HCFTL: A Locality-Aware Page-Level Flash Translation Layer

Hao Chen[1], Cheng Li[1,2], Yubiao Pan[3], Min Lyu[1,2],Yongkun Li[1,2], Yinlong Xu[1,2]
[1]School of Computer Science and Technology, University of Science and Technology of China, Hefei, China
[2]Anhui Province Key Laboratory of High Performance Computing (USTC), Hefei, China
[3]School of Computer Science and Technology, Huaqiao University, Xiamen, China
Email: cighao@mail.ustc.edu.cn, {chengli7, ykli, lvmin05, ylxu }@ustc.edu.cn, panyubiao@hqu.edu.cn

***Abstract*—The increasing capacity of SSDs requires a large amount of built-in DRAM to hold the mapping information of logical-to-physical address translation. Due to the limited size of DRAM, existing FTL schemes selectively keep some active mapping entries in a Cached Mapping Table (CMT) in DRAM, while storing the entire mapping table on flash. To improve the CMT hit ratio with limited cache space on SSDs, in this paper, we propose a novel FTL, a hot-clusterity FTL (HCFTL) that clusters mapping entries recently evicted from the cache into *dynamic translation pages* (DTPs). Given the temporal localities that those hot entries are likely to be visited in near future, loading DTPs will increase the CMT hit ratio and thus improve the FTL performance. Furthermore, we introduce an index structure to speedup the lookup of mapping entries in DTPs. Our experiments show that HCFTL can improve the CMT hit ratio by up to 41.1% and decrease the system response time by up to 33.3%, compared to state-of-the-art FTL schemes.**

I. Introduction

Due to physical constraints, a flash page must be erased before writing new data to it. To hide the unique erase-before-write feature, SSDs implement *out-of-place* updates, each of which writes the new data to a free page and invalidates the previously-mapped page. Therefore, the flash translation layer (FTL) in an SSD must maintain a mapping table to perform the logical-to-physical address translations [1]. To achieve fast translation, SSDs usually employ a built-in DRAM to store the entire mapping table. However, with the increasing capacity of SSDs, the size of mapping table likely exceeds the capacity of the built-in DRAM.

To speed up the address translation with the limited DRAM space, DFTL [1] caches some active mapping entries in *Cached Mapping Table (CMT)*, while storing entire mapping table on flash. DFTL and its variants [2], [3] always organize the mapping entries with consecutive LPNs into a translation page, which mixes cold and hot entries in the same translation page and leads the hot mapping entries evicted from cache to be scattered across many different translation pages. However, due to temporal and spatial localities, these newly evicted entries may still have a high probability to be visited again in the near future. Upon re-visiting these entries, all entries in the corresponding translation pages will be loaded, but the majority of the entries are cold and would be discarded. As a result, these approaches suffer performance bottlenecks such as low CMT hit ratio and high response time.

To improve CMT hit ratio with the limited cache space, we propose a novel FTL, a hot-clusterity FTL (HCFTL) that clusters the newly evicted mapping entries from cache into *dynamic translation pages* (DTPs). To quickly locate a mapping entry in DTPs, we design a bloom filter-based [4] DTP index (DTPI). DTPI associates each DTP with the min and max LPNs, and a bloom filter of LPNs of its mapping entries. Furthermore, to make the DTP index highly accurate, we additionally allocate an auxiliary cache to buffer evicted mapping entries, and produce dynamic pages with minimum difference between the *max* and *min* LPNs of entries in those pages. As a summary, this work makes the following contributions: (1) We divide translation pages into two parts, dynamic translation pages (DTPs) and normal translation pages. DTPs are used to accommodate the mapping entries newly evicted from the CMT. (2) We design a DTP index (DTPI) to speed up the lookup of the entries in DTPs. Furthermore, we use a small DRAM space as the auxiliary cache as Ghost-CMT to cache the entries evicted from the original CMT to improve the efficiency of the DTPI. (3) We evaluate HCFTL with various real-world enterprise workloads. The experimental results show that, compared to the state-of-the-art FTL schemes, HCFTL improves CMT hit ratio by up to 41.1% and reduces the system response time by up to 33.3%.

II. Background and Motivation

A. FTL Schemes based on DFTL

For SSDs with large capacity and limited DRAM space, DFTL stores the entire mapping table on flash chips and only keep some active mapping entries in a Cached Mapping Table (CMT) residing in DRAM. Most recent FTL proposals [2], [3] generally adopt or extend the architecture of DFTL [1] shown in Fig. 2(a). When the cache is full, it will evict a mapping entry from cache. They always pack evicted mapping entries into translation pages in the order of their LPNs. All the mapping entries in the same translation page are logically consecutive. When a mapping entry cannot be found in cache, they read the corresponding translation page but only loads the requested mapping entry into cache, ignores and discards the remaining mapping entries in that page.

978-3-9819263-2-3/DATE19/©2019 EDAA

B. Motivation

DFTL-based FTLs always pack mapping entries with strictly successive LPNs into translation pages, which mixes cold and hot entries into the same translation pages. Thus upon reading a translation page, it is very likely that only a few entries in that page are hot. Some studies [3] highlight that in DFTL, only a small fraction (less than 15%) of the entries in a cached translation page are recently used. As shown in Fig. 1, we find that among 1024 mapping entries in a page, the number of recently used entries are usually between 100 and 200. However, the DFTL-based FTLs are unaware of this access pattern, which makes the CMT hit ratio relatively low.

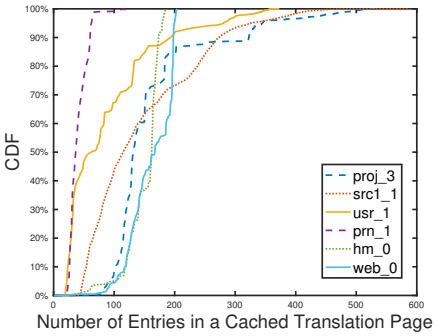


Fig. 1: Accumulative distribution of number of mapping entries in a cached translation page during running time

Limited by cost and energy consumption, it is hard to equip a large-capacity DRAM in SSDs. Moreover, the DRAM in an SSD is not only used for caching mapping table, but also used for buffering user data [5] and storing other FTL metadata [6], even fingerprints for deduplication [7], etc. Therefore, using less DRAM space for mapping table can also improve the performance of other FTL functions. So it is critical to design efficient FTLs for SSDs with limited DRAM capacity.

III. The Design of HCFTL

A. Overview of HCFTL

We aim to design a new FTL scheme, a hot-clusterity FTL that improves the hit ratio of the CMT. As shown in Fig. 2(b), compared to DFTL, we introduce into HCFTL three new components: (1) *Dynamic Translation Pages (DTPs)* are auxiliary translation pages to temporally accommodate the newly evicted entries so that hot mapping entries are very likely aggregated into the same translation pages. In addition to DTPs, we name the normal translation pages as *Static Translation Pages (STPs)*. (2) *Dynamic Translation Page Indexes (DTPIs)* are the indexes of DTPs to perform fast lookup of mapping entries in DTPs, which are stored on DRAM. (3) *Ghost-CMT* is designed as a secondary cache, which is used to caches the newly evicted mapping entries from CMT, to improve the efficiency of the DTPI.

B. Dynamic Translation Page

DTPs temporally store the mapping entries newly evicted from cache to improve the performance of subsequent reads to these entries. As shown in Fig. 3(a), DTPs also have *data area* and *out-of-band (OOB) area* like normal flash pages, but DTPs use the data area to store mapping entries, each of which has three fields, namely, *LPN*, *PPN* and *flag*, where *LPN* and *PPN* are the logical and physical page number, respectively, while the one-bit *flag* indicates if the mapping entry has been updated since it was loaded to cache. The *flag* field will be used when we merge DTPs into STPs.

Keeping too many DTPs will consume more DRAM space to store their DTPIs. More seriously, due to the false positive of bloom filters, too many DTPs will lead us to read more DTPs from flash chip to locate a target mapping entry. To eliminate this, we set a threshold to control the number of DTPs. If the number of DTPs is larger than the threshold, we will merge DTPs into STPs and free DTPs and DTPIs, which has little negative impact on the performance of SSDs because of two reasons: (1) During the working process, the DTPs can be written into flash and also can be loaded into CMT. So the number of DTPs is not always increasing. (2) The merging process will be executed in the background during the idle time of SSDs, given the fact that in the real systems, SSDs are not always busy [8].

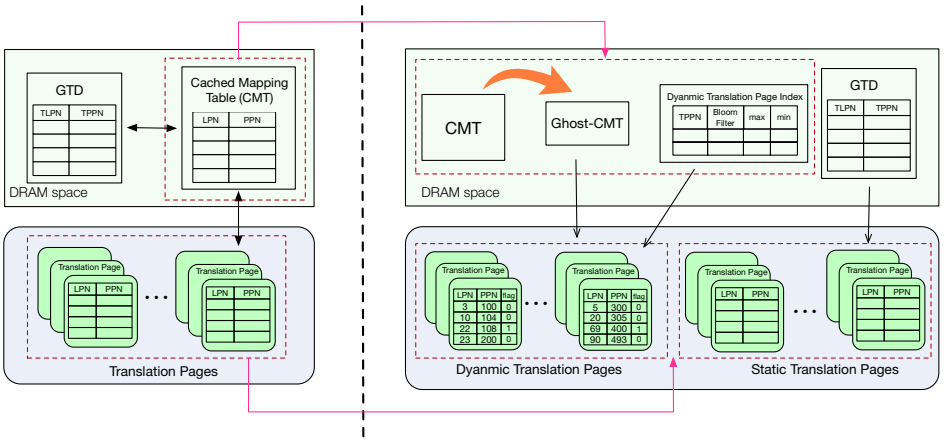
C. Dynamic Translation Page Index

To quickly locate a mapping entry in DTPs, we introduce *DTPIs* to determine if a mapping entry is in a DTP. As shown in Fig. 3(b), each DTP has a corresponding DTPI, which contains four fields, namely, *TPPN*, *bloom-filter*, *max* and *min*, where *TPPN* is the physical page number of the DTP, *max* and *min* are the maximum and minimum LPN of the mapping entries stored in the DTP respectively, and the *bloom filter* [4] is created by the LPNs of the mapping entries stored in the DTP. With *max* and *min*, we can directly conclude that a mapping entry is not in a DTP when its LPN is not in $[min..max]$ and resort to the bloom filter query to check if its LPN falls in that DTP.

We organize all DTPIs in a list and a new DTPI is always added to its tail. The recently written DTPs may have higher probability to be requested than the previously written ones due to spatial and temporal localities. Therefore, if a mapping entry is written into a DTP and requested again, it may take less time when we check the DTPIs from the tail to the head of the list when searching a mapping entry. When one of the mapping entries stored in a DTP is requested, we will load all entries in this DTP into CMT and delete its DTPI from the list, as all mapping entries in the DTP are already in cache and its DTPI is no longer needed. Accordingly, after deleting the DTPI, we invalidate the DTP, then the SSD's garbage collection process can recycle this page in the future.

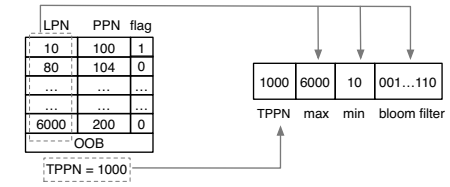
D. Ghost-CMT

To decrease the extra search delay caused by the false positive of bloom filters, we expect the difference of the *max* and *min* LPN in a DTP will not be too large, so that the number of processing bloom filter tests can not be large. So we introduce a small auxiliary cache (Ghost-CMT) to



(a) DFTL (b) HCFTL

Fig. 2: The architectures of DFTL and HCFTL



(a) dynamic translation page (b) DTPI

Fig. 3: The structures of DTP and DTPI

cache the entries evicted from CMT and then pack a fixed number of entries from Ghost-CMT to form a DTP such that $max - min$ is minimal, when Ghost-CMT is full. With minimal $max - min$, by comparing LPN with max and min, we can greatly avoid using bloom filters and consequently reduce the number of DTPs to be read due to false positive. Based on the DTPI design and the careful management of Ghost-CMT, DTPIs can be very efficient. Additionally, we set the Ghost-CMT size much smaller than CMT, so the negative impact of Ghost-CMT on the CMT hit ratio is negligible.

IV. Evaluation

A. Experimental Setup

TABLE I: SSD Parameters

SSD Capacity	128GB
# of chips	8
# of pages per block	256
Flash Page Size	4KB
Page Read Latency	25us
Page Write Latency	200us
Chip xfer Latency(per byte)	0.025us
Block Erase Latency	1.5 ms

Configurations. To evaluate the effectiveness and efficiency of HCFTL, we use a trace driven simulator, DiskSim [9] with SSD extension [10] to evaluate the performance of HCFTL. The configurations of the simulated SSDs are listed in Table I. To make a fair comparison, we respectively restrict the total DRAM space of HCFTL, DFTL and TPFTL to 2MB. For HCFTL, we set 1600 as the maximum number of DTPs, which needs 0.6MB DRAM space to store all DTPIs. Therefore, it remains 1.4MB for cached mapping table (1.26MB for CMT and 0.14MB for Ghost-CMT).

Workloads. As depicted in Table II, we use six real-world traces to evaluate the performances of different FTLs. The traces are collected on servers at Microsoft Research Cambridge [11].

TABLE II: Characteristics of Real-World Workloads

Trace	# of Req	Avg.Req. Size(KB)	Read Ratio
proj_3	2,244,644	9.75	0.95
src1_1	45,746,222	34.87	0.95
usr_1	45,283,980	49.53	0.90
prn_1	11,233,411	19.80	0.75
hm_0	3,993,316	8.89	0.35
web_0	2,029,945	15.30	0.30

B. Experimental Results

1) ***CMT Hit Ratio.*** Fig. 4 shows the CMT hit ratio with different FTL schemes and workloads. For all six workloads, HCFTL improves the CMT hit ratio by 15.5% – 800.0% and 7.8% – 41.1%, compared with DFTL and its successor TPFTL, respectively. In particular, for the *proj_3* workload, even in comparison to TPFTL, the improvement achieved by HCFTL is 41.1% and the CMT hit ratio reaches 96.10%. This is because this workload presents better spatial locality, which makes the DTPs in *proj_3* be able to hold more hot mapping entries. Consequently, loading the entries from DTPs into cache can bring better improvement on CMT hit ratio.

2) ***Response Time.*** Fig. 5 shows normalized system response time with different FTL schemes and workloads. For all six traces, in comparison to DFTL and TPFTL, HCFTL reduces the response time by from 14.0% to 40.1% and 6.1% to 33.3%, respectively. Especially, for *proj_3*, HCFTL reduces the response time by 33.3% on average, even compared with the more advanced TPFTL scheme, as it has a great improvement on CMT hit ratio (seen in Fig. 4). A high CMT hit ratio can significantly reduce the number of extra write/read operations during the address translation process, since most of the requested mapping entries are served by CMT.

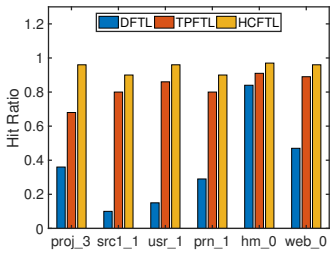


Fig. 4: Hit ratio

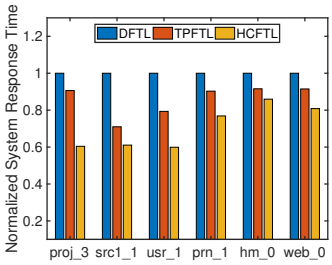


Fig. 5: Average response time

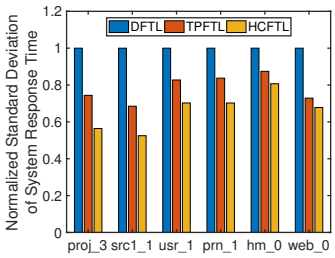
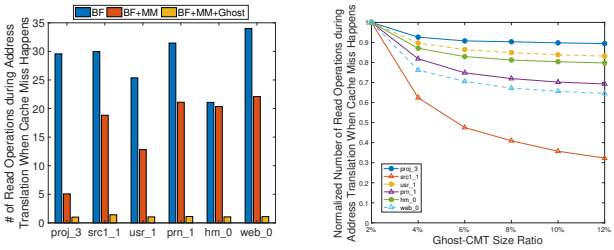


Fig. 6: Standard deviation of response time

3) ***Standard Deviation of Response Time.*** Fig. 6 shows normalized standard deviation of response time with different FTL schemes and workloads. HCFTL reduces the standard deviation of response time by 19.3% – 47.5% and 6.9% – 24.2% for the six traces compared with DFT and TPFTL, respectively. Note that, for *proj_3*, HCFTL reduces the standard deviation of response time by 24.2%. This is also because HCFTL has a higher CMT hit ratio so that the differences of response time between different requests become smaller.



(a) DTPI design (b) Ghost-CMT size

Fig. 7: The impact of DTPI design and Ghost-CMT size

4) ***Impact of DTPIs.*** Fig. 7(a) shows the efficiency of different DTPI configurations, where *BF* means we only use bloom filter, *MM* means we only use the *max* and *min* LPN to perform range check, and *Ghost-CMT* means we employ a Ghost-CMT. In Fig. 7(a), we find that DTPI with bloom filter, *max* and *min* can significantly decrease the number of extra read operations during address translation when cache miss happens. This is because if an LPN is not between *max* and *min*, we will directly return false without running any bloom filter test. Ghost-CMT can further reduce the number of extra read operations, as the difference of *max* and *min* LPN of entries in a DTP can be minimized when employing Ghost-CMT, so that the range check becomes more efficient.

5) ***Impact of Ghost-CMT Size.*** We measure the changes in the number of extra read operations when varying the ratio of the Ghost-CMT size to the total DRAM size from 2% to 12%. Fig. 7(b) shows that the bigger *Ghost-CMT Size Ratio* is, the fewer extra read operations are performed. However, interestingly, the number of extra read operations will finally become stable. Thus too large *Ghost-CMT Size Ratio* is not necessary, and in contrast, the bigger *Ghost-CMT Size Ratio* will cause more negative impact on CMT hit ratio.

V. Conclusions

With the limited built-in DRAM space, this paper proposes a novel page-level FTL scheme, HCFTL, which clusters newly evicted mapping entries into dynamic translation pages, for speeding up the subsequent accesses. HCFTL also employs indexes to quickly determining if a mapping entry is in a dynamic page. A Ghost-CMT is introduced to improve efficiency of the indexes and reduce the negative impacts imposed by bloom filters. Finally, we use trace-driven simulations to evaluate HCFTL with various enterprise workloads, which shows that HCFTL can significantly increase the CMT hit ratio and reduce system response time with limited cache size.

Acknowledgment

This work was supported by National Nature Science Foundation of China under Grant No. 617724861. The work of Yubiao Pan was supported by National Nature Science Foundation of China under Grant 61802133 and Nature Science Foundation of Fujian Province under Grant 2018J05107.

References

[1] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings," in *the International Conference on Architectural Support for Programming Languages and Operating System (ASPLOS)*, 2009.

[2] S. Jiang, L. Zhang, X. Yuan, H. Hu, and Y. Chen, "S-FTL: An efficient address translation for flash memory by exploiting spatial locality," in *27th IEEE Symposium on Mass Storage Systems and Technologies (MSST)*. IEEE, 2011, pp. 1–12.

[3] Y. Zhou, F. Wu, P. Huang, X. He, C. Xie, and J. Zhou, "An efficient page-level FTL to optimize address translation in flash memory," in *the 10th European Conference on Computer Systems*. ACM, 2015, p. 12.

[4] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[5] G. Wu, B. Eckart, and X. He, "BPAC: An adaptive write buffer management scheme for flash-based solid state drives," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE, 2010, pp. 1–6.

[6] N. Dayan, P. Bonnet, and S. Idreos, "GeckoFTL: Scalable flash translation techniques for very large flash devices," in *the 2016 International Conference on Management of Data*. ACM, 2016, pp. 327–342.

[7] F. Chen, T. Luo, and X. Zhang, "CAFTL: A content-aware flash translation layer enhancing the lifespan of flash memory based solid state drives." in *FAST*, vol. 11, 2011, pp. 77–90.

[8] N. Elyasi, M. Arjomand, A. Sivasubramaniam, M. T. Kandemir, C. R. Das, and M. Jung, "Exploiting intra-request slack to improve ssd performance," in *the 22th International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, 2017, pp. 375–388.

[9] J. S. Bucy, J. Schindler, S. W. Schlosser, and G. R. Ganger, "The disksim simulation environment version 4.0 reference manual (cmu-pdl-08-101)," *Parallel Data Laboratory*, p. 26, 2008.

[10] N. Agrawal, V. Prabhakaran, T. Wobber, J. D. Davis, M. S. Manasse, and R. Panigrahy, "Design tradeoffs for SSD performance." in *USENIX Annual Technical Conference*, vol. 8, 2008, pp. 57–70.

[11] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *ACM Transactions on Storage (TOS)*, vol. 4, no. 3, p. 10, 2008.