SCFTL: An Efficient Caching Strategy for Page-Level Flash Translation Layer

Peera Thontirawong
Department of Computer Engineering
Chulalongkorn University
Bangkok, Thailand
Email: peera.t@student.chula.ac.th

Mongkol Ekpanyapong
School of Engineering and Technology
Asian Institute of Technology
Pathumthani, Thailand
Email: mongkol@ait.ac.th

Prabhas Chongstitvatana
Department of Computer Engineering
Chulalongkorn University
Bangkok, Thailand
Email: prabhas@chula.ac.th

***Abstract*—The increasing capacity of NAND flash memory results in larger page size. Since the larger page requires longer access time, the performance of a flash translation layer (FTL) that stores its mapping table in flash pages is degraded. An economical FTL named SCFTL is proposed to avoid such issues caused by the larger page. In order to reduce spatial requirements, SCFTL employs a demand-based caching mechanism for the page mapping table. Unlike other FTLs, SCFTL facilitates two techniques for delicately exploiting the spatial locality and customizes the replacement algorithm for reducing cache miss penalty. The experiments show that the average overhead of SCFTL in terms of access time is only 6.89%; this overhead is 75.96% and 11.35% lower than the state-of-the-art FTLs. The average cache hit ratio of SCFTL is as high as 0.92 despite compact memory footprint. Because of the outstanding cache utilization, SCFTL still achieves high performance even though the page size is larger.**

I. Introduction

Flash-based storage devices are prevalent in computer systems due to various benefits over ferromagnetic storage devices. They are widely used in high performance systems because of low access latency. In addition, the flash-based storage devices are more robust to shock and consume less energy; hence, they became an essential component in most mobile systems.

However, these advantages do not come without restrictions. Unlike ferromagnetic materials, NAND flash memory is unsuitable for in-place update. In order to reprogram a flash page, which consists thousands of flash cells, the page must be prior erased. Due to the technology limitations, each page cannot be erased individually. The smallest erasable unit is one block, which is a group of hundreds pages. Moreover, the lifetime of each flash cell is limited by its program/erase (P/E) cycles. Consequently, a flash translation layer (FTL) is employed to solve these problems and provide the sector-based file system interfaces.

One of the main functions of an FTL is address translation. Since the upper level locates data by logical addresses, an FTL translates them to flash page locations or physical addresses and memorizes pairs of mapped addresses in a page mapping table [1]. Due to the fact that a flash memory contains millions of pages, enormous memory capacity is required. For instance, an 8GB flash memory with 4,096 blocks of 256 pages needs 4,096KB for the page mapping table while a block mapping table [2] takes only 16KB. However, a block mapping table translates the most significant bits of a logical address to a physical block number while a physical address offset are fixed to the least significant bits of the logical address. Owing to more flexibility, the page-level address translation scheme, which facilitates the page mapping table, usually yields better performance and lifetime.

To reduce the spatial requirement, numerous research works put constraints on their address translations [3]. Among these research works, the log buffer-based scheme [4]–[6] is the most popular. However, the log buffer-based scheme suffers from costly merge operations, which are required for rearranging data in the log buffer. Then, another renowned approach has been proposed to eliminate the merge operations [7], [8]. This approach enables fine-grained page selections within blocks and offloads the mapping table to the flash memory. Even so, it cannot fully utilize the flash memory capacity due to block dependency and therefore frequently needs block reclamation.

Recently, a novel approach to lower the memory requirement of the page-level address translation [9]–[15] has been invented. Instead of keeping the entire page mapping table inside RAM, this approach offloads it to the flash memory and caches only small portions. Therefore, the page-level address translation, which is unconstrained, is retained. However, the performance of this approach depends on its cache efficiency.

As one flash page can hold numerous mapping entries, S-FTL [11] and CDFTL [12] exploit the spatial locality by caching whole flash pages in their caches. Nevertheless, the demand of gigantic capacity flash memory drives the flash page size larger [16]. In consequence, caching the entire flash page is too extravagant and causes undesirable effects.

Due to the fact that flash page programming time is substantially longer than reading time, the cost of writing a modified mapping entry back to the flash memory is several times higher than rereading a victim back to the cache. Although the amount of cache writes-back is usually only a small fraction of cache misses, it considerably affects the average address translation time. However, the traditional cache replacement policies, such as LRU, treat modified and unmodified cache blocks equally. Even the recent cache replacement policy [17] is focusing on increasing cache hit rate without differentiating

modified cache blocks. These policies are not aware of the asymmetrical access time of a flash memory. Even though many cache replacement policies are customized for a flash memory [18]–[20], they are not designed for caching the mapping table of a flash memory. Hence, they cannot utilize the localities of mapping table accesses.

Henceforth, we proposes a novel caching strategy for the page-level address translation FTL named SCFTL. SCFTL is designed to be an efficient FTL for a large page flash memory and small RAM. By implementing two spatial locality exploitation techniques and the specialized cache replacement policy, SCFTL achieves the sublime performance of only 6.89% additional latency. It is less than half of the additional latency required by the state-of-the-art FTL, CDFTL. In addition, the average cache hit of SCFTL is as high as 92.04% despite its compact cache capacity. Furthermore, the customized cache replacement policy efficiently reduces the average number of written-back cache victims to only 0.38% of total cache accesses.

The rest of this paper is organized as follows. The related FTLs are described in Section 2. Then, Section 3 presents the proposed FTL, and its performance is evaluated in Section 4. Finally, the paper is concluded in Section 5.

II. Related Works

Although the page-level address translation has many advantages, it is not widely adopted due to the infeasible spatial requirement of the page mapping table. To implement a page-level address translation in limited RAM space, DFTL [9] stores the enormous page mapping table in several pages of the flash memory. The translation page that contains the corresponding physical address is located by a small mapping table in RAM. However, one flash page read is required to obtain the physical address, and updating the mapping table also needs one page write, which drastically burdens the performance. For these reasons, DFTL exploits the temporal locality by caching some mapping entries in RAM to reduce flash page reads and to postpone a translation page update until its modified mapping entry is evicted. Furthermore, the modified mapping entries of the same translation page are combined together into one flash write when one of them is evicted. However, the cache hit ratio of DFTL is not high because it does not take advantage of the spatial locality.

In order to increase cache hit ratio, CFTL [10] and CAST [15] add a consecutive field into their caches. Consecutive logical addresses that are mapped to consecutive physical addresses are grouped into single cache block as illustrated in Fig. 1. Due to the nature of sequential writes, their physical addresses are more likely to be consecutive. Hence, adding the consecutive field improves overall performance. In addition, CFTL improves the performance of infrequently accessed mapping entries by using the block mapping table. On the other hand, CAST biases its physical address selection to prefer contiguous data locations and therefore increases the chance of consecutive addresses.

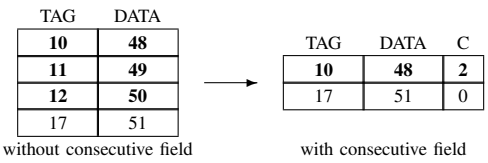

Fig. 1. An example of cache with consecutive field (C). As the physical addresses of the logical addresses 10-12 are consecutive, they can be kept together in the same cache block by setting the consecutive field to 2.

To further exploit the spatial locality, S-FTL [11] caches a whole translation page as a single cache block. Furthermore, it also reduces the cache space needed by compressing the translation page. Caching full translation pages can guarantee spatial locality exploitation; however, each translation page still demands large portion of RAM despite the compression. Hence, S-FTL is not suitable for a device with small RAM.

CDFTL [12] takes another approach to exploit the spatial locality. It employs a two-level cache hierarchy. The first level cache is similar to DFTL, while the second level cache stores translation pages. Hence, the temporal locality is exploited on the first level cache, while the spatial locality is handled by the second level cache. Nevertheless, CDFTL also suffers from the same problem as S-FTL.

Since the size of a flash page tends to grow larger [16], S-FTL and CDFTL cannot maintain the same level of performance without enlarging their caches. In addition, the larger page size means the chance that a file is spanning to multiple pages is lower; hence, the consecutive field will be less effective. In order to utilize the page-level address translation on the large page flash memory, SCFTL is introduced.

III. Design of SCFTL

SCFTL is a page-level address translation FTL that employs the efficient caching strategy. It consists of three main components: page mapping table (PMT), translation page directory (TPD) and cache mapping table (CMT). In order to achieve the page-level address translation, SCFTL stores a page mapping table in several translation pages (TPs). Each translation page keeps a group of physical page numbers mapped to consecutive logical addresses. Due to the gigantic flash page size, each translation page holds thousands of physical page numbers; hence, only few pages are needed for the complete page mapping table. TPD keeps the addresses of every translation page in RAM and indexes them by the most significant bits of logical addresses. To prevent severe performance degradation, several mapping entries are cached in CMT. Furthermore, CMT integrates two spatial locality exploitation techniques and a customized cache replacement policy to enhance its efficiency.

A. Two-Level Address Translation

As the page mapping table of SCFTL is kept inside the flash memory, the address translation has to be done by a two-level process. Generally, the physical address of a request is found in CMT; hence, the two-level address translation is

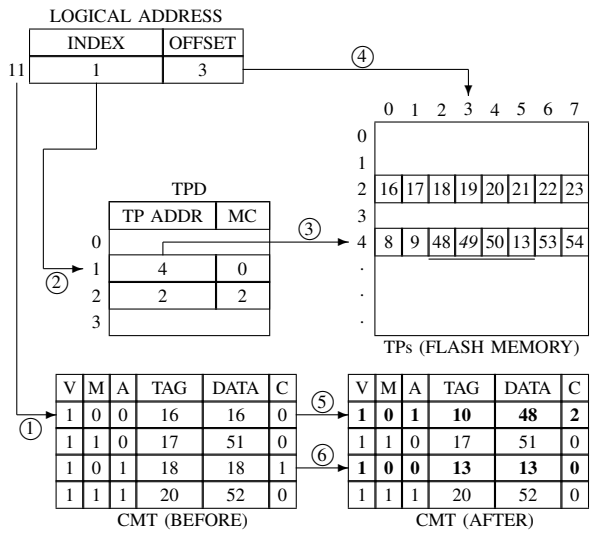

Fig. 2. An example of the SCFTL address translation. Suppose a logical address of the request is 11, and each translation page contains 8 physical addresses; the index and offset of the logical address is 1 and 3, respectively. (1) The access of the logical address 11 incurs a cache miss in CMT, and the first cache block is selected as a victim. A write-back does not occur, as the victim is not modified. Then, (2) the two-level address translation is begun, and the translation page 1 is located at the page number 4. (3) The translation page is read from the flash memory, and (4) the physical address of the request is found at the offset 3. (5) Instead of storing only one mapping entry in CMT, the consecutive physical addresses of the logical addresses 10 and 12 are fetched and stored together with the logical address 11. (6) Assume that the spatial size is 4; another mapping entry 13 will be fetched to CMT. Since this is a spatial fetching, the third cache block is selected as a victim instead of the second cache block, which has MC value lower than the threshold.

not triggered. However, the two-level address translation will be executed in case of a cache miss.

The two-level address translation split a logical address into two parts: index and offset. In the first level, TPD converts the index into the location of the related translation page, and then the located translation page is retrieved from the flash memory. After that, the second level extracts the physical address from the translation page by the offset, which is a position of the physical address in the translation page. Therefore, the logical address is finally translated to the corresponding physical address. An example of the two-level address translation is provided in step 2-4 of Fig. 2.

B. Efficient Caching Strategy

Owing to the temporal locality and the spatial locality of storage accesses, several page reads can be omitted by caching mapped physical page numbers. In addition, the caching allows the update of a translation page to be postponed; the updates on the same translation page can be combined together to minimize the number of additional page writes. However, the efficiency of CMT does not only depend on the temporal locality; it is also highly influenced by the spatial locality.

1) Spatial Locality Exploitation: As a flash page, which is the smallest read/write unit, can pack thousands of mapping entries, caching multiple entries each translation page retrieving is convenient. However, caching an entry that will not be accessed is wasting the cache space. In order to avoid caching unused mapping entries, a fine-grained spatial fetching technique is introduced.

TABLE I
VICTIM SELECTION ORDERS OF D-NRU

Attributes	Normal Fetching	Spatial Fetching
$\neg A \wedge \neg M$	1	1
$\neg A \wedge M \wedge (MC_{TP} \geq c)$	2	2
$\neg A \wedge M \wedge (MC_{TP} < c)$	3	-
$A \wedge \neg M$	4	3
$A \wedge M \wedge (MC_{TP} \geq c)$	5	4
$A \wedge M \wedge (MC_{TP} < c)$	6	-

Despite increasing the cache block size to accommodate more mapping entries, SCFTL spends several cache blocks to exploit the spatial locality. As a result, the chance of cache trashing can be controlled by limiting the amount of mapping entries can be cached in each translation page read. since SCFTL treats each mapping entry as an individual cache block, a low demanded mapping entry can be independently replaced without disturbing others. However, the fine-grained spatial fetching technique forces SCFTL to reacquire the translation page before writing back.

As the physical addresses of sequential writes are likely to be assigned sequentially, facilitating the consecutive field efficiently saves CMT capacity by combining several sequentially mapped entries into one single cache block. In contrast, the drawback of the consecutive field is the additional cache eviction because a cache block is unable to retain the same consecutive value after one of its mapping entries is updated. Henceforth, the cache block has to be split into several cache blocks. However, the split cache blocks can be merged back if their mapping entries are subjected to sequential writes.

2) Cache Replacement Policy: In order to decrease the number of translation page writes, the victim selection process has to discriminate modified cache blocks from others. However, preventing modified cache blocks from being a victim in the fully associative cache may cause inefficient cache capacity utilization. Thus, SCFTL implements a customized cache replacement policy named D-NRU.

D-NRU is very similar to NRU [21]. Each cache block contains a 1-bit flag to indicate that it was recently accessed. Besides, D-NRU takes a modified flag, which is existed in every cache block, into account when it selects a victim. Because the modified mapping entries from the same translation page can be written-back simultaneously, writing the translation page that contains many modified mapping entries is more economical. Hence, D-NRU considers the number of modified mapping entries in each translation page when selects a victim. The counters (MCs) are attached to TPD as shown in Fig. 2. Each MC is very tiny, as it only needs to count until its value reaches the worthwhile threshold.

D-NRU consists of two variants of NRU algorithms. The algorithm selection is based on the type of a mapping entry fetching: normal fetching or spatial fetching. The normal

TABLE II
8GB MLC NAND FLASH MEMORY SPECIFICATIONS [22]

Page Size	8,192 + 448 bytes (data + spare area)
Block Size	256 pages
Device Size	8GB (4,096 blocks)
Page Read Time	75µs
Page Program Time	1,300µs
Block Erase Time	3,800µs
Transfer Rate	50MB/s
Endurance	3,000 P/E cycles
Minimum ECC Requirement	24-bit ECC per 1,080 bytes

fetching has high priority, as it is caused by an I/O request. Its victim selection prefers a cache block that is not recently accessed ($\neg A$), unmodified ($\neg M$) and modified (M) with high MC value (MC_{TP}), respectively. On the other hand, the spatial fetching is initiated by spatial locality exploitation. As its mapping entry may not be reference, the cost of bringing it into the cache should be low. A modified cache block that has the MC value lower than the threshold ($MC_{TP} < c$) will not become the victim of spatial fetching. Furthermore, the recently accessed flag is not set for the cache block that is brought in by spatial fetching. The orders of D-NRU victim selection are provided in Table I, and examples are illustrated in step 1 and 6 of Fig. 2.

IV. PERFORMANCE EVALUATION

In this section, SCFTL will be compared with two state-of-the-art FTLs: DFTL and CDFTL. The experiments are done on the 8GB NAND flash memory [22] as specify in Table II. It is simulated by a customized FlashSim simulator [23] with the cache size is roughly set to 16KB, which is equal to the memory footprint required by a block mapping table. As CDFTL prefers the second level cache to be large, the two-level cache of CDFTL is configured to 2KB and 16KB, respectively.

To evaluate the performance of the FTLs, several workload traces are selected from Storage Performance Council (SPC) [24] and Microsoft Research Cambridge (MSRC) [25]. In case of SPC benchmarks, `Financial` are I/O traces from OLTP applications, while `WebSearch` are I/O traces from a popular search engine. For MSRC benchmarks, the traces from the storage volume 0 of enterprise data centers running various applications are selected. The details of these traces can be found on their publication [25].

As shown in Table II, the page programming time is about 17 times longer than the reading; the penalty time of a cache miss that requires a victim to be written back is much higher. Consequently, the cache hit ratio is insufficient to measure the performance of the cache in the FTLs. In this paper, we introduce another metric called cache write-back ratio ($WB\ Ratio$), which is a proportion between the number of cache blocks that was written back ($num_{writeback}$) and the number of cache accesses (num_{access}) (1).

TABLE III
COMPARISON OF CACHE REPLACEMENT POLICIES ON SCFTL

Policy	$Miss\ Ratio$ (%)	$WB\ Ratio$ (%)	T_{PC} (%)
LRU	11.30	0.81	12.13
NRU	9.85	4.62	16.04
D-NRU-2	7.67	0.51	7.42
D-NRU-3	7.96	0.38	6.89
D-NRU-4	13.46	0.48	10.62

$$WB\ Ratio = \frac{num_{writeback}}{num_{access}} \tag{1}$$

Although the average system response time is a widely adopted performance measurement of FTLs, its value is mainly dominated by data access time. To observe the effect of FTLs more precisely, we measure the percentage change of the average system response time (T_{PC}) from the ideal page-level address translation FTL (PFTL) [1], which can be calculated by (2). Since the page mapping table of PFTL is completely held in RAM, the overhead of PFTL is neglectable. Consequently, T_{PC} is the percentage of extra time required to complete address translation, and a lower value means better performance. Furthermore, the percentage change is already normalized; it can be fairly compared across various benchmarks.

$$T_{PC} = \frac{T - T_{PFTL}}{T_{PFTL}} \times 100 \tag{2}$$

A. Performance of the Efficient Caching Strategy

The performance of D-NRU is shown in Table III. The suffix number of D-NRU is the number of MC bits. According to the design, D-NRU avoids cache trashing by not setting a recently accessed flag for spatial fetching, which in turn lowers cache miss ratio. In addition, it prevents spatial fetching from replacing low beneficial modified cache blocks. Consequently, cache write-back ratio significantly decreases as preventing premature cache writing-back provides additional time to gather more modified mapping entries from the same translation page. However, over protecting, which means too few victim candidates, will heighten the risk of cache trashing and therefore results in high cache miss ratio.

In Fig. 3, SCFTL shows the average cache miss of 7.96%. Due to the very small cache size configuration, the miss ratios of DFTL are very high. Its average cache miss is significantly higher than SCFTL by 75.96%. This enhancement is the impact of the spatial locality exploitation. In addition, the average cache miss of SCFTL is 11.13% lower than CDFTL because SCFTL can preserve the diversity of logical addresses better than the small first level cache of CDFTL.

The comparison of cache write-back ratios between FTLs is shown in Fig. 4. Because D-NRU-3 in SCFTL works efficiently, the average cache write-back is reduced to 0.38%, which is 5.33% lower than DFTL. Due to the small size of the second level cache in CDFTL, the number of cached translation pages is insufficient to absorb the amount of modified

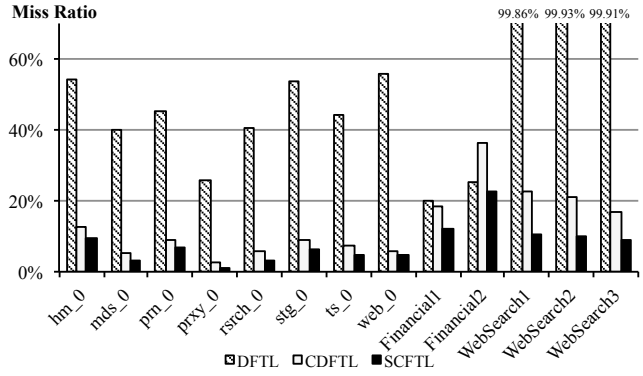

Fig. 3. $Miss\ Ratio$ of 16KB cache configuration FTLs

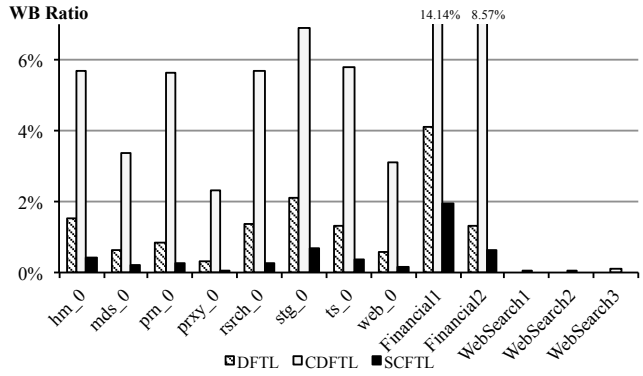

Fig. 4. $WB\ Ratio$ of 16KB cache configuration FTLs

mapping entries evicted from the first level cache. As a result, the average cache write-back of CDFTL is surprisingly high; it is 46.22% higher than SCFTL.

Finally, the average system response time of FTLs are compared in Fig. 5. Due to exceptional cache performance of SCFTL, its average T_{PC} is only 6.89%. It is lower than DFTL and CDFTL, by 82.85% and 18.24%, respectively. The exceedingly high T_{PC} of DFTL in `mds_0` is a result of cache misses in densely read requests, which in turn causes cumulative delay.

In order to match SCFTL performance, CDFTL needs 128KB of the second level cache. Furthermore, SCFTL is still able to excel in 4KB cache configuration, which is smaller than the flash page size and incapability for CDFTL, with 19.28% average T_{PC}.

B. Impact on Flash Memory Lifetime

Because the flash memory endurance is limited by the P/E cycles, the number of extra block erases from PFTL is measured. Owing to the very low cache write-back ratio of SCFTL, the additional block erasure is barely needed. The average block erase count of SCFTL is only 0.67% increased from PFTL, while DFTL and CDFTL are 1.72% and 7.12%, respectively. Consequently, SCFTL is having very little effect on the flash memory lifetime.

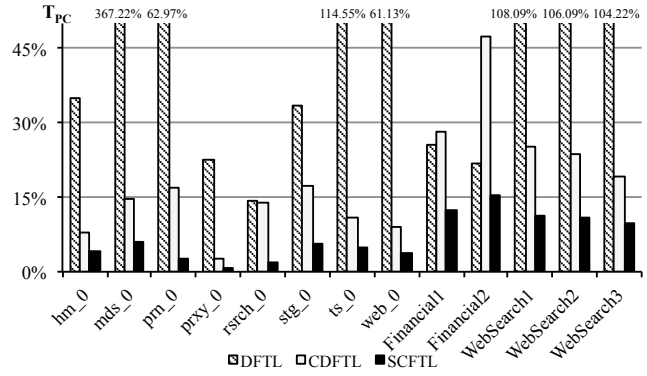

Fig. 5. T_{PC} of 16KB cache configuration FTLs

C. Memory Requirements

According to Table II, the total number of pages is 4096×256. As each page consists 8,192 bytes, 2,048 of 4-byte physical address can be contained. Only 512 translation pages, which are about 0.05% of the total pages, are required for SCFTL, DFTL and CDFTL.

As SCFTL keeps TPD and CMT in RAM, the amount of RAM needed is the summation of these two components. TPD is a simple mapping table with counters. Each line contains a 4-byte translation page address and a 3-bit MC; hence, only 2.25KB of RAM is needed by TPD. Every cache block of CMT consists a 4-byte tag, a 4-byte mapping entry, a 5-bit consecutive field, a 1-bit valid flag, a 1-bit modified flag and a 1-bit recently accessed flag; therefore, each cache block is 9 bytes. The total size of CMT is 18KB with 2,048 cache blocks. Therefore, SCFTL requires only 20.25KB of RAM, while DFTL and CDFTL require 18.50KB and 20.25KB of RAM, respectively.

V. Conclusion

Since the flash memory tends to have larger pages, it is necessary to include this constraint in the design of an FTL. To overcome this restriction, we proposes SCFTL an efficient caching strategy for a page-level address translation FTL. In order to utilize the cache, SCFTL facilitates the fine-grained spatial locality exploitation, the consecutive field and D-NRU, which is a customized cache replacement policy. In spite of limited memory space, SCFTL successfully exploits the spatial locality and reduces the number of cache writes-back. In the 16KB cache configuration, SCFTL needs only 6.89% additional average system response time from the FTL that keeps the complete page mapping table in RAM. The average overhead time of SCFTL is 75.96% and 11.35% lower than DFTL and CDFTL, respectively. In addition, the average cache miss and cache write-back of SCFTL are as low as 7.96% and 0.38%, respectively.

Because the degree of spatial locality is varied in each workload, dynamically adjusting the spatial fetching size should further increase the efficiency of SCFTL. In addition, SCFTL does not have any mapping restriction; it would

be interesting to discover the performance of cutting-edge garbage collections and wear levelers on SCFTL.

REFERENCES

[1] A. Ban, "Flash file system," United State Patent 5 404 485, Apr. 4, 1995.

[2] A. Ban, "Flash file system optimized for page-mode flash technologies," United State Patent 5 937 425, Aug. 10, 1999.

[3] D. Ma, J. Feng, and G. Li, "A survey of address translation technologies for flash memories," *ACM Comput. Surv.*, vol. 46, no. 3, pp. 36:1–36:39, Jan. 2014.

[4] 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 Trans. Embed. Comput. Syst.*, vol. 6, no. 3, pp. 18:1–18:27, Jul. 2007.

[5] S. Lee, D. Shin, Y.-J. Kim, and J. Kim, "LAST: Locality-aware sector translation for NAND flash memory-based storage systems," *SIGOPS Oper. Syst. Rev.*, vol. 42, no. 6, pp. 36–42, Oct. 2008.

[6] Y. Guan, G. Wang, Y. Wang, R. Chen, and Z. Shao, "BLog: Block-level log-block management for NAND flash memorystorage systems," in *14th ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES)*, Jun. 2013, pp. 111–120.

[7] J.-U. Kang, H. Jo, J.-S. Kim, and J. Lee, "A superblock-based flash translation layer for NAND flash memory," in *6th ACM & IEEE International Conference on Embedded Software (EMSOFT)*, Aug. 2006, pp. 161–170.

[8] Z. Qin, Y. Wang, D. Liu, Z. Shao, and Y. Guan, "MNFTL: An efficient flash translation layer for MLC NAND flash memory storage systems," in *48th Design Automation Conference (DAC)*, Jun. 2011, pp. 17–22.

[9] A. Gupta, Y. Kim, and B. Urgaonkar, "DFTL: A flash translation layer employing demand-based selective caching of page-level address mappings," in *14th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, Mar. 2009, pp. 229–240.

[10] D. Park, B. Debnath, and D. Du, "CFTL: A convertible flash translation layer adaptive to data access patterns," in *ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, Jun. 2010, pp. 365–366.

[11] 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 *IEEE 27th Symposium on Mass Storage Systems and Technologies (MSST)*, May 2011, pp. 1–12.

[12] Z. Qin, Y. Wang, D. Liu, and Z. Shao, "A two-level caching mechanism for demand-based page-level address mapping in nand flash memory storage systems," in *17th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*, Apr. 2011, pp. 157–166.

[13] D. Ma, J. Feng, and G. Li, "LazyFTL: A page-level flash translation layer optimized for NAND flash memory," in *ACM SIGMOD International Conference on Management of Data (SIGMOD)*, Jun. 2011, pp. 1–12.

[14] Y.-H. Chang, C.-K. Hsieh, P.-C. Huang, and P.-C. Hsiu, "A caching-oriented management design for the performance enhancement of solid-state drives," *Trans. Storage*, vol. 8, no. 1, pp. 3:1–3:21, Feb. 2012.

[15] Z. Xu, R. Li, and C.-Z. Xu, "CAST: A page-level FTL with compact address mapping and parallel data blocks," in *IEEE 31st International Performance Computing and Communications Conference (IPCCC)*, Dec. 2012, pp. 142–151.

[16] M. Abraham, "The impact of NAND lithography trends on system design," presented at the Flash Memory Summit, Aug. 2013.

[17] A. Jaleel, K. B. Theobald, S. C. Steely, Jr., and J. Emer, "High performance cache replacement using re-reference interval prediction (RRIP)," in *37th Annual International Symposium on Computer Architecture (ISCA)*, Jun. 2010, pp. 60–71.

[18] S.-y. Park, D. Jung, J.-u. Kang, J.-s. Kim, and J. Lee, "CFLRU: A replacement algorithm for flash memory," in *International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, Oct. 2006, pp. 234–241.

[19] H. Jung, H. Shim, S. Park, S. Kang, and J. Cha, "LRU-WSR: Integration of LRU and writes sequence reordering for flash memory," *IEEE Transactions on Consumer Electronics*, vol. 54, no. 3, pp. 1215–1223, Aug. 2008.

[20] H. Kim, M. Ryu, and U. Ramachandran, "What is a good buffer cache replacement scheme for mobile flash storage?" in *12th ACM SIGMETRICS/PERFORMANCE Joint International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*, Jun. 2012, pp. 235–246.

[21] *UltraSPARC T2™ Supplement to the UltraSPARC Architecture 2007*, Sun Microsystems, Sep. 2007.

[22] "MT29F64G08C," Data Sheet, Micron Technology.

[23] Y. Kim, B. Tauras, A. Gupta, and B. Urgaonkar, "FlashSim: A simulator for NAND flash-based solid-state drives," in *First International Conference on Advances in System Simulation (SIMUL)*, Sep. 2009, pp. 125–131.

[24] (2013, Aug.) SPC traces. Storage Performance Council. [Online]. Available: http://traces.cs.umass.edu/index.php/Storage

[25] D. Narayanan, A. Donnelly, and A. Rowstron, "Write off-loading: Practical power management for enterprise storage," *Trans. Storage*, vol. 4, no. 3, pp. 10:1–10:23, Nov. 2008.