LETTER

An Empirical Study of FTL Performance in Conjunction with File System Pursuing Data Integrity*

In Hwan DOH[†], Myoung Sub SHIM[†], *Nonmembers*, Eunsam KIM[†a)], *Member*, Jongmoo CHOI[††], Donghee LEE[†††], *and* Sam H. NOH[†], *Nonmembers*

SUMMARY Due to the detachability of Flash storage, which is a dominant portable storage, data integrity stored in Flash storages becomes an important issue. This study considers the performance of Flash Translation Layer (FTL) schemes embedded in Flash storages in conjunction with file system behavior that pursue high data integrity. To assure extreme data integrity, file systems synchronously write all file data to storage accompanying hot write references. In this study, we concentrate on the effect of hot write references on Flash storage, and we consider the effect of absorbing the hot write references via nonvolatile write cache on the performance of the FTL schemes in Flash storage. In so doing, we quantify the performance of typical FTL schemes for a realistic digital camera workload that contains hot write references through experiments on a real system environment. Results show that for the workload with hot write references FTL performance does not conform with previously reported studies. We also conclude that the impact of the underlying FTL schemes on the performance of Flash storage is dramatically reduced by absorbing the hot write references via nonvolatile write cache.

key words: *Flash Translation Layer (FTL), flash memory, file system, metadata, non-volatile RAM, write cache*

1. Introduction

As Flash memory technology is getting mature, Flash storage such as USB Flash drives are prevalent in various computing environments. Flash storage normally adopts an FTL to efficiently manage the underlying Flash memory. As the efficiency of FTL dominates the performance of Flash storage, issues related to improving FTL performance have been the focus of much research [5], [8], [9]. In addition to the I/O performance enhancement of Flash storage, it is important to assure the data integrity stored in Flash memory against system crashes such as sudden power failure or unexpected removal of the Flash storage. Hence, to assure high data integrity, file systems may possibly write all the file data and metadata synchronously, choosing to sacrifice write performance.

Efforts of file systems to support high data integrity may cause different effects on the performance of Flash storage according to the FTL schemes adopted in the Flash storage. The synchronous write requests for files is accompanied with frequent updates for the file system metadata. We refer to these intermittent update requests for a specific context as the *hot write reference*. The hot write references may be tolerable for some FTL schemes, while for others, these may lead to unacceptable write performance. To support high data integrity without performance degradation, various forms of nonvolatile write cache may possibly be adopted to absorb these hot write references. Again, this effort may cause different effects on different FTL schemes in terms of execution performance.

In this study, we evaluate the performance of typical FTL schemes in regards to the hot write references incurred by synchronous writes of the VFAT file system on a real experimental environment. In addition, we quantify and analyze the effect of absorbing the hot write references via Nonvolatile RAM (NVRAM) on the FTL schemes. Employing NVRAM also allows the file system to ensure high data integrity of Flash storages without sacrificing performance.

The experimental results show that, first, the FTL schemes perform differently from what has been previously reported. Specifically, for some situations the Log block scheme shows worse performance compared to the Replacement block scheme, while for some others the FAST scheme performs worse than the Log block scheme due to the hot write references generated by synchronous file writes. Second, by absorbing the hot reference pattern, the performance of the FTL schemes can be dramatically improved, while at the same time, the performance gap among the FTL schemes may be reduced considerably.

The remainder of this paper is organized as follows: In the next section, we briefly review related works and the FTL schemes that we consider. Then, in Sect. 3, we present the methodology for absorbing hot write references that was used in our experiments. In Sect. 4, we describe the hardware setup, software implementation, and workloads for our evaluation. We discuss the experimental results in Sect. 5. Finally, in Sect. 6, we conclude with a summary and directions for future research.

2. FTL Schemes Considered

The typical FTL schemes that we consider in this study are the Replacement-block scheme, the Log-block scheme, and the FAST scheme [2], [8], [9]. For a better understanding of our work, the overview of the three schemes is presented

Manuscript received September 17, 2009.
Manuscript revised April 1, 2010.

[†]The authors are with the Hongik University, Korea.
[††]The author is with the Dankook University, Korea.
[†††]The author is with the University of Seoul, Korea.

*This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MEST) (No.R0A-2007-000-20071-0, No.R01-2008-000-12028-0).”

a) E-mail: eskim@hongik.ac.kr

DOI: 10.1587/transinf.E93.D.2302

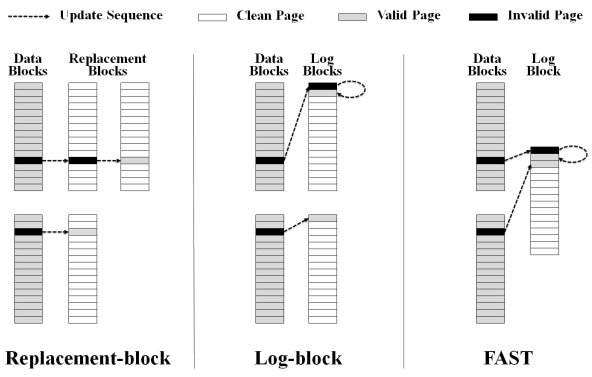


Fig. 1 Comparisons of update sequences for the FTL schemes considered (Replacement-block, Log-block, and FAST).

focusing on their differences. Figure 1 illustrates the distinct actions taken for the Replacement-block, Log-block, and FAST scheme for a sequence of update requests. The FTL schemes process two consecutive update requests for the same Logical Block Number (LBN) in Flash memory and, in turn, an update request for the LBN located in another block.

As block erase operations are accompanied by allocating these additional blocks for replacement or logging, the Replacement-block scheme seems to be easily suffered from the hot write references. Although the Log-block scheme appears to overcome the weakness of the Replacement-block scheme, it still may suffer from low utilization of log blocks when frequent random update requests come across a wide range of blocks. In turn, the FAST scheme appears to resolve the weakness of the Log-block scheme.

The performance of FTL schemes are very closely related to the behavior of the underlying file system. However, most studies related to FTL schemes have been simulation based studies, concentrating on the performance of FTL schemes independent of file system behavior. In this work, we evaluate the FTL schemes on a real system environment, and the performance of the schemes are evaluated in conjunction with a file system behavior, in particular, when the file system pursues high data integrity.

3. Absorbing Hot Write References

This study focuses on the effect of hot write references and on the effect of the nonvolatile write cache for the hot write references on the FTL schemes. The hot write references incurred to support high data integrity can be absorbed by various forms of nonvolatile write cache. Note that employing nonvolatile write cache does not degrade data integrity since the file system that we employ writes all file data and metadata synchronously to nonvolatile storage by invoking the fsync() function. This indicates that the contents cached in the nonvolatile write cache always can be retained irrespective of system failure. In this section, we describe the various ways hot write references may be absorbed, and in particular, the method that is adopted in our evaluation.

Nonvolatile write caches can come in various forms according to the software layer in which the cache is adopted, the nonvolatile media that the cached data is stored in, and the method that the cache selects hot write references. Specifically, first, the write cache can be located in the file system layer [10], the block device driver layer [6], and the FTL layer [7]. Second, for nonvolatile write caches, there are various forms of nonvolatile storage media such as Battery-backed RAM and Nonvolatile RAM (NVRAM). The NVRAM such as FeRAM (Ferro-electric RAM), PRAM (Phase-change RAM), and MRAM (Magnetoresistive RAM), which have recently caught the interest of major semiconductor companies may also be used as nonvolatile write cache storage [3]. Third, hot write references may be absorbed in various ways. Many schemes for selecting hot references based on recency, frequency, and characteristics of file system metadata have been proposed [4], [6].

In our evaluation, we consider the VFAT file system provided in Linux as the host file system since various Flash storages are pre-formatted by the FAT file system, which is the dominant file system for Flash storage devices. For high data integrity, we assume that the applications always write files synchronously. The write cache that we consider is located in the file system layer and the nonvolatile media for our write cache is NVRAM, specifically, FeRAM. As hot write references, we take the write references for file system metadata such as File Allocation Table (FAT) and DOS_DIR_ENTRY. Hence, we modify the VFAT file system to make use of an NVRAM write cache for its metadata.

4. Experimental Setup

Our experiments are conducted on a real embedded system environment. In this section, we describe the hardware system that we use in our experiments and briefly mention the implementation of the FTLs. We also introduce the file system workloads that we consider.

4.1 Hardware and Software Implementation

For the performance evaluation, the FTL schemes are deployed in a real embedded system development board. The embedded development board has 400 MHz CPU, 64 MB SDRAM and 64 MB NAND Flash memory. The NVRAM daughter board developed in-house is able to attach a maximum of 64 MB of FeRAM. For the evaluations, the file system always mounts a 32 MB partition of the NAND Flash memory and exploits just 128 KB of FeRAM, which is enough to absorb the whole file system metadata in our benchmark configuration. The NVRAM appears in the physical memory address space and can be directly accessed from the CPU via memory mapped addressing.

For evaluation, we implement the FTL schemes that we consider in Linux 2.6.21. For the Replacement block scheme, specifically, we modify the NFTL module included in Linux 2.6.21, that is, an FTL provided only for Flash devices developed by M-System [2]. Both the Log block scheme and the FAST scheme are implemented on the MTD (Memory Technology Device) layer. The Log block scheme and the FAST scheme is implemented based on the description given in the paper by Kim et al. and Lee et al., respectively [8], [9]. For all the experiments we set both the number of log blocks and the number of replacement blocks to 64.

4.2 File System Workload

To the best of our knowledge, there is no de facto realistic file system workload that reflects the characteristics of embedded systems. Thus, we develop a new benchmark program that we call Camera-MS500 that simulates file system operations for the camera module built into a popular real world mobile phone. Since the goal of our experiments is to evaluate the performance of the FTLs in conjunction with the NVRAM cache, we chose the benchmark for the camera that is one of the common products that employs FTLs. By utilizing the BitPim tool published as a Sourceforge project, we can observe the changes of the file system after taking pictures and deleting the pictures [1]. Using this observation, we implement the Camera-MS500 benchmark program that generates a realistic file system workload for a digital camera.

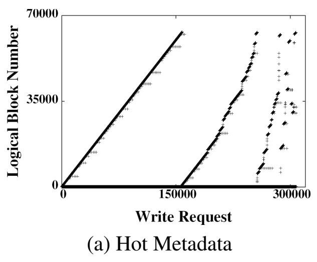
The workload that the Camera-MS500 generates consists of several photo files and a file that is frequently updated to retain index information such as the creation date and user descriptions for each photo file. The size of an acyclic photo index file increases by 96 B in proportion to the number of photo files. When photo files are created or deleted, this photo index file synchronously creates or deletes index information for the corresponding photo file.

The benchmark initially has no file. Then, it executes a specified number of transactions, in our case, set to 10. During a transaction, the benchmark generates 20 photo files whose size ranges from 300 KB to 500 KB and deletes half of the generated files. If the total number of photo files is greater than 60, all the photo files are deleted to make space available. In our configuration, the total footprint that the Camera-MS500 benchmark generates is 79.67 MB. To support high data integrity, we implement it so that each photo file and photo index file writes synchronously in 512 B units.

5. Performance Evaluation

In this study, we are interested in how performance changes as various FTL schemes are employed when the file system provides high data integrity to Flash storage. In this section, first, we discuss the write reference patterns represented by the workload. Then we discuss the effect of hot write references and the removal of the references via NVRAM on the FTL schemes in detail.

In Fig. 2, we see the write reference pattern generated by the Camera-MS500 benchmark. Each graph represents the (y-axis) the Logical Block Number (LBN) for write referenced blocks as (x-axis) write requests are processed during the execution of the Camera-MS500 benchmark. Figure 2 (a) is for workload including hot write references, while Fig. 2 (b) is for workload where the hot write references are absorbed via NVRAM.



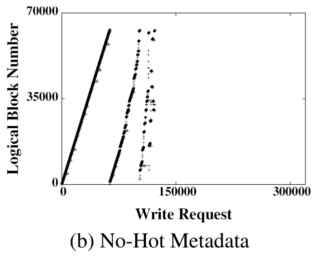


Fig. 2 Write reference patterns for the Camera-MS500 benchmark: (a) before and (b) after absorbing hot write references.

Table 1 Total execution time measured at the application layer for the Camera-MS500 benchmark: before (referred to as "Hot Metadata") and after (referred to as "No-Hot Metadata") absorbing hot write references (Unit: second).

	Hot Metadata	No-Hot Metadata
Repl. FTL	9731.15	324.65
Log FTL	1116.10	350.33
FAST FTL	2200.72	331.29

The Camera-MS500 benchmark generates sequential write references with hot write references for file system metadata, which is represented in the low numbered sectors. We observe that 60% of write references is intensive to 0.2% of total LBNs for the Camera-MS500 benchmark. This is shown on Fig. 2 (a). After absorbing the hot write references via NVRAM write cache, the total number of write requests shown in Fig. 2 (b) decreases drastically as shown by the disappearance of the dots along the x-axis for Fig. 2 (a).

Let us discuss the performance comparisons of the FTL schemes for the Camera-MS500 benchmark and then, analyze the performance variations incurred by absorbing hot write references. Table 1 represents the performance of the FTL schemes for the Camera-MS500 benchmark in terms of the total execution time measured at the application layer.

Let us discuss the performance for each FTL scheme for the Camera-MS500 benchmark. The Log-block scheme outperforms not only the Replacement-block scheme, but also the FAST scheme for the Camera-MS500 workload that contains hot write references. As expected, the Replacement-block scheme suffers from hot write references whereas both the Log-block scheme and FAST scheme endure hot write references well.

Contrary to expectation, the FAST scheme does not perform as well as the Log-block scheme for this workload. The reason behind this is that the FAST scheme uses a log block for all data blocks in Flash memory whereas the Log-block scheme uses a log block for a single data block as mentioned in Sect. 2. In so doing, the FAST scheme misses more on opportunities to switch merge the log block to a data block on sequential write references as the log block is mixed with hot write references for other data blocks as well.

As shown in Table 1, when hot write references are absorbed via NVRAM, the performance improvement varies from 3 to 30 times for the different FTLs. Specifically, for the Replacement-block scheme the improvement is 30 times, while for the Log-block scheme and the FAST scheme, it is 3.2 times and 6.6 times, respectively. Note that the workload contains sequential write patterns with numerous hot write references for the file system metadata, and the hot write references, that is, file system metadata writes, can be absorbed via NVRAM write cache. Total 60% of write requests from the file system are reduced in Camera-MS500 by absorbing hot write references. Also from the table, we can see that the performance difference among the FTL schemes become negligible after absorbing the hot write references.

It is worth noting that the Replacement-block scheme outperforms the Log-block scheme after absorbing hot write references. The reason behind this is that the advantage of the Log-block scheme over the Replacement-block scheme becomes minimal as the majority of the workload becomes sequential after absorbing hot write references. Furthermore, due to its simplicity, the overhead of managing the mapping table for the Replacement-block scheme is minimal, leading to better performance in a real system environment.

6. Conclusions

In this paper, we considered the effect of hot write references on the performance of FTL schemes when file systems ensure high data integrity. We also consider the effect of absorbing these hot write references via NVRAM on the performance of the FTL schemes in Flash storage.

Through the evaluations, we observe that the performance results of the FTL schemes, when hot write references exist, do not conform with results of previous studies on FTL schemes. We also conclude that the impact of the underlying FTL schemes on the performance of Flash storage is dramatically reduced by absorbing the hot write references via NVRAM.

Though we have presented the empirical study of FTL schemes, the work here is still preliminary. As mentioned earlier, nonvolatile caches for hot write references can be deployed in various ways. As future work, we intend to broaden our study to NVRAM write cache in the FTL layer and the cache management policy issues for efficiently absorbing the hot write references.

References

[1] BitPim Sourceforge Project, http://www.bitpim.org
[2] A. Ban, Flash file system. United States Patent, no.5, 404, 485, 1995.
[3] G.W. Burr, B.N. Kurdi, J.C. Scott, C.H. Lam, K. Gopalakrishnan, and R.S. Shenoy, "Overview of candidate device technologies for storage-class memory," IBM J. Res. Dev., vol.52, no.4–5, pp.439–447, 2008.
[4] I.H. Doh, J. Choi, D. Lee, and S.H. Noh, "Exploiting non-volatile RAM to enhance flash file system performance," Proc. EMSOFT'07, 2007.
[5] E. Gal and S. Toledo, "Algorithms and data structures for flash memories," ACM Comput. Surv., vol.37, no.2, 2005.
[6] B.S. Gill and D.S. Modha, "WOW: Wise ordering for writes. Combining spatial and temporal locality in non-volatile caches," Proc. FAST'05, 2005.
[7] H. Kim and S. Ahn, "BPLRU: A buffer management scheme for improving random writes in flash storage," Proc. FAST'08, 2008.
[8] J. Kim, J.M. Kim, S.H. Noh, S.L. Min, and Y. Cho, "A space-efficient flash translation layer for compactflash systems," IEEE Trans. Consum. Electron., vol.48, no.2, pp.366–375, 2002.
[9] 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. Sys., vol.6, no.3, article no.18, 2007.
[10] K. Salem and S. Akyürek, "Management of partially safe buffers," IEEE Trans. Comput., vol.44, no.3, pp.394–407, 1995.