LeaFTL: A Learning-based Flash Translation Layer for Solid-State Drives*

Jinghan Sun[§] Shaobo Li[§] Yunxin Sun[†] Chao Sun[‡] Dejan Vucinic[‡] Jian Huang[§]

[§]University of Illinois at Urbana-Champaign [†]ETH Zurich [‡]Western Digital Research

Abstract

In modern solid-state drives (SSDs), the indexing of flash pages is a critical component in their storage controllers. It not only affects the data access performance, but also determines the efficiency of the precious in-device DRAM resource. A variety of address mapping schemes and optimizations have been proposed. However, most of them were developed with human-driven heuristics.

In this paper, we present a learning-based flash translation layer (FTL), named LeaFTL, which learns the address mapping to tolerate dynamic data access patterns via linear regression at runtime. By grouping a large set of mapping entries into a learned segment, it significantly reduces the memory footprint of the address mapping table, which further benefits the data caching in SSD controllers. LeaFTL also employs various optimization techniques, including out-of-band metadata verification to tolerate mispredictions, optimized flash allocation, and dynamic compaction of learned index segments. We implement LeaFTL with both a validated SSD simulator and a real open-channel SSD board. Our evaluation with various storage workloads demonstrates that LeaFTL saves the memory consumption of the mapping table by 2.9× and improves the storage performance by 1.4× on average, in comparison with state-of-the-art FTL schemes.

1 Background and Motivation

Flash-based SSDs have become an indispensable part of modern storage systems, as they outperform conventional hard-disk drives (HDDs) by orders of magnitude, and their cost is close to that of HDDs. The SSD capacity continues to boost by increasing the number of flash channels and chips with the rapidly shrinking process and manufacturing technology.

The flash translation layer (FTL) is the core component of managing flash in SSDs, which handles address translation, garbage collection (GC), and wear leveling. The FTL also maintains metadata structures for different functionalities, such as address translation and valid page tracking, and caches them in the in-device DRAM (SSD DRAM) for improved performance.

Among these data structures, the address mapping table has the largest memory footprint. There are three types of address mapping table: page-level mapping, block-level mapping, and hybrid mapping. Modern SSDs usually use page-level mapping, as it offers the best performance for address translation and incurs minimal GC overhead. However, the page-level mapping scheme has the largest mapping table size, as it stores LPA-to-PPA address translation for each flash page.

The address mapping table significantly affects the performance of SSDs, as it not only determines the efficiency of indexing flash pages, but also affects the utilization of SSD DRAM. Moreover, due to the limitations of the cost and power budget in SSD controllers, it is challenging for SSD vendors to scale the in-device DRAM capacity. This challenge becomes even worse with the increasing flash memory capacity in an SSD, as larger capacity usually requires a larger address mapping table for indexing.

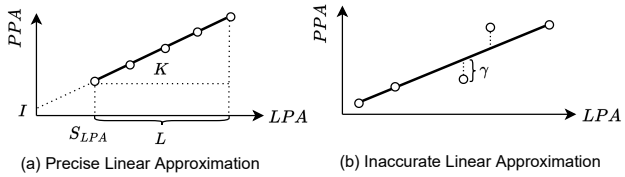

Figure 1: Visualization of learned index segments.

To improve the address mapping for SSDs, various optimization schemes have been developed. However, most of them were developed based on human-driven heuristics, and cannot capture dynamic data access patterns at runtime. In this work, we focus on utilizing simple yet effective machine learning (ML) techniques to automate the address mapping table management in the SSDs, with the capability of learning diverse and dynamic data access patterns to reduce the memory footprint and improve SSD DRAM utilization.

2 Design and Implementation

Key Ideas of LeaFTL. To reduce the large memory footprint of the page-level mapping scheme, LeaFTL uses piecewise linear regression to identify various LPA-PPA mapping patterns and group them into learned index segments (see Figure 1 (a)). Each learned index segment can be simply represented with (S_{LPA}, L, K, I), where $[S_{LPA}, S_{LPA} + L]$ denotes the interval of LPAs, K is the slope of the segment, and I is the intercept of the segment, and each of segments takes only 8 bytes (1 byte for S_{LPA} and L, 2 bytes for K, and 4 bytes for I). Compared to the page-level mapping scheme, LeaFTL reduces the mapping table size by a factor of $m * avg(L)/8$, where m is the size (usually 8 bytes) of each entry in the on-demand page-level mapping table, and $avg(L)$ is the average number of LPA-PPA mappings that can be represented in a segment. We show that $avg(L)$ is 20.3, according to our study of various storage workloads [4].

LeaFTL can relax the linear regression to capture more flash access patterns. As shown in Figure 1 (b), it can learn a mapping pattern with guaranteed error bound [-γ, γ] and create an approximate index segment. Upon creation of an approximate segment, LeaFTL uses a Conflict Resolution Buffer (CRB) to store its indexed LPAs. CRB helps LeaFTL check whether a given LPA belongs to the approximate segment.

Improving the Learning Efficiency. Flash pages are buffered in SSD controllers and flushed to the flash chips at a flash block granularity. This allows LeaFTL to learn more space-efficient index segments by reordering the flash pages in the data buffer in ascending order of their LPAs. It ensures a monotonic address mapping between LPAs and PPAs, which reduces the number of learned index segments and further saves memory space.

Managing Learned Index Segments. The direct updates to learned index segments are expensive since we have to relearn

*This work has been published at ASPLOS 2023 [4].

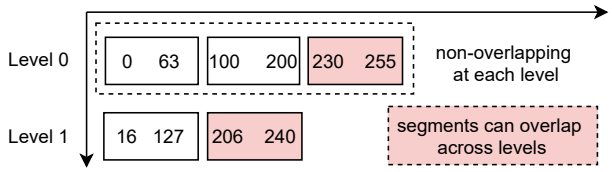

Figure 2: The learned index segments are managed in a log-structured manner in LeaFTL.

the index segments with new PPAs. This relearning procedure not only breaks the existing LPA-to-PPA mapping patterns but also involves additional flash accesses. To address this issue, LeaFTL manages the learned index segments in a log-structured manner, as shown in Figure 2. Therefore, the newly learned index segments will be appended to the log structure, while the existing learned segments can still serve address translations for LPAs whose mappings have not been updated.

LeaFTL supports the following operations: (1) *Segment Creation*: Once the data buffer is filled, LeaFTL takes the LPAs and PPAs of the flash pages in the buffer as the input, sorts the LPA-PPA mappings and uses greedy piecewise linear regression to learn the index segment; (2) *Segment Update*: LeaFTL places the newly learned segment in the topmost level of the log-structured mapping table and inserts its indexed LPAs into the CRB. If any existing segment has overlapping LPA ranges with the inserted segment, LeaFTL merges them by invalidating the outdated LPAs of the existing segment. The existing segment will be moved to the next level if the overlap still exists; (3) *LPA Lookup*: For a given LPA, LeaFTL conducts a binary search from the topmost level of the log-structured mapping table. If LPA is indexed by a learned segment, LeaFTL uses the formula $PPA = f(LPA) = \lceil K * LPA + I \rceil$, $LPA \in [S_{LPA}, S_{LPA} + L]$ to obtain the PPA. Otherwise, LeaFTL continues to search for the next level. When the segment is approximate, LeaFTL also checks the reverse mapping in the out-of-band (OOB) metadata of each flash page to check the correctness of the predicted PPA; (4) *Segment Compaction*: To save more memory space, LeaFTL periodically triggers segment compaction by merging segments with overlapping LPA ranges and removing their outdated CRB entries.

Handling Address Misprediction. LeaFTL uses the out-of-band (OOB) metadata of each flash page to verify the correctness of the translation. LeaFTL stores the reverse mappings of its neighbor PPAs in the OOB metadata. Since with a $PPA_{learned}$ obtained from an approximate segment, its error bound $[-\gamma, \gamma]$ guarantees that the correct PPA is in the range of $[PPA_{learned} - \gamma, PPA_{learned} + \gamma]$. Thus, upon a misprediction, LeaFTL will read the flash page with $PPA_{learned}$, and use its OOB to find the correct PPA with only one extra flash access.

Preserving Other Core FTL Functions. LeaFTL preserves the core functions such as GC and wear leveling. It follows the same GC and wear leveling policies in modern SSDs. When the number of free blocks in an SSD is below a threshold, the SSD controller will trigger the GC execution, which greedily selects the candidate blocks with a minimal number of valid pages and move the valid pages to the free blocks. LeaFTL updates the address mapping by placing these valid pages into the DRAM buffer, sorting them by their LPAs, and learning new index segments. LeaFTL ensures wear leveling by using the throttling and swapping mechanism developed in existing GC and updating address mappings for migrated blocks.

Key Contributions. We make the following contributions:

- We present a learning-based FTL, which can learn various data access patterns and turn them into index segments for reducing the storage cost of the mapping table.
- We develop an error-tolerant address translation mechanism to handle address mispredictions caused by the learned indexes, with minimal extra flash accesses.
- We preserve the core FTL functions and enable the coordination between the learning procedure of the address mapping table with the flash block allocation and GC to maximize the efficiency of the learned FTL.
- We manage the learned segments in an optimized log-structured manner and enable compaction to further improve the space efficiency for the address mapping.

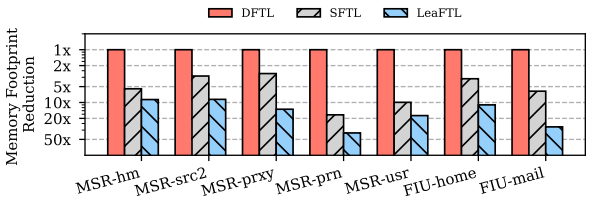

Figure 3: The reduction on the mapping table size of LeaFTL, in comparison with DFTL and SFTL.

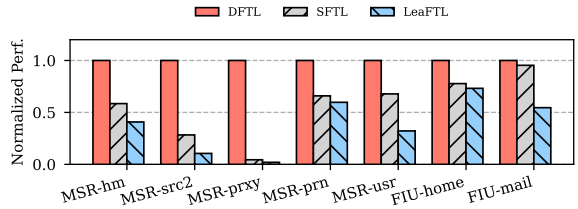

Figure 4: SSD performance when using its DRAM mainly for the address mapping table (lower is better).

LeaFTL Implementation. We implement LeaFTL based on a validated trace-driven simulator WiscSim [2] and develop a real system prototype with an open-channel SSD.

3 Evaluation

We evaluate the benefits of LeaFTL with various workloads, such as enterprise server workloads from Microsoft Research Cambridge [3], academic computer workloads from FIU [1], and data-intensive applications.

Our evaluation shows that (1) LeaFTL reduces the mapping table size by 7.5–37.7×, compared to the page-level mapping scheme DFTL (see Figure 3); (2) LeaFTL improves the storage performance by 1.4× on average, compared to SFTL (see Figure 4); (3) LeaFTL achieves additional memory savings and performance benefits with a larger error-tolerance, and it demonstrates generality for different SSD configuration. We presented the detailed evaluation in [4].

References

[1] FIU, "Fiu server traces." http://iotta.snia.org/traces/block-io/390, 2009.

[2] Jun He, Sudarsun Kannan, Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau, "The Unwritten Contract of Solid State Drives," in *the Twelfth European Conference on Computer Systems (EuroSys'17)*, April 2017.

[3] Microsoft, "Msr cambridge traces." http://iotta.snia.org/traces/block-io/388, 2007.

[4] J. Sun, S. Li, Y. Sun, C. Sun, D. Vucinic, and J. Huang, "LeaFTL: A learning-based flash translation layer for solid-state drives," in *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '23)*, ACM, Mar. 2023.