

Impact of NVRAM Write Cache for File System Metadata on I/O Performance in Embedded Systems[†]

In Hwan Doh

Hyo J. Lee

Young Je Moon

Eunsam Kim

Hongik University
Seoul 121-791, Korea

{ihdoh, hjee, yjmoon}@mail.hongik.ac.kr, eskim@hongik.ac.kr

Jongmoo Choi
Dankook University
Seoul 140-714, Korea
choijm@dankook.ac.kr

Donghee Lee
University of Seoul
Seoul 130-743, Korea
dhl_express@uos.ac.kr

Sam H. Noh
Hongik University
Seoul 121-791, Korea
samhnoh@hongik.ac.kr

ABSTRACT

File systems make use of part of DRAM as the buffer cache to enhance its performance in traditional systems. In this paper, we consider the use of Non-Volatile RAM (NVRAM) as a write cache for metadata of the file system in embedded systems. NVRAM is a state-of-the-art memory that provides characteristics of both non-volatility and random byte addressability. By making NVRAM a write cache for dirty metadata, we retain the same integrity of a file system that always synchronously writes its metadata to storage, while at the same time improving file system performance to the level of a file system that always writes asynchronously. To show quantitative results, we develop an embedded board with NVRAM and modify the VFAT file system provided in Linux 2.6.21 to accommodate the NVRAM write cache. The experimental results show that substantial reductions in execution time are possible from an application viewpoint. Another consequence of the write cache is its benefits at the FTL layer, leading to improved wear leveling of Flash memory and increased energy savings, which are important measures in embedded systems.

Categories and Subject Descriptors

C.3 [Computer Systems Organization]: Special-Purpose And Application-Based Systems—*Real-time and embedded systems*; D.4.3 [Operation Systems]: File System Management—*Access methods, directory structures*

General Terms

Design, Experimentation, Measurement, Performance

[†]This work was supported by the Korea Science and Engineering Foundation(KOSEF) grant funded by the Korea government(MOST) (No. R0A-2007-000-20071-0)

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SAC'09 March 8-12, 2009, Honolulu, Hawaii, U.S.A.

Copyright 2009 ACM 978-1-60558-166-8/09/03 ...\$5.00.

Keywords

Non-Volatile RAM (NVRAM), Flash memory, File System, Metadata, Flash Translation Layer (FTL)

1. INTRODUCTION

We consider the use of Non-Volatile RAM (NVRAM) as a write cache for metadata of the file system in embedded systems. Through this use of NVRAM, we show that we can retain the same integrity of a file system that always synchronously writes its metadata, while in terms of speed, performing at the level of a file system that always writes asynchronously. We also show that by making use of NVRAM as a write cache other benefits such as improved wear leveling and energy efficiency in Flash memory are possible.

NVRAM is a state-of-the-art memory that provides characteristics of both non-volatility and random byte addressability. This is in contrast to Flash memory, which, in general, does not provide random byte addressability, and DRAM, which does not provide non-volatility. Major semiconductor companies such as Samsung, Intel, and IBM have spotlighted FeRAM (Ferroelectric RAM), MRAM (Magnetoresistive RAM) and PRAM (Phase-change RAM) as representative types of NVRAM [1, 2, 3]. As semiconductor companies concentrate on the development of NVRAM, we can anticipate that NVRAM will, in the near future, become an everyday component, first, in embedded systems, then later, in commodity computer systems by replacing ROM and/or Flash memory and/or DRAM.

In this study, we consider the use of NVRAM as a write cache for file system metadata. The motivation behind this study begins from the use of caches in file systems, in particular, for write requests. Figure 1(a) shows the number of writes to particular sectors of secondary storage when a cache does not exist. This would be a situation when all write requests are synchronously written to secondary storage. We see that for some low numbered sectors extensive access requests are being made. These lower sectors, in reality, represent sectors where metadata of the file system is being stored. The effect of (volatile) caching is shown in Figure 1(b). With volatile caches, caches are occasionally flushed to storage to synchronize the data in cache and those in storage. We see that with this, substantial write counts are reduced for many of the lower numbered sectors. This is the effect of traditional caches in conventional file systems.

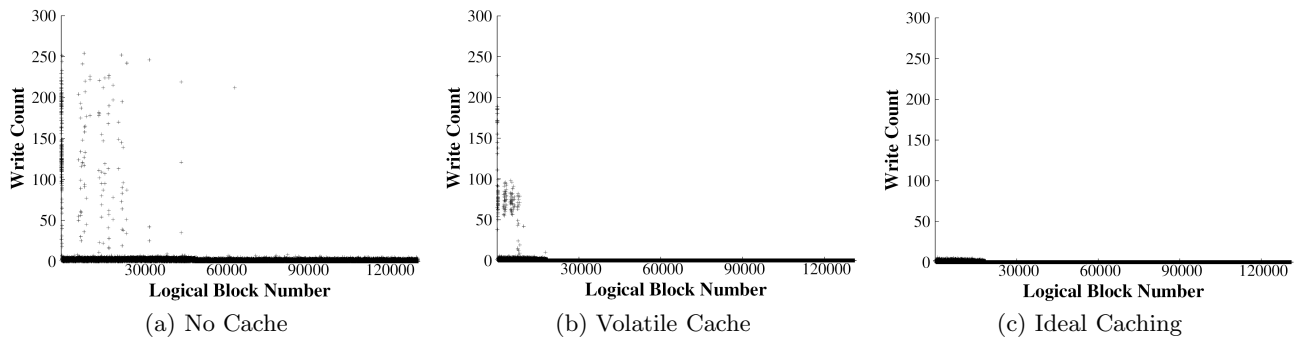


Figure 1: Effects of the NVRAM write cache for file system metadata

For this study, we go a step further and make use of NVRAM as a write cache. We started out this study with a conjecture that by making use of NVRAM as a write cache, this cache will be able to filter out even more requests so that the requests observed at storage will look something like Figure 1(c). The NVRAM write cache comes with an added bonus that integrity does not have to be sacrificed at all even with the performance gains that may be achieved.

Based on this motivation, in this study, we consider using NVRAM as a write cache in the FAT file system, a dominant file system for embedded systems. Through real implementations, we show that by using limited NVRAM write cache and simply caching metadata, we can, indeed, obtain substantial performance gains. We also show that by so doing we obtain benefits at the Flash Translation Layer (FTL) as well. Specifically, wear leveling improves dramatically and energy efficiency improves considerable as well.

The remainder of this paper is organized as follows. In the next section, we briefly review related works and backgrounds. Then, in Section 3, we present the NVFAT file system that is a modification of the VFAT file system to make use of the NVRAM write cache for VFAT file system metadata. In Section 4, we describe experimental environments, that is, the hardware setup and workloads used in our experiments. We discuss the experimental results in Section 5. Finally, in Section 6, we conclude with a summary and directions for future research.

2. BACKGROUND AND RELATED WORKS

In this section, the characteristics of NVRAM is briefly mentioned and compared with those of Flash memories and typical RAM. Then, we present some previous works related to this study.

2.1 Non-Volatile RAM

NVRAM has distinctive characteristics compared with both Flash memories and SDRAM. NVRAM such as FeRAM and MRAM has non-volatility and byte addressability each of which is found in Flash memory and SDRAM, respectively. Note that Flash memories do not support byte addressability except for read operations in NOR Flash memory. The access time of NVRAM, which is roughly one to two orders of magnitude slower than SDRAM, is much faster than Flash memory. Notably, for write access, NVRAM is roughly three orders of magnitude faster than Flash memory. Specifically, the write access time of NVRAM is in the hundreds of nanosecond range while writes in Flash memory

is in the hundreds of microsecond range. Furthermore, the poor write access time of Flash memory can be exacerbated when it is accompanied by an erasure operation that can spend time in the millisecond range. In addition to its fast access time, NVRAM has endurance as high as SDRAM. While Flash memory may be worn out after around 10,000 to 100,000 block erasures, NVRAM has, for practical purposes, no such limitation.

2.2 Related Works

Much research has already been conducted on issues related to improving file system performance by dealing with metadata differently from file data and also by exploiting NVRAM. Also, there are considerable studies that exploit NVRAM as write cache.

Since the characteristics of file system metadata and file data are quite distinct, several file systems that make use of NVRAM as an extension of metadata storage have been proposed. The HeRMES file system proposed by Miller et al. adopts MRAM to store metadata, while storing file data in disk [14]. The Conquest file system also considers NVRAM as storage for metadata and relatively small files [15]. Here the NVRAM that the authors consider is battery-backed RAM. In these works, NVRAM is considered in conjunction with the hard disk drive. Different from these works, Doh et al. propose the MiNV file system that maintains all the metadata in NVRAM while storing all the file data in Flash memory [7]. Our work is distinctive from all of these works as dedicated NVRAM write cache for file system metadata is considered over Flash memory storage along with FTL.

There are also studies regarding other uses of NVRAM. In the early 1990s, research on making use of NVRAM in general purpose computer systems were conducted. However, NVRAM being considered at the time was mostly battery-backed RAM. Specifically, Baker et al. show that write traffic can be significantly reduced with the help of NVRAM in a network file system environment [5]. Chen et al. proposed the Rio file cache that supports fault tolerance in file systems without degrading performance by using NVRAM [6]. Recently, the write-aware buffer management scheme that considers NVRAM as a write buffer cache was proposed by Lee et al. [13]. Kim and Ahn propose a write buffer management scheme called BPLRU for improving random write performance of Flash storages in desktop environments [11]. Different from these works, our study concerns about the impact of write cache for file system metadata on the I/O performance in embedded systems.

3. NVFAT: VFAT FILE SYSTEM WITH NVRAM WRITE CACHE CONSIDERED

In this section, we discuss the implementation of NVFAT. NVFAT is a modification of the VFAT file system in Linux such that the VFAT file system metadata may be managed in the NVRAM write cache.

3.1 NVFAT: Adapting the VFAT File System

The architecture of the I/O subsystem that adopts NVRAM write cache for file system metadata is a configuration much like that shown in Figure 2 (b) compared with conventional I/O path, Figure 2 (a). For our prototype implementation, we take the VFAT file system in the Linux 2.6.21. In the NVRAM write cache, only dirty metadata for the VFAT file system are maintained in disk sector size units, that is, 512 bytes. The VFAT file system metadata consists of the FAT (File Allocation Table) and the MSDOS_DIR_ENTRY, which contains such information as the file or directory name, ownership, access permission, and so on. Whenever the contents of these two kinds of metadata are changed, the whole sector containing the updated metadata is synchronously written to NVRAM. In so doing, the NVFAT file system now has the same level of file system metadata integrity as a VFAT file system that always synchronously writes its metadata to storage.

To accommodate NVRAM write cache, two main modifications are made to the VFAT file system. First, it is modified to call the NVRAM write cache interface instead of the original interface for accessing secondary storage. Second, provisions are made for the VFAT file system to flush all the cached metadata in NVRAM into secondary storage.

Specifically, the `sb_bread()` and `mark_buffer_dirty()` functions for VFAT file system are replaced by the `nvfat_sb_bread()` and `nvfat_buffer_dirty()` functions provided for NVRAM write cache access. The `sb_bread()` and `mark_buffer_dirty()` function is invoked whenever the VFAT file system metadata is read from and written to the storage, respectively. This is done in sector size (512B) units. The `nvfat_sb_bread()` and `nvfat_buffer_dirty()` functions perform similar functionality but considers NVRAM write cache as well. For these functions, we also update cached metadata in 512B sector units as well. This is a choice that we make for compatibility reasons for future deployment with other file systems.

To flush cached metadata in the NVFAT file system, we provide the `nvfat_flush_nvram()` function. Under normal operation, this function is invoked only when the `fat_put_super()` function is executed during the un-mount process of the NVFAT file system. This process is essential for maintaining

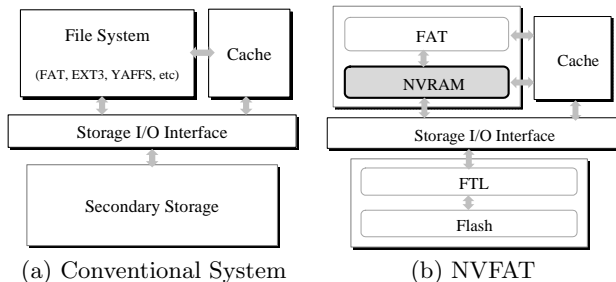


Figure 2: Architecture of the I/O subsystem

consistency between the file system metadata in NVRAM and that stored in detachable storage such as a USB memory card and removable Flash memory storage. If we do not consider detachable storage, this process for maintaining metadata consistency may not be required.

3.2 NVRAM Write Cache Module for NVFAT

The NVRAM write cache module employed to manage the NVFAT file system metadata can be dynamically loaded and executed as a Linux kernel module. This module consists of two parts, that is, the NVRAM management sub-module, which manages the NVRAM allocation, and the write cache management sub-module, which deploys specific policies for managing the write cache.

To manage the NVRAM, we adopt the BGET Memory Allocator that is a comprehensive memory manager. This allocator is efficient in terms of both time and space, and has been frequently employed in embedded system applications [4].

In managing the write cache, we employ a hash table for fast look up of LBNs (Logical Block Number). For cache management, we employ the Least Recently Written (LRW) destaging scheme, which is analogous to the LRU replacement scheme [9]. As a block replacement scheme is necessary for traditional volatile cache management, a destaging scheme is necessary for write caches that retain dirty blocks. Destaging is a process of deciding how many blocks in cache to flush to storage, selecting the blocks to flush, and deciding when to actually flush these blocks. As destaging can have a strong influence on the performance of a system, considerable research has been conducted on this issue [5, 9, 13]. Even though the destage scheme is important to improve the write cache performance, we do not consider this issue here as this is beyond the scope of this study. Instead, we simply employ a naïve destage scheme. In our study, destage is triggered only when there is no room for the newly updated metadata, and only one sector is flushed, based on the LRW replacement scheme, to make room for this one new data item.

4. EXPERIMENTAL ENVIRONMENT

In this section, we discuss the hardware experimental environment, and then present the file system workloads used in the experimental evaluation.

4.1 Hardware Setup

Figure 3 shows an embedded development board with an NVRAM daughter board that we developed. The NVRAM daughter board, as shown in Figure 3(a), has four slots

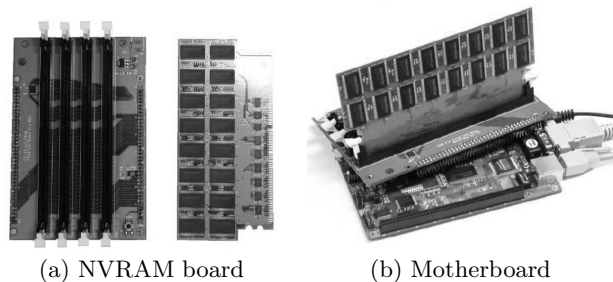


Figure 3: Real embedded board with NVRAM

with each slot capable of loading a 16MB NVRAM module. For our experiments, we use only a maximum of 140KB of NVRAM, a small percentage available in our environment. We use the state-of-the-art FeRAM chip produced by Ramtron for our NVRAM modules [1]. When the NVRAM board is attached to the mother board as shown in Figure 3(b), the NVRAM appears as part of the physical memory address space like SDRAM in the motherboard and can be accessed via memory mapped addressing in Linux. The embedded development board that we use has a PXA255 processor that is based on the XScale architecture. The board has 64MB of SDRAM and 64MB of NAND Flash memory of which a 32MB partition is used for the VFAT and NVFAT file system.

4.2 File System Workloads

For the experiments, we consider the PostMark 1.5 benchmark for generating file system workloads [10]. In our evaluation, the PostMark benchmark creates ten directories along with a set of 1,000 initial files with random sizes ranging from 512B to 16KB and executes a specified number of transactions, in our case, set to 10,000. To remove the buffering effect in the application level, our configuration does not make use of the standard I/O library. The other parameters not mentioned are set to default values. In our configuration, the PostMark benchmark generates a total footprint of 61.03MB.

We modify the benchmark so that synchronous writes could be generated with certain proportion. In the original PostMark benchmark all writes, that is, including both file data and metadata, are done asynchronously. Throughout the experiments all writes are done in 512 byte units. In our experiments, we use the original PostMark benchmark, that is, with 100% asynchronous writes, and a modified Postmark benchmark with all writes done synchronously. We refer to these benchmarks as Async-PostMark and Sync-PostMark, respectively, throughout our discussions.

5. PERFORMANCE EVALUATION

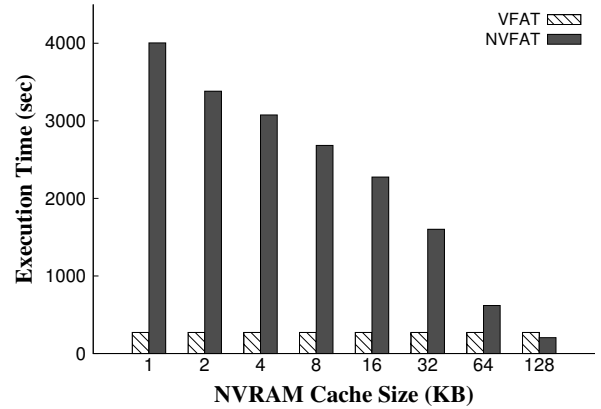
To discuss the impact of exploiting NVRAM as a metadata write cache on I/O performance in an embedded system environment, we analyze the experimental results from the file system and FTL viewpoints.

5.1 File System Viewpoint

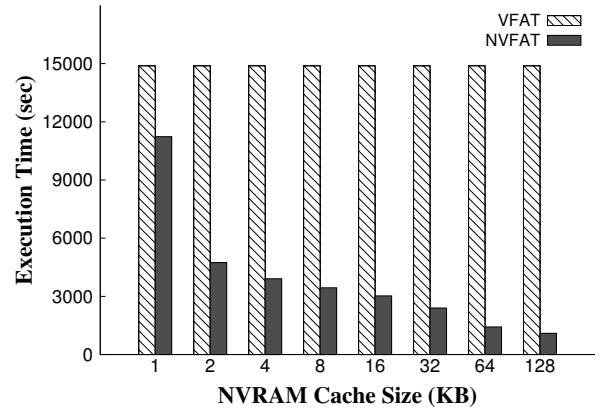
The main point of this subsection is in comparing the performance of VFAT and NVFAT. Note that in terms of maintaining the integrity of the file system, NVFAT provides full integrity at all times while VFAT does this only at synchronous write points. This said, we do not mention the integrity issue any further, but only concentrate on the performance issue.

To show the impact of using NVRAM write cache for file system metadata on the performance from the file system viewpoint, we execute the workloads discussed in the previous section on both VFAT and NVFAT. For each of these file systems, we consider the Log block FTL scheme, which is a dominant FTL scheme over Flash memory, proposed by Kim and others [12]. For the evaluation, we implement the Log block FTL scheme in Linux 2.6.21 and the number of log blocks used in the Log block FTL scheme is set to 64.

Figure 4(a) and (b) show the execution times (y -axis) observed by the application when run with NVFAT and



(a) Async-Postmark



(b) Sync-Postmark

Figure 4: Execution times as NVRAM write cache size grows (exponentially)

VFAT as the amount of NVRAM used in NVFAT is increased (x -axis) for each of the Async-PostMark and Sync-PostMark benchmark, respectively. Note that the NVRAM cache size increases exponentially, though the maximum size we consider is only 128KB. In our evaluation, the amount of NVRAM used as a percentage of the 32MB Flash memory partition ranges from 0.031% for 1KB to up to 0.4% for 128KB. These values may be significant as in FAT file systems the FAT size increases with the partition size. Recall that these are extreme situations with Async-PostMark representing a workload with all asynchronous writes and Sync-PostMark representing a workload with every write being synchronous.

We observe from Figure 4(a) that NVFAT always performs worse than VFAT for Async-PostMark except for the 128KB NVRAM cache size. This is obvious as the VFAT file system will always cache all the metadata in RAM. Note that the system memory capacity is double the size of the Flash partition in our experimental setup. This upper hand in performance, of course, is gained by sacrificing integrity.

Though the performance difference is substantial with very little NVRAM, this difference reduces dramatically with 64KB of NVRAM. Then, NVFAT eventually outperforms VFAT by 32.5% when the NVRAM cache size becomes 128KB that is large enough to absorb all the file system metadata.

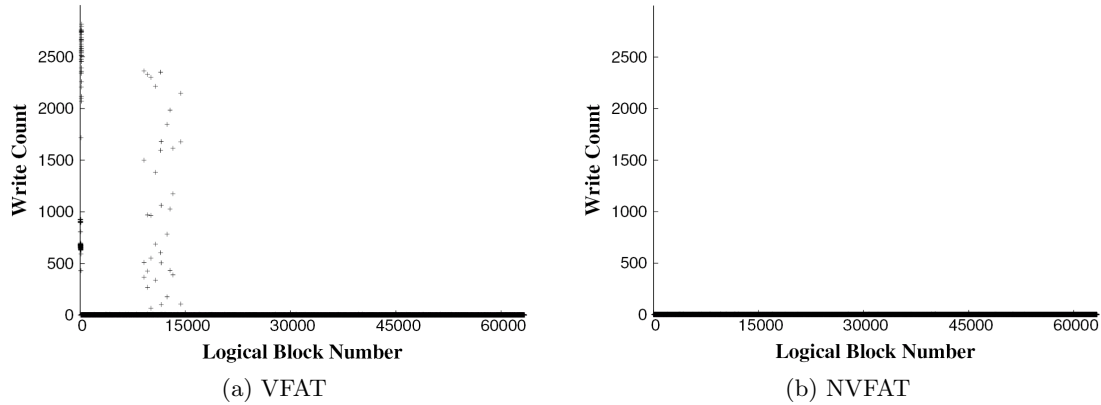


Figure 5: Write requests from file systems to FTL for Sync-PostMark

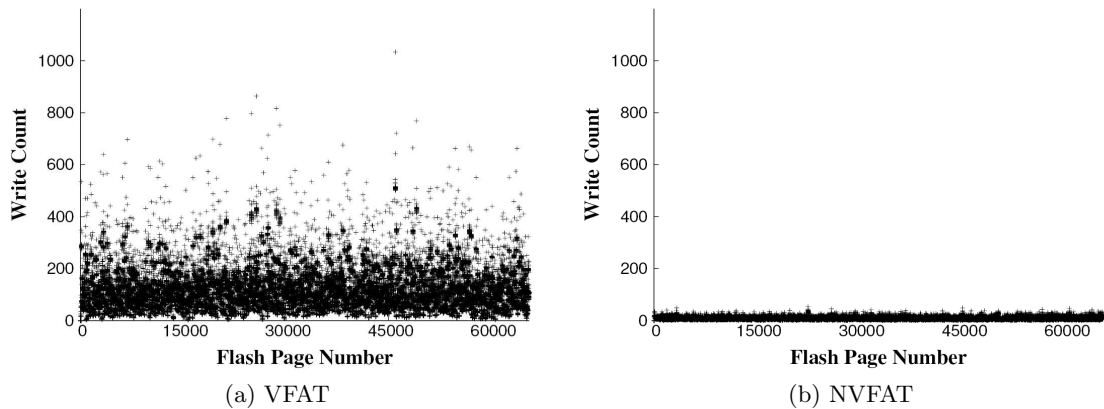


Figure 6: Write requests from FTL to Flash for Sync-PostMark

This performance improvement of NVFAT with the 128KB NVRAM cache comes from the following reason. To assure minimal file system integrity, conventional operating systems such as Linux periodically trigger synchronous write requests for all dirty data cached in volatile RAM. For this periodical synchronous writes, the VFAT file system writes all dirty metadata as well as dirty file data to nonvolatile storage. The NVFAT file system, however, writes only dirty file data cached in volatile RAM (excluding the metadata) as the metadata are already stored in nonvolatile storage, that is, NVRAM.

The results for Sync-PostMark in Figure 4(b) represent the upper bound in gains possible by employing NVRAM write cache of each size. With just 128KB of NVRAM execution time of 14,883 seconds is reduced to 1,096 seconds. Though we do not report the actual numbers here, we experimentally confirm that as the rate of synchronous writes increase the performance gains gets wider as well.

5.2 FTL Viewpoint

In this subsection, we consider the effect of NVRAM write cache on the FTL and Flash memory. Consider Figures 5 and 6 that are observations made for VFAT and NVFAT with the Sync-PostMark benchmark.

Figure 5 shows the number of write requests made for each LBN observed by the FTL for the VFAT and NVFAT

file system. We see that with VFAT there are two rising columns on the left-hand side in Figure 5(a). These columns show that numerous requests for particular LBNs are being made. These LBNs represent the metadata. For NVFAT with 128KB NVRAM cache shown in Figure 5(b) these columns disappear. The reason is that all these requests are being absorbed by the NVRAM write cache. The consequence of these results are shown in Figure 6. Figure 6 shows the number of write requests made to particular pages in Flash memory by the FTL. We observe that the number of requests made with VFAT is much higher, ranging widely from the teens to up to 1,034, than those made with NVFAT where most range below 54.

These observations tell us, essentially, that with NVFAT smaller number of writes will occur. The reduced number of writes lead to smaller number of erasures and copy operations for block cleaning. Such reductions lead to enhancements in wear leveling and energy efficiency as evidenced by the following.

Wear Leveling: Figure 7 presents the average erase count and the standard deviation shown as an error bar for all blocks. These results are measured during the experiments for the Sync-PostMark benchmark in Figure 4. We observe that the average as well as the standard deviation decreases as NVRAM write cache increases. The fact that the standard deviation is reduced as well tells us that wear

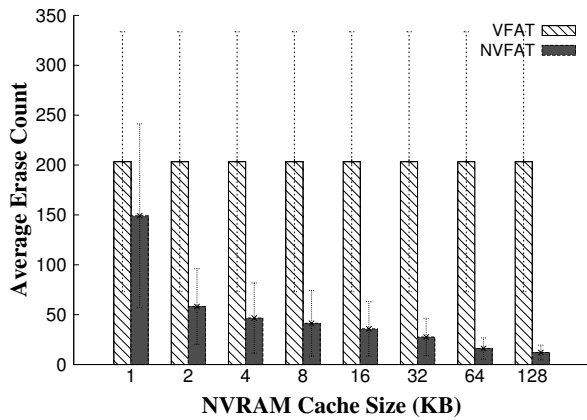


Figure 7: Average erase count (per block) and its standard deviation for Sync-Postmark

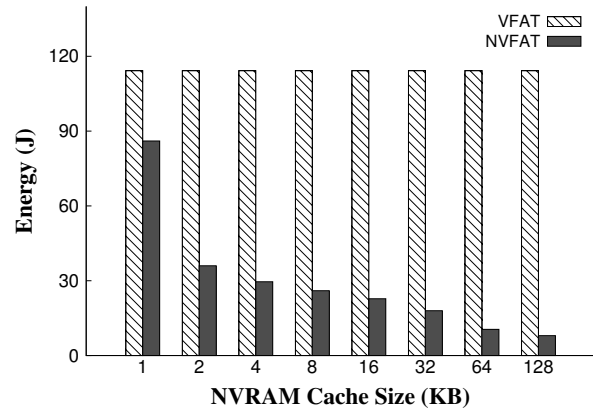


Figure 8: Energy consumed for Sync-Postmark

leveling improves with NVFAT despite the fact that no extra wear leveling effort is being made.

Energy Efficiency: Energy consumption at the Flash memory level may be calculated using models provided by previous studies [8]. In our study, we make similar calculations of energy consumed by taking the read and program count (in sector units), and erase count (in block units) and multiplying these numbers by the energy consumed by each operation. Figure 8 shows the energy consumed (in Joules) as a function of the NVRAM write cache size. The improvements are substantial. The energy consumed is dramatically reduced from 114.2J with VFAT to 7.9J with NVFAT when the NVRAM cache size is 128KB.

6. SUMMARY AND FUTURE WORK

In this paper, we considered the use of Non-Volatile RAM (NVRAM) as a write cache for metadata of the file system in embedded systems. The results show that substantial reductions in execution time are possible from an application viewpoint when adopting the NVRAM write cache. Another important consequence of the write cache that we find is its benefit at the FTL layer, leading to improved wear leveling of Flash memory and increased energy savings, which are important measures for embedded systems.

Though we have shown the potential of making use of NVRAM, the work here is still preliminary. There are still many issues that could be considered to enhance performance. As mentioned earlier, development of better destaging schemes is necessary. Another issue is that of retaining dirty file data in the NVRAM write cache as well. Though we only considered VFAT in this study, there is no reason to limit the deployment of NVRAM write cache to this file system. Finally, we hope to apply the technique presented in this study to real world systems and applications.

7. REFERENCES

- [1] Ramtron International, <http://www.ramtron.com>.
- [2] Freescale Semiconductor, <http://www.freescale.com>.
- [3] Technology Review, <http://www.technologyreview.com/Infotech/20148>.
- [4] BGET Memory Allocator, <http://www.fourmilab.ch/bget>.
- [5] M. Baker, S. Asami, E. Deprit, J. Ouseterhout, and M. Seltzer. Non-Volatile Memory for Fast, Reliable File

- Systems. In *Proceedings of the 5th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-V)*, pages 10–22, 1992.
- [6] P. M. Chen, W. T. Ng, S. Chandra, C. Aycocck, G. Rajamani, and D. Lowell. The Rio File Cache: Surviving Operating System Crashes. In *Proceedings of the 7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)*, pages 74–83, 1996.
- [7] I. H. Doh, J. Choi, D. Lee, and S. H. Noh. Exploiting Non-Volatile RAM to Enhance Flash File System Performance. In *Proceedings of the 7th ACM & IEEE International Conference on Embedded Software (EMSOFT'07)*, pages 164–173, 2007.
- [8] Y. Du, M. Cai, and J. Dong. Adaptive Energy-Aware Design of a Multi-Bank Flash-Memory Storage System. In *Proceedings of the 11th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA'05)*, pages 311–316, 2005.
- [9] B. S. Gill and D. S. Modha. WOW: Wise Ordering for Writes . Combining Spatial and Temporal Locality in Non-Volatile Caches. In *Proceedings of the 4th USENIX Conference on File and Storage Technologies (FAST'05)*, pages 129–142, 2005.
- [10] J. Katcher. PostMark: a New Filesystem Benchmark. Technical Report TR3022, Network Appliance, 1997.
- [11] H. Kim and S. Ahn. BPLRU: A Buffer Management Scheme for Improving Random Writes in Flash Storage. In *Proceedings of the 6th USENIX Conference on File and Storage Technologies (FAST'08)*, pages 239–252, 2008.
- [12] J. Kim, J. M. Kim, S. H. Noh, S. L. Min, and Y. Cho. A Space-Efficient Flash Translation layer for CompactFlash Systems. *IEEE Transactions on Consumer Electronic*, 48(2):366–375, 2002.
- [13] K. H. Lee, I. H. Doh, J. Choi, D. Lee, and S. H. Noh. Write-Aware Buffer Cache Management Scheme for Nonvolatile RAM. In *Proceedings of the 3rd Conference on IASTED International Conference (ACST'07)*, pages 29–35, 2007.
- [14] E. L. Miller, S. A. Brandt, and D. D. E. Long. HeRMES: High-Performance Reliable MRAM-Enabled Storage. In *Proceedings of the 8th Workshop on Hot Topics in Operating Systems (HOTOS-VIII)*, pages 95–99, 2001.
- [15] A.-I. A. Wang, G. Kuenning, P. Reiher, and G. Popek. The Conquest File System: Better Performance Through a Disk/Persistent-RAM Hybrid Design. *ACM Transactions on Storage*, 2(3):309–348, 2006.