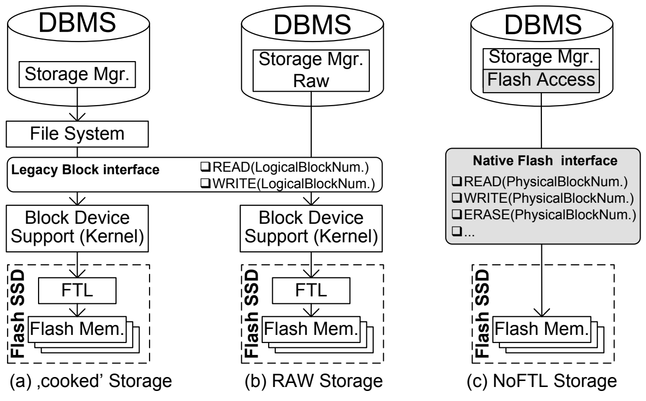


Figure 1: DBMS storage alternatives

Historically, database systems assume direct control over the hardware and the I/O stack to increase performance. Traditionally a DBMS would use a file system based ("cooked") storage on traditional block devices (Figure 1.a). Database systems on raw storage (Figure 1.b) eliminate file system overhead, enable raw storage access and direct physical data placement, achieving better performance [14]. Newer approaches propose a departure from block device interfaces, achieving: atomic writes, computational efficiency and parallelism [15], stripped down FTL and a native interface to host [4, 10]. With *NoFTL* (Figure 1.c) we consider native Flash access, and explore approaches to natural integration of FTL functionality in the DBMS.

Contributions. *NoFTL* removes all intermediate abstraction layers along the critical I/O path (block device interface, file system and FTL), and enables the DBMS to control the Flash memory directly. This minimizes the overhead of garbage collection (GC) and wear-leveling (WL), allowing the DBMS to efficiently utilize the Flash memory. The contributions of this paper are: (i) we implemented NoFTL on real hardware based on the OpenSSD research platform as well as on a real-time data-driven Flash emula-

10.5441/002/edbt.2015.47

tor; the latter was validated against OpenSSD and extended to support parallelism; (ii) we incorporated different FTLs (DFTL, Faster); (iii) live TPC-C, -B and -H tests under Shore-MT indicate a *NoFTL* performance improvement of 1.5x to 2.4x. NoFTL has been initially demonstrated in [8]. In contrast to [8] the current demonstration has the following improvements: (a) the real time emulator has been extended to handle parallelism (Section 3.2); (b) NoFTL has been implemented on the hardware research platform - OpenSSD (Section 3.3); (c) the database integration is deepened.

2. RELATED WORK

Numerous designs of FTLs that can be classified as Page-Block- or Hybrid-/Log-Block- Mapping FTLs have been proposed ([16], [7], [11], [12] etc.). An evaluation and comparison of different FTLs is provided in [5, 6, 13]. DFTL is introduced in [7] as a page-mapping FTL. There are multiple Flash simulation frameworks such as FlashSim [9] or DiskSim. There is ongoing research on omitting certain on-device FTL functionalities: [15] is not using the block I/O interface; [4] presents a hybrid approach which can bypass the on-device FTL. Specialized Flash Server Storage moves the FTL from a device into the driver, such as FusionIO [1]. NoFTL completely removes the on-device FTL, enabling the DBMS to take full control over the Flash device. An earlier demonstration of NoFTL is provided in [8], while here we: i) extend the emulator with parallelism; ii) port and present NoFTL on real hardware - the OpenSSD board; and iii) further integrate NoFTL into the DBMS.

3. THE NOFTL CONCEPT

Due to the black-box design of modern SSDs neither (i) the information about internal Flash architecture can be utilized by data placement algorithms in the DBMS; nor (ii) the DBMS status information about stored data and I/O (runtime & history) can be used to optimize the FTL. Furthermore, the DBMS can experience significant fluctuations in I/O latency and throughput that are state-dependent and result from expensive FTL operations (e.g., WL and GC). For instance, the average 4KB random write latency on a SLC SSD is 0.450ms, while frequent FTL-specific outliers under heavy load can reach 80ms [5]. In the same time, the efficiency of the FTL maintenance functionality is strongly coupled to limited on-device computational resources (e.g., single ASIC controller and up to 512MB of RAM).

NoFTL is in an attempt to overcome the aforementioned disadvantages. *Under NoFTL the DBMS operates directly on native Flash memory*, without intermediate layers such as file system, block-device layer or on-device FTL (Figure 1). The Flash maintenance (address translation, GC, WL, etc.) is integrated into the DBMS. Such an integration is based on the following important observation: *large parts of the FTL naturally leverage the functionality of existing DBMS modules such as the storage manager, the free space manager or the transaction manager* (Figure 2).

The general integration strategy is the optimization of Flash maintenance and DBMS algorithms based on the: (i) usage of more powerful computational and memory resources of the host system (e.g., address mapping); (ii) usage of the DBMS run-time information and knowledge about the stored data and I/O (e.g., WL & GC); (iii) elimination of redundant functionality along the I/O path (e.g., buffer management, free space management and address mapping in file system and FTL); (iv) optimisation of DBMS data placement and access algorithms based on the Flash layout (e.g., assignment of DBMS background flushers to physical address space regions).

Noteworthy is that *under NoFTL the DBMS is not confronted with the intricate low-level NAND control.* The Flash SSD is still assumed to have a thin hardware management layer (Figure 2) providing low-level NAND chip management, such as timing and synchronisation, low-level row and column address translations, channel and bus management. The functionality of the controller can optionally be implemented as a kernel driver (cheap but slow).

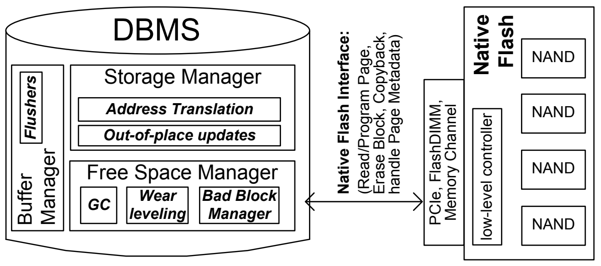


Figure 2: General Architecture of NoFTL

The minimal set of commands that the *native Flash interface* provides is: PAGE_READ and PAGE_PROGRAM with data transfer; COPYBACK_PROGRAM and BLOCK_ERASE without transfer of user data. Meaningful are also the variants of those commands to support reading or writing the series of pages (not necessary logically adjacent), which would be further translated into appropriate optimized commands according to the Flash specification (like READ/PROGRAM _CACHE_RANDOM/SEQUENTIAL in ONFI NAND). The protocol must also include the identification command (similar to HDIO_GETGEO for HDDs), which allows the DBMS to receive detailed information about the architecture of the Flash SSD (e.g., channels, LUNs, Flash type, etc.).

3.1 Host Memory Resources

The logical-to-physical address translation is the core component of an out-of-place update strategy, and is one of the most memory consuming subsystems within modern Flash SSDs. Since the amount of on-device memory is insufficient to hold a complete mapping table at page-level granularity, multiple alternatives were proposed in the recent years. Three of them, recognized as state-of-the-art, are page-level FTLs DFTL (demand-based FTL) [7] and LazyFTL [12], as well as the hybrid mapping scheme FASTer [11].

DFTL and LazyFTL keep mapping information at page-level granularity, but only a small fraction of it is cached. They introduce computational (maintenance and look-ups of cached mappings) and I/O overheads (page-ins and -outs to fetch from and store mappings on Flash). Our earlier results [8] indicate a performance slowdown of DFTL over pure page-level mapping (where the whole mapping table is cached) of up to 3.7x under TPC-C and -B benchmarks. In FASTer the larger part of Flash memory is mapped at block-level granularity (data block area), while only a small part (log block area) uses page-level mappings. All updates and write requests are first performed in the log block area, and as soon as free space in that region runs out those updates are merged with the corresponding blocks in the data block area. Merges result in expensive additional background I/Os (Figure 3). For TPC-B, -C and -E the garbage collection overhead in FASTer is almost twice as high as in NoFTL (Figure 3). This overhead negatively influences the foreground performance. Solely the use of DBMS-integrated page-level mapping in NoFTL results in 2.4x and 2.25x improvement in transactional throughput (TPS) for TPC-C and -B, respectively. The high write amplification in FASTer significantly reduces the longevity of the Flash SSD.

IO type	TPC-C SF=30		TPC-B SF=350		TPC-E 1K Customers	
	Absolute	Relative	Absolute	Relative	Absolute	Relative
COPYBACK	16 465 930	1.98x	17 295 713	2.15x	1 805 540	1.97x
ERASE	129 317	1.73x	135 839	1.82x	14 231	1.68x
Off-line trace-driven testing. Traces were recorded on in-memory database running the benchmarks for 60 minutes.						

Figure 3: Absolute and relative I/O overhead of garbage collection under FASTer and NoFTL.

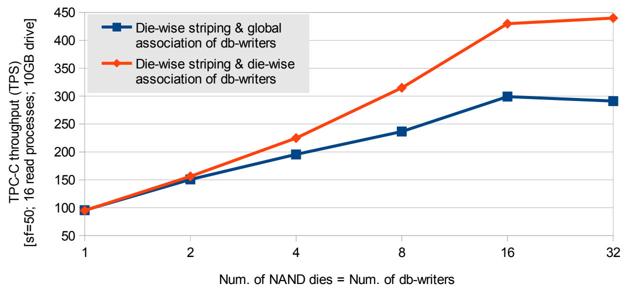
3.2 Utilization of Flash parallelism

Many DBMS algorithms can be optimized based on the architecture of underlying Flash SSDs. Direct control over physical data placement allows to efficiently utilize native Flash parallelism. For instance, SATA2 allows for at most 32 concurrent I/O commands; whereas a commodity Flash SSD with 8 to 10 chips is able to execute up to 160 concurrent I/Os (8-16 commands/chip). To make better use of the available Flash parallelism, we have incorporated the knowledge about Flash architecture into the logic of database writer processes (db-writers). The basic idea is to remove the contention for physical resources among db-writers. Instead of having multiple db-writers, where each is responsible for a subset of dirty pages from the whole address space, we have assigned each db-writer to a certain physical region (i.e., set of NAND chips). Therefore, each db-writer receives a distinct subset of dirty pages that belongs to a corresponding physical address space, and does not compete for physical storage with db-writers assigned to other regions. Depending on the workload and the size of the regions it is also possible to assign several db-writers to a single region. We have implemented this optimization in Shore-MT. We can demonstrate an improvement of throughput over the initial implementation with the same number of background writer processes of up to 1.5x for TPC-C and 1.43x for TPC-B benchmarks (see Figure 4). With an increasing amount of Flash parallelism and more db-writers (leveraging the parallelism), the difference in the transactional throughput increases. In the standard approach with a global assignment the response time for each single db-writer increases, due to the higher contention for Flash chips.

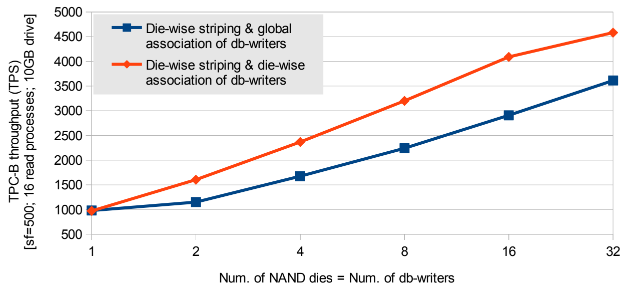
3.3 NoFTL Testbed

We have implemented the NoFTL concept in Shore-MT [2], which is a recognised open-source storage engine supporting ACID transactions, ARIES-type logging, Indices, Buffer management. Furthermore, Shore-MT supports raw devices and standard TPC benchmark implementations. The NoFTL-version of the storage engine was evaluated on the real hardware OpenSSD board, as well as on our enhanced version of the real-time Flash emulator.

OpenSSD board. The OpenSSD project aims to provide an open hardware Flash research platform (see Figure 5). It allows to program the firmware running on the on-



(a) TPC-C



(b) TPC-B

Figure 4: Tx. throughput of TPC-C/-B with global and Flash-aware assignment of db-writers.

device controller to a certain extent, and to test different FTL schemes and algorithms. The board contains a set of operational FTLs, among them the popular FASTer scheme [11]. To make OpenSSD perform as a native Flash board we have removed the FTLs and modified the I/O protocol to support the native Flash (ATA Pass-Through, Section 3).

Real-time Flash emulator. We have enhanced our real-time Flash emulator [8] to support complex highly parallel Flash architectures. The emulator is implemented as a device driver in the Linux kernel. The usage of low-level kernel primitives guarantees high precision ($\sim 1\mu s$) in simulation of I/O latencies. There is no loss of accuracy with increasing capacity of the emulated drive, however, the latter is limited by the available RAM resources of the host system. The emulated Flash storage provides either a block-device interface, emulating a SSD or a character device interface, emulating native Flash. The emulator allows to investigate parallelism, different Flash layouts or NAND types (SLC,MLC,TLC) as well as characteristics such as wear, which is not possible with OpenSSD. The emulator's behavior and characteristics have been validated against the OpenSSD platform.

4. DEMONSTRATION

The demonstration is performed based on two platforms: the OpenSSD hardware research platform and a real time Flash emulator. We compare NoFTL against the conventional DBMS storage, based on black-box Flash SSDs (Figure 1.a, 1.b). For the latter scenario (Figure 6.a) we choose two state-of-the-art FTL as counterparts: (i) DFTL [7] (page-level mapping); and (ii) FASTer [11] (hybrid mapping). All demonstration scenarios are performed live either on real hardware (OpenSSD board), or on the Flash emulator.

Demo Scenario 1 - Validation of Flash emulator.

In this scenario we stress the emulator with the Linux FIO tool to showcase: (1) Its accuracy and reconfigurability, i.e., test different internal architectures of Flash memory on synthetic benchmarks; and (2) Investigate the DBMS

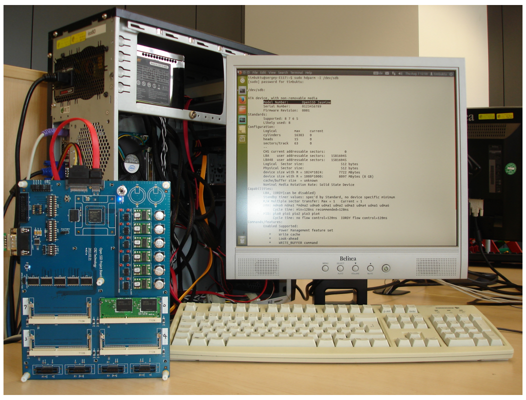


Figure 5: OpenSSD connected to the test-bed.

utilisation of richer parallelism under NoFTL to improve transactional throughput. We also perform the live validation of the Flash emulator against the OpenSSD hardware by configuring the former with the properties and architecture of latter and comparing the performance results of live TPC benchmarks. For each demonstration run the audience is presented the resulting diagrams for transactional throughput and response time as well as statistics regarding erasures, writes and garbage collection activity.

Demo Scenario 2 - DBMS performance under NoFTL.

The general architecture of the NoFTL demonstration testbed is depicted in Figure 6. During the demonstration the audience can select any of the TPC benchmarks (-H, -B, -C or -E) and a demonstration platform (OpenSSD or Flash emulator). Furthermore, the audience can configure the Flash layout as well as the number of DBMS flushers to experience the influence of the different strategies. With an increasing amount of Flash parallelism and more db-writers (leveraging the parallelism), the difference in the transactional throughput increases. Test results comprise Shore-MT's output, intermediate and average transactional throughput, as well as detailed statistics of I/O operations and GC overhead. Furthermore, we demonstrate the influence of the improved ("Flash-aware") allocation and assignment strategy of background writers in Shore-MT (Sect. 3.2).

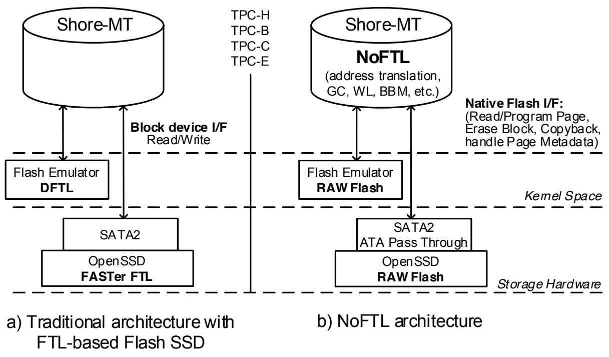


Figure 6: Demonstration scenarios.

5. CONCLUSIONS

We demonstrate *NoFTL* - an approach that yields a significant simplification of the I/O stack; integrates Flash management in the DBMS; allows direct access to storage and exposure of a native Flash interface. *NoFTL* is implemented in Shore-MT, on top of the OpenSSD hardware research platform and a real-time data-driven Flash emulator. We validate live the real-time Emulator, and are able to showcase different Flash layouts throughout the demonstration. Under *NoFTL* Flash management algorithms can benefit from the richer resources of the host system. We demonstrate stable and predictable performance and an improvement of up to 2.4x under TPC-C. The speedup results from a reduced garbage collection overhead (2x less erases and copybacks) due to better database integration of FTL functionality. Furthermore, the low erase count under NoFTL effectively doubles the lifetime of the Flash storage. In addition, we demonstrate the utilisation of native Flash parallelism under *NoFTL*. With its Flash-aware DBMS writer assignment strategy NoFTL achieves 1.5x higher transaction throughput due to reduced Flash chip contention.

Acknowledgements. This work was supported by the DFG projects "Flashy-DB" and "Topology of Technology".

6. REFERENCES

[1] Going beyond ssd: The fusionio software defined flash memory approach, 2013.
[2] http://diaswww.epfl.ch/shore-mt/, 2013.
[3] N. Agrawal and e. A. Prabhakaran. Design tradeoffs for ssd performance. In *Proc. ATC*, pages 57–70, 2008.
[4] P. Bonnet, L. Bouganim, I. Koltsidas, and S. D. Viglas. System co-design and data management for flash devices. In *Proc. VLDB 2011*, 2011.
[5] F. Chen, D. A. Koufaty, and X. Zhang. Understanding intrinsic characteristics and system implications of flash memory based solid state drives. In *Proc. SIGMETRICS'09*, 2009.
[6] 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.*, 55(5,):332 – 343, 2009.
[7] A. Gupta, Y. Kim, and B. Urgaonkar. Dftl: A flash translation layer employing demand-based selective caching of page-level address mappings. In *Proc. ASPLOS XIV*, pages 229–240, 2009.
[8] S. Hardock, I. Petrov, R. Gottstein, and A. Buchmann. Noftl: Database systems on ftl-less flash storage. *Proc. VLDB Endow.*, 6(12), 2013.
[9] Y. Kim, B. Tauras, A. Gupta, and B. Urgaonkar. Flashsim: A simulator for nand flash-based solid-state drives. In *In Proc. SIMUL'09*, pages 125–131, 2009.
[10] I. Koltsidas and S. D. Viglas. Data management over flash memory. In *Proc.SIGMOD '11*, 2011.
[11] S.-P. Lim, S.-W. Lee, and B. Moon. Faster ftl for enterprise-class flash memory ssds. In *Proc. SNAPI'10.*
[12] D. Ma, J. Feng, and G. Li. Lazyftl: A page-level flash translation layer optimized for nand flash memory. In *Proc. SIGMOD '11*, pages 1–12, 2011.
[13] D. Ma, J. Feng, and G. Li. A survey of address translation technologies for flash memories. *ACM Comput. Surv.*, 46(3):1–39, 2014.
[14] Oracle. A quantitative comparison between raw devices and file systems for implementing oracle databases. white paper. 2004.
[15] X. Ouyang, D. Nellans, R. Wipfel, D. Flynn, and D. K. Panda. Beyond block i/o: Rethinking traditional storage primitives. In *Proc. HPCA*, 2011.
[16] C. Park, W. Cheon, J. Kang, K. Roh, W. Cho, and J.-S. Kim. A reconfigurable ftl architecture for nand flash-based applications. *TECS*, 7(4):38:1–38:23, 2008.