NoFTL: Database Systems on FTL-less Flash Storage (Extended Abstract)

Sergey Hardock
Databases and Distributed Systems Group
TU-Darmstadt, Germany
hardock@dvs.tu-darmstadt.de

Ilia Petrov
Data Management Lab
Reutlingen University, Germany
ilia.petrov@reutlingen-university.de

Robert Gottstein
Databases and Distributed Systems Group
TU-Darmstadt, Germany
gottstein@dvs.tu-darmstadt.de

Alejandro Buchmann
Databases and Distributed Systems Group
TU-Darmstadt, Germany
buchmann@dvs.tu-darmstadt.de

ABSTRACT

The database architecture and workhorse algorithms have been designed to compensate for hard disk properties. The I/O characteristics of Flash memories have significant impact on database systems and many algorithms and approaches taking advantage of those have been proposed recently. Nonetheless on system level Flash storage devices are still treated as HDD compatible block devices, black boxes and fast HDD replacements. This backwards compatibility (both software and hardware) masks the native behaviour, incurs significant complexity and decreases I/O performance, making it non-robust and unpredictable. Database systems have a long tradition of operating directly on RAW storage natively, utilising the physical characteristics of storage media to improve performance.

In this paper we demonstrate an approach called *NoFTL* that goes a step further. We show that allowing for native Flash access and integrating parts of the FTL functionality into the database system yields significant performance increase and simplification of the I/O stack. We created a real-time data-driven Flash emulator and integrated it accordingly into Shore-MT. We demonstrate a performance improvement of up to 3.7x compared to Shore-MT on RAW block-device Flash storage under various TPC workloads.

1. INTRODUCTION

Many key database architectural principles and workhorse algorithms have been designed to leverage the properties of HDD. Flash memories are a new technology crucial to database systems, which comes with a set of different I/O characteristics. A large body of algorithmic approaches has been proposed over the last years to natively address Flash properties. Nonetheless on system level, Flash devices still support the same block level interface as HDD. On the one hand, the block device compatibility favours adoption by making replacement seamless. On the other hand, as a legacy interface, it is a major source of unpredictability, non-robustness. The negative performance impact ultimately precludes any Flash relevant optimisations.

The *Flash Translation Layer (FTL)* is an on-device layer that ensures low-level block interface compatibility, masking physical characteristics, and making a Flash device behave like a hard drive [6, 4]. Some of the negative FTL aspects are: (i) Unpredictable and state-dependent performance due to background processes [6, 5]; (ii) adverse performance impact due to limited on-device computational resources [5, 13]; (iii) redundant functionality also present at different layers along the critical I/O path [5, 13].

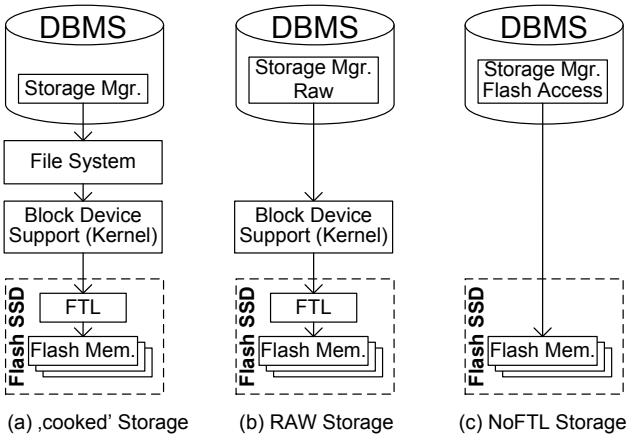

Figure 1: DBMS storage alternatives: (a) Traditional 'cooked' DBMS storage; (b) RAW DBMS storage; (c) NoFTL

Database systems have a long history of simplifying the I/O stack to increase performance. Traditional setups would employ 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 [18]. Newer approaches propose departing from block device interfaces, achieving: atomic writes, computational efficiency and parallelism [20], stripped down FTL and a native interface to host [5, 13]. With *NoFTL* (Figure 1.c) we stop treating Flash devices as a closed system, consider native Flash access, and explore FTL integration in the DBMS.

Contributions and Demo. In this paper we demonstrate an approach called *NoFTL*. We argue for a significant simplification of the I/O stack; integrating Flash management in the database and using DBMS knowledge to control storage; direct access and exposure of a native Flash interface; utilisation of database server's computational resources instead of on-device resources. The contributions of this paper are: (i) we implemented a real-time data-driven Flash emulator as a character device driver; (ii) we incorporated a DFTL implementation; (iii) we extended Shore-MT with a page mapping FTL and integrated the real-time simulator; (iv) live TPC-C, TPC-B and TPC-H tests under Shore-MT indicate a *NoFTL* performance improvement of 1.5x to 3.7x over the DFTL configuration.

2. RELATED WORK

In the past numerous designs of FTLs have been proposed (e.g. [21], [11], [15], [14], [16], [17] etc.). Such approaches can be classified as Page-, Block- or Hybrid-/Log-Block- Mapping FTLs. An evaluation and comparison of different FTLs is provided in [7] and [8]. DFTL is a page-mapping FTL and is introduced in [11]. There are multiple Flash simulation frameworks such as FlashSim [12] or DiskSim. There is further research on omitting certain on-device FTL functionalities, e.g. an approach that is not using the block I/O interface is presented in [20], [5] 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 application to take full control of the Flash storage device.

3. THE NOFTL APPROACH

At the core of the *NoFTL* evaluation and demonstration (Figure. 2) is a real-time data-driven Flash emulator, simulating a Flash device according to the ONFI standard, while storing the data in a large RAM buffer. The emulated Flash device is attached to Shore-MT [3], which is a recognised storage engine supporting ACID transactions, ARIES-type logging, Indices, Buffer management. Furthermore, Shore-MT supports raw devices and standard TPC benchmark implementations.

The *NoFTL* emulator exposes two interfaces: a native Flash interface (Figure 2.a) and a block device interface (Figure 2.b). In the former case (*NoFTL*) we also extended Shore-MT with a Page-Mapping FTL, including a DatabasePage-to-PhyiscalPage mapping, integrating Flash space management and wear levelling in the Shore-MT storage manager. The latter case required an FTL implementation to ensure

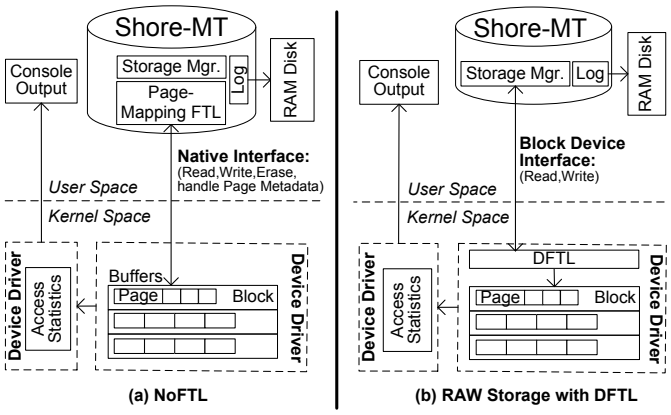

Figure 2: Architecture

block device functionality. DFTL [11] was chosen since it provides better results than most hybrid FTL schemes proposed in the recent years. Both alternatives are implemented as device drivers. We gather reliable device statistics not interfering with the real-time simulator by using an additional device driver, in both configurations.

The Flash device emulation is based on the observation that typical Flash latencies (of 25μs and 250μs write of a 4KB page and 700μs for a block erase – Micron MT29F16G08ABABA) provide enough CPU time to perform the emulation. The device size and layout are configurable and designed according to the ONFI specification: a Flash device contains a number of chips comprising multiple planes (between 1 and 4), which in turn comprise multiple dies (typically 4). The current *NoFTL* emulator architecture and implementation allows for very accurate timings and robust performance. We validated the *NoFTL* emulator under different workloads using I/O benchmarking tools(*FIO*) and against offline Flash Simulators (FlashSim, DiskSim).

Coupling the emulator to Shore-MT is the second cornerstone of the demonstrated approach. For *NoFTL* (Figure 2.a) to work, typical FTL functionality was integrated into Shore-MT: (i) page-mapping, wear levelling, and garbage collection; (ii) integration of the native Flash interface (native use of *read, write and erase* at the respective granularity); (iii) eliminating redundant functionality.

The experimental analysis was performed on an Intel Xeon server with two quad-core Intel Xeon 5630 2.5 GHz processors (256 KB L1 cache, 1 MB L2 cache and 12 MB L3 cache) and 48 GB RAM and a QPI bus architecture. We instrumented Shore-MT for both TPC-C (scale factors: 5, 25 and 50), TPC-B (scale factors: 48, 200, 500) and TPC-H (scale factor: 1) on different Flash disk sizes (1 GB, 5GB and 10GB). We compare the *NoFTL* scenario (Figure 2.a) against the RAW Flash with *DFTL* scenario (Figure 2.b). DFTL was configured with different mapping cache sizes (2%, 15%, 35% of all mappings cached).

The experimental results are shown in Figure 3. *NoFTL* is 1.5x to 3.7x faster than comparable *DFTL* configurations. The speedup increases with increasing volume sizes. Three major factors contribute to the speedup: (a) limited on-device computational resources; (b) FTL restrictions; (c) redundant functionality.

Limited on-device computational resources: DFTL and the FTL in general is executed on slow on-device hardware. In the emulator we account for the physical I/O required to

TPC-C: SF=5 TPC-B: SF=48 DISK: 1GB	NoFTL		DFTL 2%			DFTL 15%			DFTL 35%		
	Tx/s	STDEV	Tx/s	STDEV	Slow-down	Tx/s	STDEV	Slow-down	Tx/s	STDEV	Slow-down
TPC-C	43.34	0.34	26.64	0.35	1.63	25.40	0.34	1.71	30.00	0.87	1.44
TPC-B	462	1.05	275.7	3.70	1.67	254	2.84	1.82	295.9	2.26	1.56

TPC-C: SF=25 TPC-B:SF=200 DISK: 5GB	NoFTL		DFTL 2%			DFTL 15%			DFTL 35%		
	Tx/s	STDEV	Tx/s	STDEV	Slow-down	Tx/s	STDEV	Slow-down	Tx/s	STDEV	Slow-down
TPC-C	41.4	0.22	22.7	0.15	1.82	21.5	0.09	1.92	19.0	0.17	2.18
TPC-B	449.7	7.87	267.2	6.70	1.68	233.0	2.46	1.93	149.4	10.4	3.01

TPC-C: SF=50 TPC-B:SF=500 DISK: 10GB	NoFTL		DFTL 2%			DFTL 15%			DFTL 35%		
	Tx/s	STDEV	Tx/s	STDEV	Slow-down	Tx/s	STDEV	Slow-down	Tx/s	STDEV	Slow-down
TPC-C	41.2	0.38	19.1	0.31	2.16	13.8	0.13	2.98	10.9	0.42	3.77
TPC-B	409.7	3.01	215.5	0.94	1.90	163.3	15.0	2.51	113.7	9.97	3.60

TPC-H: SF=1 DISK: 1.5GB	NoFTL		DFTL 2%			DFTL 15%			DFTL 35%		
	Sec.	STDEV	Sec.	STDEV	Slow-down	Sec.	STDEV	Slow-down	Sec.	STDEV	Slow-down
Query 1	58.7	0.54	79.1	0.05	1.35	121.9	0.27	2.08	178.2	3.75	3.04
Query 6	58.4	0.42	75.2	0.05	1.29	98.6	0.11	1.69	149.5	3.77	2.56
Query 12	109.3	11.36	120.5	1.80	1.10	157.1	5.94	1.44	208.0	5.56	1.90
Query 14	98.4	2.72	113.5	0.63	1.15	142.2	2.9	1.45	194.7	3.05	1.98

Figure 3: TPC-B, TPC-C and TPC-H results NoFTL and DFTL

page in and page out the physical-to-logical address mapping table as well as the computational overhead. On real devices the less powerful on-device CPU will have an even more negative impact. In *NoFTL* scenarios the FTL fully benefits from the DB server's ample computational resources.

FTL restrictions: SSD vendors offer sparse details about the implemented algorithms. Research findings converge towards hybrid FTL schemes. We opted for DFTL (a page-mapping approach), which represents an optimistic choice: DFTL wastes less paging I/Os for the mapping tabs and has more efficient garbage collection than Hybrid FTL schemes. Nonetheless, for large mapping buffers DFTL incurs high computational overhead due to mapping table cache maintenance, clearly visible for large data (5 or 10GB and 35% cached mappings) – Figure 3.

Redundant functionality: in terms of address mapping, space and invalidation management, page and block placement database storage managers, file systems and FTL schemes contain similar functionality. Eliminating some layers of abstraction and integrating functionality is where *NoFTL* has most potential. Due to the Shore-MT integration we managed to: (i) reduce the number of block erases; (ii) couple garbage collection to Shore-MT space management; (iii) simplify wear-levelling with DB information about dirty block eviction. Possible further extensions result from integration with MVCC and Log-based Storage Managers [9], access paths [10], buffer management and eviction strategies [19], etc.

4. DEMONSTRATION DESCRIPTION

In this section we describe the demonstration of the main features of *NoFTL*. We also describe the main scenarios and how the audience can interact with the system.

High-level Description. The main scenario involves instrumentation and comparative testing of NoFTL versus DFTL devices. We introduce the audience to the system and explain what the expected influence of the different knobs is. We than let the audience pick a test scenario configuration and perform comparative benchmarking.

Entry-level I/O scenario. We let the audience stress the NoFTL emulator with a simple I/O benchmarking tool - *FIO*. The audience will experience the different statistics for different metrics e.g. reads, writes, overwrites, IOPS, etc. and the influence of different system parameters on the performance (see Figure 4).

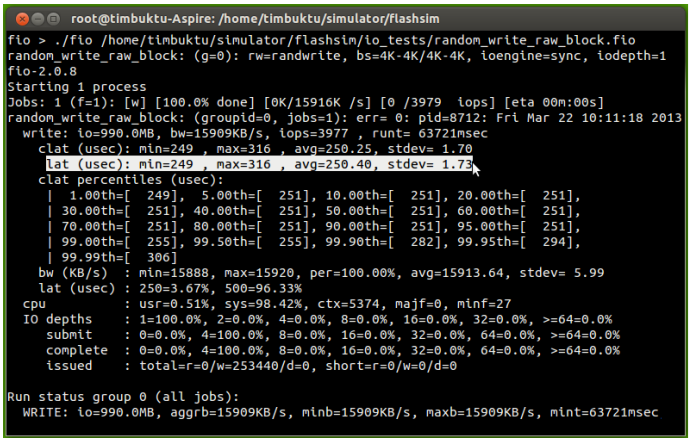

Figure 4: Screenshot testrun FIO on DFTL

Shore-MT scenarios. We than let the audience to pick a TPC benchmark (TPC-B, TPC-C or TPC-H) and dataset size depending to the personal preference.

Phase I. A device layout is selected and the device is being initialised (Figure 5). In addition, the benchmark data set is loaded and the system is prepared for benchmarking.

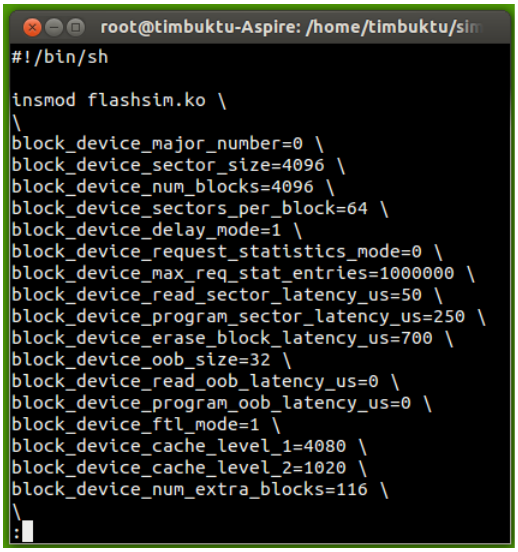

Figure 5: Device Layout Instrumentation

Phase II. Having loaded the data into Shore-MT, the chosen TPC benchmark is run under the NoFTL and DFTL setups for 10 min. The audience tracks the execution progress both in terms of Shore-MT performance statistics (Figure 6) but also in terms of low-level device and request statistics (Figure 7). A preview of this stage is provided in [2].

5. CONCLUSIONS

In this paper we demonstrated an approach called *NoFTL*. We argue for a significant simplification of the I/O stack;

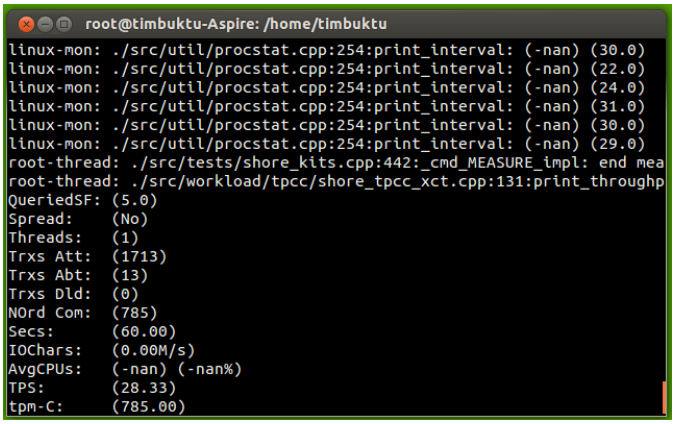

Figure 6: Screenshot testrun Shore-MT on DFTL

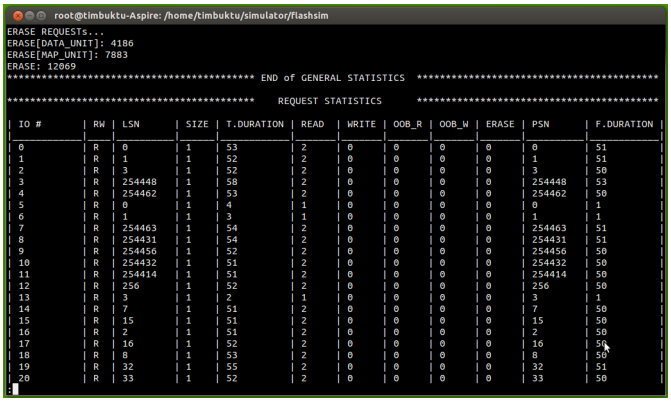

Figure 7: Screenshot NoFTL Request Statistics

integrating Flash management in the database and using DBMS knowledge to control storage; direct access to storage and exposure of native Flash interface. We also argue that the performance gain can be maximised by even closer DBMS integration. *NoFTL* is implemented as a real-time data-driven Flash emulator. We integrated it into Shore-MT, which was also extended with a page mapping FTL. As a comparative FTL-system, we incorporated a DFTL implementation. We demonstrate live TPC tests under Shore-MT indicating a *NoFTL* performance improvement of up to 3.7x.

Acknowledgements

We wish to thank Goetz Graefe for his helpful comments and suggestions on earlier drafts of this paper. This work was supported by the DFG (Deutsche Forschungsgemeinschaft) project “Flashy-DB”.

6. REFERENCES

[1] Going beyond ssd: The fusionio software defined flash memory approach, 2013. `www.fusionio.com/white-papers/beyond-ssd/`.

[2] Noftl demo preview. `http://dblab.reutlingen-university.de/tl_files/downloads/demo_noftl.mp4`, 2013.

[3] Shoremt storage engine. `http://diaswww.epfl.ch/shore-mt/`, 2013.

[4] N. Agrawal and e. A. Prabhakaran. Design tradeoffs for ssd performance. In *Proc. ATC'08*, pages 57–70, 2008.

[5] P. Bonnet, L. Bouganim, I. Koltsidas, and S. D. Viglas. System co-design and data management for flash devices. In *Proc. VLDB 2011*, 2011.

[6] 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*, pages 181–192, 2009.

[7] 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*, pages 181–192, 2009.

[8] 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(56):332 – 343, 2009.

[9] R. Gottstein, I. Petrov, and A. Buchmann. Append storage in multi-version databases on flash. In *Proc. of BNCOD*, 2013.

[10] G. Graefe. Write-optimized b-trees. In *Proc. VLDB'04*, pages 672–683, 2004.

[11] 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*, ASPLOS XIV, pages 229–240, 2009.

[12] 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.

[13] I. Koltsidas and S. D. Viglas. Data management over flash memory. In *Proc.SIGMOD '11*, 2011.

[14] S.-W. Lee, W.-K. Choi, and D.-J. Park. Fast: An efficient flash translation layer for flash memory. In X. e. a. Zhou, editor, *Emerging Directions in Embedded and Ubiquitous Computing*, Lecture Notes in Computer Science. 2006.

[15] S.-W. Lee, D.-J. Park, T.-S. Chung, D.-H. Lee, S. Park, and H.-J. Song. A log buffer-based flash translation layer using fully-associative sector translation. *ACM TECS*, 6(3), July 2007.

[16] S.-P. Lim, S.-W. Lee, and B. Moon. Faster ftl for enterprise-class flash memory ssds. In *SNAPI)*, 2010.

[17] D. Ma, J. Feng, and G. Li. Lazyftl: a page-level flash translation layer optimized for nand flash memory. In *Proc. SIGMOD '11*, 2011.

[18] Oracle. A quantitative comparison between raw devices and file systems for implementing oracle databases. white paper. 2004. `www.oracle.com/technetwork/database/performance/twp-oracle-hp-files-130020.pdf`.

[19] Y. Ou, J. Xu, and T. Härder. Towards an efficient flash-based mid-tier cache. In *DEXA*, pages 55–70, 2012.

[20] X. Ouyang, D. W. Nellans, R. Wipfel, and D. Flynn. Beyond block i/o: Rethinking traditional storage primitives. In *HPCA*, pages 301–311, 2011.

[21] C. Park, W. Cheon, J. Kang, K. Roh, W. Cho, and J.-S. Kim. A reconfigurable ftl (flash translation layer) architecture for nand flash-based applications. *ACM Trans. Embed. Comput. Syst.*, 7(4):38:1–38:23, 2008.