Addressing Scalability and Consistency Issues in Hybrid File System for BPRAM and NAND Flash

Quan Taizhong Jinsoo Yoo Jaemin Jung Youjip Won
Dept. of Electrical and Computer Engineering
Hanyang University, Seoul, Korea
{coolquan | jedisty | jmjung | yjwon}@ece.hanyang.ac.kr

Abstract

Recently, a number of work suggested to use Byte-Addressable NVRAM(BPRAM) to address the drawbacks of NAND Flash based file system. These works use BPRAM to reduce write latency of NAND Flash, or to reduce the mount latency of log-structured file system for NAND Flash. Our work is aligned with the preceding works in that we develop a hybrid file system technique for NAND Flash and BPRAM. In this work, we address the scalability issue of BPRAM in a hybrid file system. Given the current state of art device technology, e.g. NAND Flash and FRAM, we believe that BPRAM size and NAND Flash size should be 1:1000 in hierarchical storage system and file system structure should be designed to satisfy this storage hierarchy. In our hybrid file system, BPRAM is responsible for harboring metadata in the file system. In the current widely used file system for NAND Flash, there are 16 Byte metadata for 512 Bytes of data (BPRAM to NAND size ratio is 3:100). To address the discrepancy in scalability between BPRAM and NAND Flash, we develop mount related compact metadata data structure for BPRAM and hybrid compression *technique which is specifically tailored for compressing file metadata for log-structured file system. We develop a flexible yet efficient data structure to represent a set of compressed metadata while effectively coping with size variability caused by compression. As a result, we dramatically reduce the file metadata size. We implement the file system in PXA320 core with Linux 2.6. The file system mount time and file system recovery time decrease by order of magnitude.*

Keywords*: NAND Flash, BPRAM, Hybrid File System, Hierarchical Storage, LFS, Metadata Compression*

1. Introduction

1.1. Motivation

NAND flash is read or written in page units, but erased in block units that consists of many pages. This inconsistency of operation unit makes it hard to implement effective storage system. Compared to NAND flash memory, byte-addressable Persistent RAM (BPRAM) has advantages in durability, byte-addressability, and speed. Moreover, unlike from flash memory, overwrite is allowed in BPRAM. Although BPRAM has a great potential, BPRAM leaves much to be desired to be used as data storage due to its scale and price. Recently, a number of work suggest the usage of BPRAM to address the drawbacks for NAND Flash storage device: mount latency of log-structured file system for NAND Flash[9, 4], recovery latency of log structured file system [16, 3, 10], reliability of solid state drive [13] and etc. While all these works seem to nicely resolve the drawbacks of the NAND Flash file system, they overlook one important fact that *BPRAM is orders of magnitude smaller than NAND Flash*. In particular, the size of state of art FRAM and NAND Flash device is 64 Mbit and 8 Gbit, respectively. In this work, we develop hierarchical file system for hybrid storage where BPRAM harbors metadata for file and file system to reduce mount latency and recovery overhead. To make the storage cost-effective, we aim at making the BPRAM size and NAND flash size ratio 1:1000.

1.2. Related Work

Since NAND flash memory does not support in-place update, file systems for flash memory adopt log-structured approach. The log-structured flash file systems build the structures of a file system in the main memory at mount time [14]. The in-memory structures include file metadata, directory structure, etc. To construct the in-memory structures, a file system scans the NAND flash memory. Since

the scan operation covers most of the NAND flash devices's operations, it takes much more time to complete. Therefore, flash file systems of early days suffer from the significantly long mount latency.

A number of file systems have been proposed to reduce the mount latency. Yim et al. [23] and Bityuckiy [3] proposed a snapshot to improve the mount speed. In their proposal, file systems locates the snapshot in a certain area of NAND flash. Instead of scanning the flash memory, they read snapshots to mount a file system. Alike the snapshot, a checkpoint is used in the recent version of YAFFS. However, the snapshot and checkpoint require additional time to be written in the NAND flash memory at unmount time. Moreover, scanning the file system partition is required to prepare the in-memory structures, when the checkpoint or snapshot is incorrect.

RFFS [15] stores the page data and page metadata separately. Even though it may reduce the mount latency, mount latency is still proportional to the size of flash memory. Wu et al. [22] proposed a method for efficient initialization and crash recovery for flash-memory file system. MNFS [10] uses NAND Flash block as basic unit to maintain metadata to improve the file system mount latency.

Some of the recent works proposed a file system for BPRAM and hard disk drive. The conquest file system [21] is a disk/persistent-RAM hybrid file system. It improves performance by storing metadata and frequently accessed small files in persistent RAM. MRAMFS [6] is a hybrid system that stores small files and metadata in MRAM. It proposes to compress the data and metadata when writing to BPRAM. It compared several compression algorithms and showed that LZO [2] is suitable for files of large size. HeRMES [12] realizes that more than 50% of the file system accesses are metadata accesses. They use NVRAM as metadata storage and write buffer. FRASH [9] uses BPRAM as storage of metadata in flash file system. It uses "Copy-On-Mount" technique to speed up the mount procedure. It stores in-memory data and on-disk data in BPRAM and copies the in-memory data into the main memory during mount phase. It regularly synchronizes in-memory data to BPRAM. When the system crashes, FRASH has to read the on-disk structure region of NVRAM and reconstructs the in-memory data structure region in the NVRAM. MiNVFS [4] maintains all the metadata in NVRAM, while storing all file data in flash memory. PFFS [17] stores metadata of file system in PRAM and constructs double indirect index structure for data page management. FRASH, MiNVFS and PFFS still require significant amount of BPRAM to maintain a file metadata. Our earlier work, CMFS(Compressed Metadata File System)[18] proposes compressing file metadata to maintain it in BPRAM. In CMFS, however, the size of the metadata reduces only by 10% - 15%. A set of all metadata in the file system is compressed and decompressed as a single unit. This makes the synchronization of metadata between DRAM and BPRAM very slow.

2. Background

2.1. Byte-addressable Persistent RAM

Byte-addressable Persistent RAM (BPRAM) is a byte-addressable and non-volatile memory. With these functional characteristics, the read/write speed of BPRAM is expected to be close to that of DRAM. There are many kind of BPRAMs: PRAM (Phase Change RAM) [11], MRAM (Magnetostrictive RAM) [20], RRAM (Resistive RAM) [7], and FRAM(Ferro-electric RAM) [5]. Table 1 summarizes the characteristics of BPRAM technologies and conventional memory technologies. Each BPRAM technologies has its own pros and cons. MRAM has the best read/write performance while it has its disadvantages in cost and capacity. The largest MRAM which current state of art technology can afford is 16MBit [8]. The size of PRAM [19] is much bigger than MRAM and FRAM. However, the endurance is limited and the power consumption is higher than other BPRAMs. Among the current non-volatile memory technologies, FRAM is the most matured technology.

2.2. Log-structured File System for Flash Memory

NAND flash memory cannot be written without a erase operation. Moreover, the unit of the erase operation is much larger than the unit of write operation. To erase safely, a file system has to ensure that every data which will be deleted is useless. To address these issues, there exist a FTL approach and a file system approach. A FTL (Flash Translation Layer) provides a conventional block device abstraction on the NAND flash memory. If the FTL is provided, existing file systems for hard disks can use NAND flash memory without any modification. However, FTL requires an additional circuit or computation overhead. Our work is based on the file system approach. In this approach, a file system itself handles the characteristic of NAND flash memory. Flash file systems use the log-structured approach to treat NAND flash memory. A log-structured file system writes everything on the append-only log, it never overwrites existing data. Thus, every data is scattered over the log without a strict rule. When a file access is requested, a file system needs to find a metadata of the file. However, a file system cannot directly find the location of the metadata because it is just appended before. To read the metadata, it is required to scan the whole log. To eliminate scanning the log for every request, a log-structure file system maintains metadata in the memory when it is operating. Nevertheless, scanning the whole log is required to mount the file system

Item	DRAM	FRAM	PRAM	MRAM	NOR	NAND
centering Byte Addressable	YES	YES	YES	YES	Read only	NO
Non-volatile	NO	YES	YES	YES	YES	YES
Read	10ns	70ns	68ns	35ns	85ns	15us
Write	10ns	70ns	180ns	35ns	6.5us	200us
Erase	none	none	none	none	700ms	2ms
Power consumption	High	Low	High	Low	High	High
Capacity	High	Low	High	Low	High	Very High
Endurance	10^{15}	10^{15}	$> 10^7$	10^{15}	100K	100K
Prototype Size		64Mbit	512Mbit	16MBit		

Table 1: Comparison of Non-volatile RAM Characteristics [9]

because there is no metadata for the file system in the memory. The mount time can be excessively high, because the scanning time increases in proportion to the partition size. This excessive time consumption is a significant problem in the embedded system and mobile products. Whenever the system reboots, large amount of time is used in mounting a file system.

3. Structure of Compressed Metadata File System

In this work, we develop a hierarchical file system, CMFS (Compressed Metadata File System) for BPRAM and NAND Flash. CMFS at its inception stage has been introduced in our earlier work [14], which is a minor modification to YAFFS [18]. CMFS proposed in this paper is completely overhauled since then. We develop hybrid coding algorithm which well exploits the characteristics of file system metadata, checkpoint based file system recovery scheme, and more elaborate data structure for hybrid storage.

3.1. Design of CMFS file system

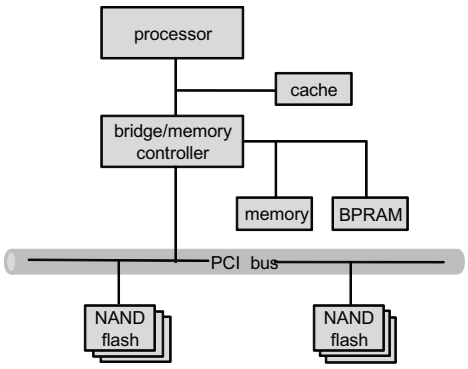


Figure 1: Hierarchical Storage Organization

With conventional I/O interface, the byte-addressability and fast access latency are limited to the slower I/O bus. To fully exploit the characteristics of BPRAM, we assume that BPRAM is exposed to the system bus in our work. NAND flash memory is connected to the I/O bus and CMFS accesses NAND flash memory with a MTD driver. A MTD driver directly reads or writes NAND flash memory without FTL. Overall organization is illustrated in figure 1.

Table 2 shows the characteristics of various hybrid file systems including CMFS. Existing file systems require huge size of BPRAM which is very expensive. There are 32MB spare area for a 1GB NAND flash. If the size of NAND flash memory is increased to 1,000GB, 32GB of spare area is required. Because flash file systems uses spare area to store metadata of file system, storing metadata of file system in BPRAM is not practical. MiNVFS [4] requires 18MB of BPRAM for 32MB flash memory in case that the file system has 65,536 files which are 512bytes in size. Although MRAMFS compresses the metadata, it requires huge size of BPRAM. To reduce the required amount of BPRAM, CMFS eliminates needless portion from the original metadata of the flash file system. Unfortunately, it can not sufficiently reduce the amount of metadata. To minimize the required amount of BPRAM, CMFS compresses the metadata. We carefully select compression algorithm against various kinds of data and develop hybrid coding algorithm to efficiently compress metadata in BPRAM. Compared with the other file systems, CMFS requires the smallest size of BPRAM. The details of compression method are explained in next section.

Existing hybrid file systems do not provide a recovery method against the system crash. When a system shuts down abnormally, file systems cannot ensure that the metadata in BPRAM is correct. If the corruption of metadata in BPRAM is detected, file systems cannot use them without recovery. Even though BPRAM is used, a file system has to scan the whole area of NAND flash to reorganize the metadata into the BPRAM. CMFS provides a recovery method that doesn't require scanning the whole NAND flash. When

File system	In-memorydata	Compress metadata	synchronization	Crash recovery
MiNVFS [4]	●			
FRASH [9]	●		●	
MRAMFS [6]		●		
CMFS	●	●	●	●

Table 2: Characteristics of file systems

the corruption of metadata in BPRAM is detected, CMFS recovers it and uses it in mounting the file system partition. Because this eliminates the need of scanning the NAND flash memory, CMFS provides a fast mount speed in case that the system shuts down incorrectly. To support the recovery method efficiently, we develop synchronization and update methods for CMFS.

3.2. File system objects

CMFS defines three kind of compressed metadata in the BPRAM, called BPRAM metadata: $CMFS\_validitymarker$, $CMFS\_device$, and $CMFS\_object$. $CMFS\_validitymarker$ is used to identify whether the BPRAM_metadata is valid. $CMFS\_device$ stores the statistical information about the file system partition. Statistical information is divided into partition statistical information and block statistical information. The partition statistical information contains the overall statistic information on the file system partition: number of free pages, number of allocated pages, etc. The block statistical information is maintained for each blocks in NAND flash memory. It contains the number of free pages and the number of allocated pages for each block. Each physical block has a corresponding block information in the BPRAM. Every block has a sequence number. We can find it in the block statistical information. Block has three states: empty, full, and allocated. When the block is empty, the sequence number of it is 0. A unique sequence number is given to every block before it is used. If the block is not empty, its sequence number is larger than 8. The block which is being allocated has the biggest sequence number.

The size of $CMFS\_device$ is determined by the file system partition and not changed. There is one $CMFS\_object$ per file or directory in BPRAM. It is used to construct a file and directory structures in CMFS. It contains the file information including id of its own and the parent directory. In addition, $CMFS\_object$ has PAT information (Physical Address Translation), which is used to translate the file offset to the physical page number. The size of $CMFS\_object$ depends on the file size which determines the number of PAT structures. PAT structures are loaded into the memory and rebuilt at a tree at mount time. The PAT structure in the $CMFS\_object$ has an additional four bytes to represent a

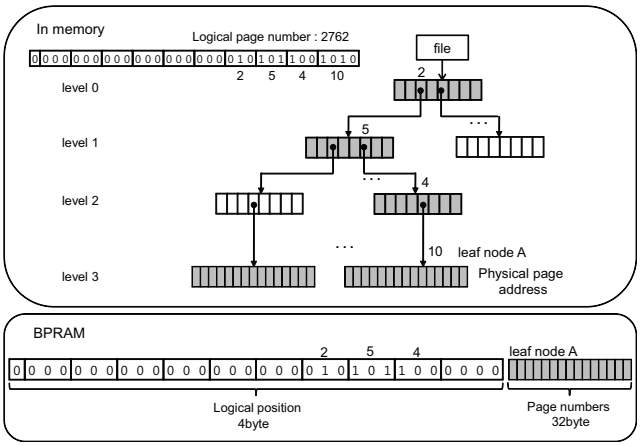


Figure 2: PAT structure in memory and BPRAM

location in the PAT tree. As shown in figure 2, the first four bytes represent the position in the PAT tree and next 32 bytes direct to leaf node of the PAT tree. Therefore, the size of PAT information in the BPRAM is 36 bytes.

Figure 2 illustrates the structure of the PAT tree. The leaf node of PAT stores 16-bit physical page numbers while the other nodes consists of eight node pointers with size of 32 byte. Assume that CMFS accesses the data with logical page number of 2762. The lowest four bits indicates the position of physical page number in leaf node. Starting from the fifth LSB, every three bits are the index for the node. PAT tree is traversed from the root node to get a physical page number associated with the logical address 2762. File system knows levels of PAT from the metadata and determines which node is leaf node.

$CMFS\_device$ and $CMFS\_object$ are much smaller than in-memory data structures. For example, in-memory device information is 3648bytes while device information in $CMFS\_device$ is 40bytes. In-memory object size is 128bytes while $CMFS\_object$ is 28bytes. Considering that the size of BPRAM is tiny, we also compress the BPRAM metadata. The structures of BPRAM metadata are shown in figure 3.

CMFS_validitymaker 8bytes

field type	field name	field size
unsigned int	SCM_valid	4bytes
unsigned int	data_valid	4bytes

CMFS_object 24bytes + 36bytes * n

field type	field name	size
int	structType	4bytes
unsigned int	objectId	4bytes
unsigned int	parentId	4bytes
int	hdrChunk	4bytes
enum	variantType:3	2bytes
char	deleted:1	1byte
char	softDeleted:1	
char	unlinked:1	
char	fake:1	
char	renameAllowed:1	
char	unlinkAllowed:1	
char	serial	4bytes
int	nDataChunks	4bytes
array	PAT	(36bytes*n)

CMFS_device 40bytes + 8bytes * nblock

field type	field name	field size
partition information 40bytes		
int	sturctType	4bytes
int	nErasedBlocks	4bytes
int	allocationBlock	4bytes
unsigned int	allocationPage	4bytes
int	nFreeChunks	4bytes
int	nDeletdFiles	4bytes
int	nUnlinkedFiles	4bytes
int	nBackgroundDeletions	4bytes
unsigned int	sequenceNumber	4bytes
unsigned int	oldestDirtySequence	4bytes
block information 8bytes * nblock		
int	softDeletions:10	4bytes
int	pageInUse:10	
unsigned int	blockState:4	
int	needsRetiring:1	
int	skipErasedCheck:1	
int	gcPrioritise:1	
int	chunkErrorStrikes:3	
int	hasShrinkHeader:1	
int	sequenceNumber	4bytes

Figure 3: BPRAM data structure

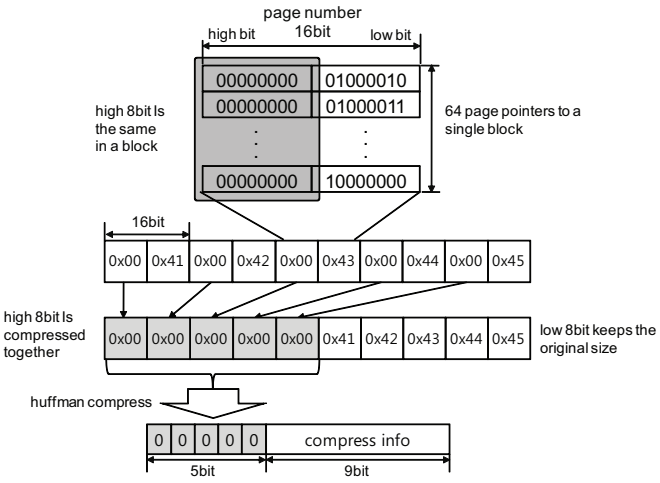


Figure 4: Hybrid coding algorithm

4. Hybrid Coding

CMFS compresses the data structures in BPRAM, e.g. $CMFS\_validitymarker$, $CMFS\_device$ and $CMFS\_object$, to reduce the required size of BPRAM. To efficiently compress, we carefully select compression algorithm for each data structures. We considered five compression algorithms: huffman [1], rice [1], shannonfano [1], run length encoding [1], and LZO [2]. Because the size of $CMFS\_validitymarker$ is only 8-byte and fixed, CMFS does not compress it. Most portion of BPRAM is used to store $CMFS\_device$ and $CMFS\_object$ rather than $CMFS\_validitymarker$. We focus our effort in finding optimal compression algorithm for $CMFS\_device$ and $CMFS\_object$. The size of $CMFS\_device$ is determined by the partition size. In our inspection, all compression algorithms have good compression ratios in compressing $CMFS\_device$. Among the five compression algorithms, the LZO [2] algorithm provides the best performance. It means that the LZO algorithm faces the least performance overhead. Thus, CMFS uses the LZO compression algorithm to compress $CMFS\_device$.

From the compression ratio's perspective, it is very important to effectively compress the PAT structure in $CMFS\_object$. We examine the compression ratio of five compression algorithms in compressing $CMFS\_object$. None of these show satisfying result. In this work, we carefully examine the characteristics of PAT structure and developed hybrid coding which is specifically tailored for the compression of $CMFS\_object$. PAT structure contains physical page numbers (16 bits). Within a file, we find that higher 8-bits of the page numbers are identical in most cases. This is because the pages in the same NAND flash block have the same higher 8-bits value. Further, in YAFFS, which is of baseline file system for CMFS, a file system allocates free pages from the allocation blocks.Pages in the allocation block are sequentially allocated to the write requests. So, in hybrid coding, we apply huffman compression algorithm to compress the higher 8-bits. Because lower 8-bits of each page in PAT are unique, it cannot be compressed further. Therefore, we do not apply compression for lower 8-bits value. Figure 4 illustrates the example of hybrid coding.

5. Mounting File System

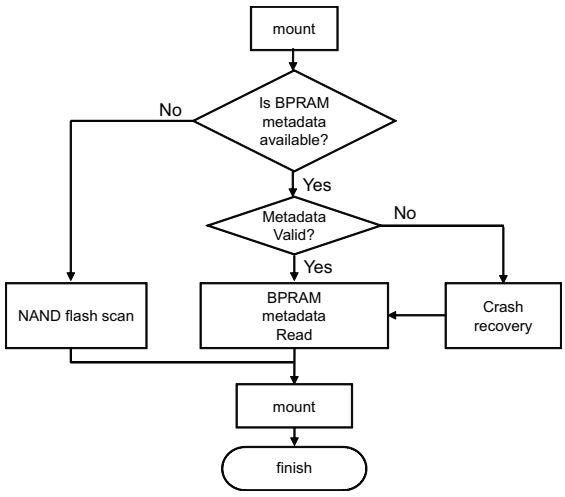


Figure 5: Procedure of mount

CMFS stores compressed metadata of the file system in BPRAM. To construct the in-memory data structures, CMFS reads and decompresses BPRAM metadata at mount time. Since BPRAM is much faster than NAND flash mem-

ory and the CMFS compresses the metadata which is stored in the BPRAM, time to read the file system metadata from the memory device diminishes. Although the decompression overhead exists at mount time, the overall mount latency is dramatically reduced.

Figure 5 illustrates the possible mount scenarios of CMFS. BPRAM metadata is available if and only if it can be decompressed without any errors. The BPRAM metadata can be unavailable when the CMFS cannot decompress the BPRAM metadata, for example if the system crashes while updating the compressed BPRAM metadata. In this case, decompression of the BPRAM metadata is impossible. To reconstruct the BPRAM metadata, it is required to scan the NAND flash memory. When the BPRAM metadata is available, the CMFS checks the $CMFS\_validitymarker$ to verify that BPRAM metadata is valid. If BPRAM metadata is valid, the CMFS reads the compressed metadata from the BPRAM and decompresses it. The mount of CMFS completes without additional accesses to the NAND flash memory. In case the BPRAM metadata is available out invalid, CMFS performs crash recovery to restore the BPRAM metadata. CMFS decompresses BPRAM metadata and restore it according to the metadata in the NAND flash. To reduce accesses to the NAND flash memory in crash recovery, CMFS reads only the essential portion from the NAND flash memory. Detailed description about crash recovery is discussed in section 7.

CMFS reads and decompresses BPRAM metadata only at the mount time. Since every file metadata exists in the main memory after the mount, every file I/Os are handled with the in-memory metadata structure. However, the compression of metadata is performed if any modifications to the file system are triggered. We discuss the BPRAM metadata update and compression in the next section.

6. Metadata Update Method

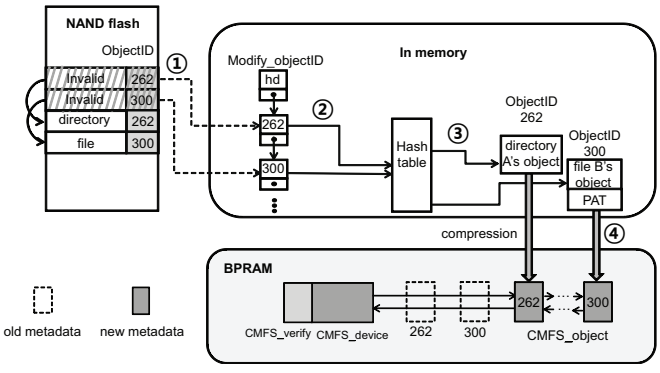


Figure 6: Update method

When any changes are induced on the CMFS, the changes are applied into the NAND flash memory. However, CMFS does not update the BPRAM metadata at the same time. Since BPRAM metadata update includes compression overhead, frequent updates would result in the significant performance drop. CMFS employs a lazy update policy to compress and update the metadata in the BPRAM.

Figure 6 illustrates the case when pages of directory A and file B are modified. When requests occur, CMFS marks $CMFS\_validitymarker$ as invalid. The $CMFS\_validitymarker$ is used to represent any updates that are not adopted in BPRAM metadata yet. Then, a new page is written into the NAND flash and the old pages are marked invalid. After NAND flash memory is updated, the objectID of the modified object is stored in $Modify\_objectID$ and requests are finished. $Modify\_objectID$ is the in-memory data structure that maintains the objectID of modified objects. To update the BPRAM metadata, the system periodically checks if $Modify\_objectID$ exists. In our implementation, it is performed every five seconds. If it exists, the $CMFS\_objects$ which corresponds to the objectid of $Modify\_objectID$ is reconstructed and compressed in the memory. The previous $CMFS\_objects$ is deleted and the area is reclaimed. The in-memory object contains the pointer to a previous $CMFS\_objects$. The system allocates reconstructed $CMFS\_objects$ in BPRAM. Then, the pointer in the in-memory object is updated to point to the new $CMFS\_objects$. At last, the $CMFS\_device$ is rebuilt and the $CMFS\_validitymarker$ is set to valid.

7. Crash Recovery

When the system crashes unexpectedly, it is required to verify the compressed metadata in BPRAM at mount time. As described in section 5, we need to scan all metadata from the NAND flash device when compressed metadata is unavailable. If the BPRAM metadata is available, CMFS check if it is vaild. The BPRAM metadata update of CMFS ensures that the BPRAM metadata is valid only when all updates are performed on the NAND flash memory and the BPRAM. Consequently, the CMFS does not need to access NAND flash device to build in-memory structures unless BPRAM metadata is invalid. However, in case that BPRAM metadata is invalid, it is required to restore the BPRAM metadata to a valid status. Because of unavailability of in-place updates, the old pages are always maintained in the NAND flash memory. In addition, the NAND flash is updated before the BPRAM. Therefore, the CMFS recovers the BPRAM metadata according to the metadata in NAND flash memory.

It is very time consuming to read every metadata from the NAND flash device. To avoid unnecessary accesses to the NAND flash memory, the CMFS uses a sequence

number in the BPRAM, described in section 3.2. The sequence number is also maintained in the NAND flash memory. When any block is written or erased, the sequence number of the block is updated in the NAND flash memory and the number is increased. However, the sequence number in the BPRAM is modified when BPRAM metadata updated. Thus, the CMFS can find which block has been changed after the last BPRAM metadata update by comparing the sequence numbers of the NAND flash memory and the BPRAM. To obtain the sequence numbers from the NAND flash memory, the CMFS scans the sequence numbers of every blocks from the NAND flash memory. Because there is one sequence number per block, the CMFS accesses the NAND flush memory once per block.

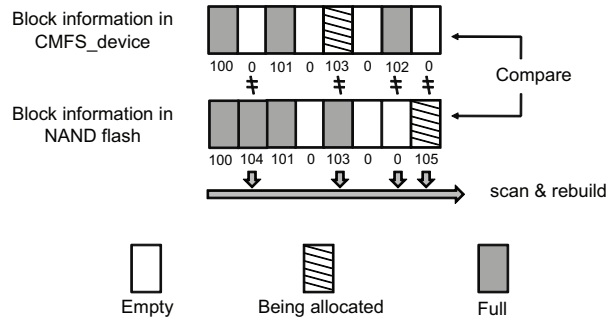


Figure 7: Compare sequence number

Figure 7 shows the procedure for comparing the block information in the CMFS and the NAND flash. The number below each block is the sequence number of each block. There are three block states: empty, full or being allocated. The sequence number in the BPRAM is inconsistent with the NAND flash. There are three kinds of inconsistencies in this figure. First, when a new block is allocated after the last update, the sequence number is zero in the BPRAM, while the sequence number in the NAND flash is not. The blocks whose sequence number in the NAND flash is 104 and 105 are belong to this case. Second, blocks can be erased by the garbage collection after BPRAM metadata update. In BPRAM, the sequence number is not zero, but it is zero in NAND flash (block sequence number is 102 in BPRAM). We should modify the statistical information but we do not need to scan the block because it is empty. Third, there can be an inconsistency even though the sequence number is the same in the BPRAM and the NAND flash. If the sequence number is the same in the BPRAM and the NAND flash, the block has not changed since the last successful update. However, there can be a special inconsistency case if the system crashes while the status of the block information in the BPRAM is being allocated(the block whose sequence number is 103 in the NAND flash). To figure out this inconsistency, the CMFS compares the status of a block as well as its sequence number. At last, we can find that there are four blocks which have been changed after the last update in figure 7. We call the four blocks the recovery area and we define the size of this area as recovery size in this paper.

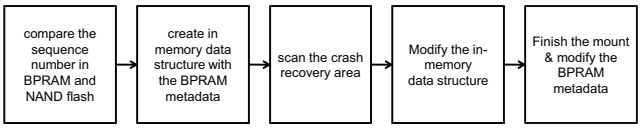


Figure 8: Crash recovery

After comparing the sequence numbers, the CMFS creates the in-memory data structure with the invalid BPRAM metadata. Then, the CMFS modifies it by scanning the NAND flash memory in the recovery area. When the partition is successfully mounted, the BPRAM metadata is finally updated. The overall procedure of crash recovery is shown in figure 8. The crash recovery of the CMFS requires only a few accesses to the NAND flash memory since the CMFS scans only pages in the recovery area.

YAFFS, one of the state-of-the-art flash file system, stores the checkpoint in the NAND flash memory. If the checksum of the checkpoint is correct, the mount can be performed rapidly. However, if crash occurs unexpectedly, the checksum will be incorrect since the checkpoint is written at unmount or explicit sync(). The failed checkpoint is ignored and YAFFS starts scanning NAND flash memory to mount a file system. Consequently, the mount time of the YAFFS is proportional to the partition size even though it supports checkpoint.

8. Experiment

8.1. Experimental Environment

We developed the CMFS on an embedded system with Marvell PXA320 (806MHz), 128MB SDRAM, and 128 MB large block NAND flash memory. We modified the YAFFS to implemented the CMFS on the linux kernel 2.6.24. A portion of DRAM is used as the BPRAM area, since BPRAM is not available in our embedded system environment.

8.2. Compression Rate

We compared our hybrid coding with existing compression algorithms, e.g. huffman [1], rice [1], shannonfano [1], run length encoding [1] and LZO [2]. We tested 10 different files and calculated the average compression rates. Figure 9 illustrate the compression rates of various compression algorithms. the X-axis and the Y-axis represent file sizes and the compression ratio, respectively. When the file size is bigger than 16KB, the performance of our hybrid coding is

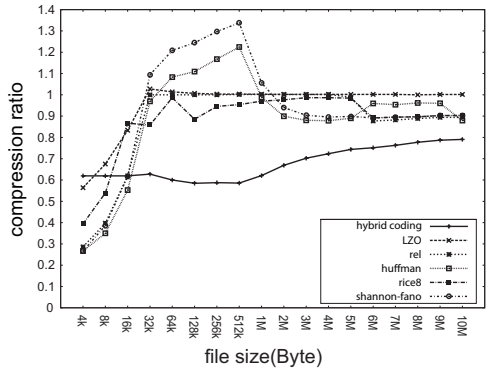


Figure 9: Compress rate

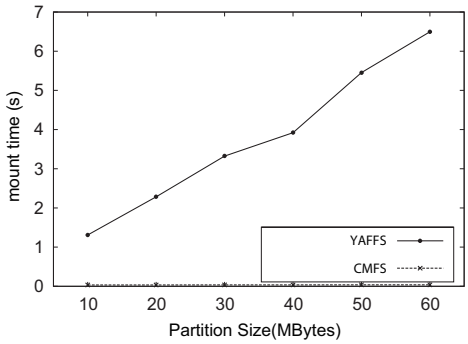


Figure 10: Mount Latency

better than others (figure 9). Although the efficiency of our hybrid coding decreases as the file size increases, the compression rate of our hybrid coding is still better than others.

8.3. Mount Latency

We measured the mount latency of the CMFS and YAFFS [14] when the BPRAM metadata is valid varying percentages of NAND flash utilization (figure 10). In YAFFS, as the usage space of the NAND flash increases, the mount latency linearly increases. This is because the spare area we need to scan increases with the usage space of the NAND flash. In the CMFS, we only need to scan the super block and the BPRAM area to mount the file system partition. CMFS significantly reduces the mount latency.

Figure 11 illustrates the LZO compression time of $CMFS\_device$ during mount time. We measured the compression times for different partition sizes. Since $CMFS\_device$ contains block information, its size is proportional to the partition size. Hence, the LZO compression time increases as the partition size increases.

When the partition size is fixed, the size of $CMFS\_object$ is a significant factor in the mount time. The size of $CMFS\_object$ influences the reading

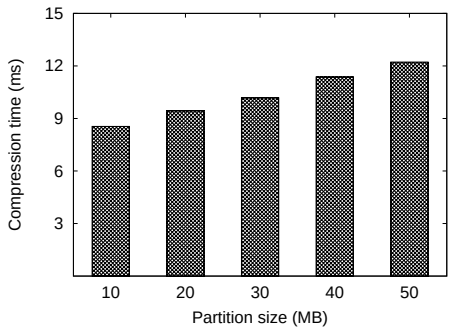


Figure 11: LZO compression time

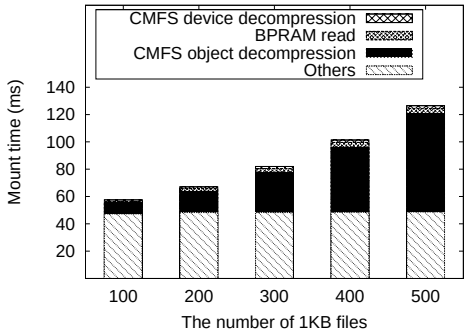


Figure 12: Mount time varying the number of 1-KB files

time of compressed metadata from the BPRAM as well as the decompression time of $CMFS\_object$ We measured the mount time varying the number of 100 KB files in 50 MB partition. Figure 12 illustrates the result. As the size of $CMFS\_object$ increases, the compression overhead increases linearly.

8.4. Recovery

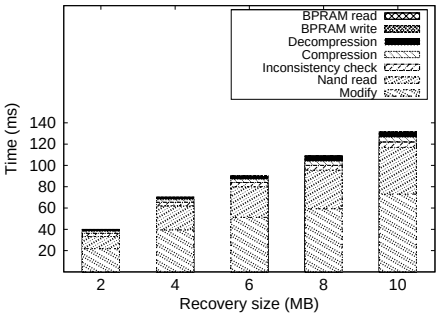


Figure 13: Detailed recovery time

We measured the recovery time of a 50 MB NAND flash partition varying the recovery size. The recovery is triggered in the mount procedure when

$CMFS\_validitymarker$ is set as invalid. Figure 13 illustrates the detailed time of the recovery time. The recovery time consists of followings: $BPRAMread$ (read BPRAM metadata from BPRAM), $BPRAMwrite$ (write BPRAM metadata to BPRAM), $decompression$ (decompress BPRAM metadata), $compression$ (compress BPRAM metadata), $inconsistencycheck$ (compare sequence numbers between BPRAM and NAND flash memory), $NANDread$ (read block information and scan inconsistent blocks from NAND flash memory), and $modify$ (modify BPRAM metadata as up-to-date). The recovery time increases as the recovery size increases. In figure 13, the BPRAM read/write time is tiny (<1%). 83% to 89% of the recovery time is consumed by NAND read and modify time. The compression/decompression and inconsistency check takes up 10%–15%.

In figure 12, the mount latency of the CMFS is about 126ms when 500 1KB files exist. In this case, recovery time requires 131ms when recovery size is 10 MB. A total of 257ms is required when the recovery size is 10 MB and 500 1KB files exit. It is still much less than the mount time of YAFFS. Thus, CMFS significantly improves the mount speed, even when BPRAM metadata is invalid.

8.5. Update Time Measurement

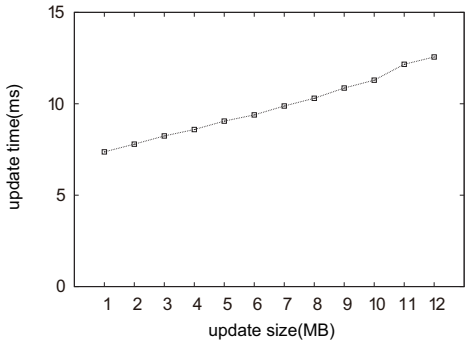
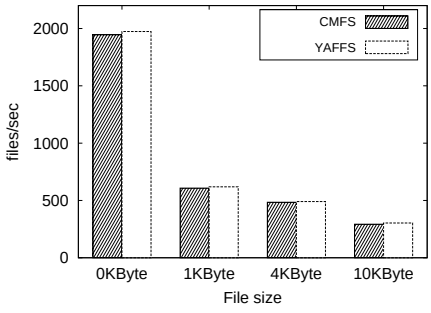
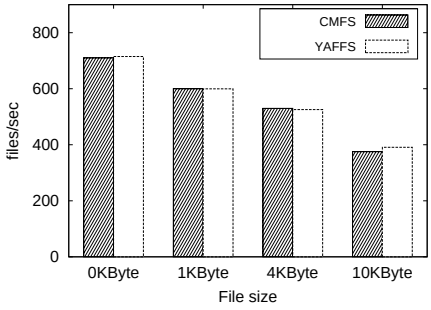


Figure 14: Update time

We measured the latency of the update procedure which is invoked every five seconds. Figure 14 shows the update time varying with the size of modified file data. In this experiment, there are already 10 MB files in the CMFS partition. In our environment, 12 MB of data can be modified within five seconds at most. Consequently, the update procedure takes 13 ms in the worst case. This means that CMFS sacrifices 2.6% of the performance to keep the BPRAM metadata valid.



(a) File Creation



(b) File Deletion

Figure 15: Metadata update performance (LMBENCH)

8.6. I/O Performance

In CMFS, the metadata of a file system is periodically compressed and stored into the BPRAM. In order to examine the performance impact of the periodic compression and store, we measured file creation/deletion performance and write performance. We used LMBENCH to measure the file creation/deletion performance. Figure 15a and figure 15b illustrate the file creation performance and the file deletion performance. We performed the experiment by creating and deleting 0 KByte, 1 KByte, 4 KByte, and 10 KByte files. Figure 16 illustrates the write performance examined by IOZONE benchmark. Throughout I/O performances of CMFS, the performance impact of the metadata compression was insignificant.

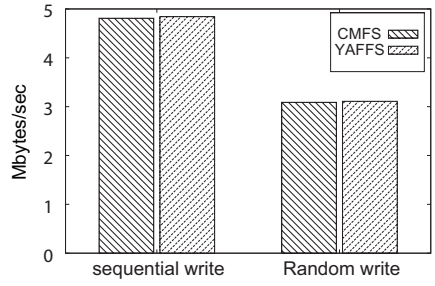


Figure 16: Sequential/Random write

9. Conclusion

In this work, we develop a hierarchical file system, CMFS for BPRAM and NAND Flash. The contribution of the CMFS file system is as follows. First, we develop a hybrid coding to rapidly compress the metadata reducing the size effectively. Second, CMFS provides crash recovery methods to reduce the mount time after the crash. Third, the overhead of compression and decompression is insignificant in CMFS. In summary, we successfully developed a state of art hierarchical file system. With a slight performance loss, CMFS provides fast mount speed and high CPM. In addition, the recovery method of CMFS makes it possible to provide fast mount speed even after a crash.

Acknowledgements

This research was supported by the Korea Science and Engineering Foundation (KOSEF) grant funded by the Korea government (MEST) through National Research Lab (R0A-2010-0018893).

References

[1] *Basic Compression Library http://bcl.comli.eu/*.

[2] *Markus F.X.J Oberhumer. LZO data compression library http://www.oberhumer.com/opensource/lzo/*.

[3] A. Bityutskiy. JFFS3 design issues.

[4] I. Doh, J. Choi, D. Lee, and S. Noh. Exploiting non-volatile RAM to enhance flash file system performance. In *Proceedings of the 7th ACM & IEEE international conference on Embedded software*, page 173. ACM, 2007.

[5] S. Eaton, D. Butler, M. Parris, D. Wilson, and H. McNeillie. A ferroelectric nonvolatile memory. In *Solid-State Circuits Conference, 1988. Digest of Technical Papers. ISSCC. 1988 IEEE International*, page 130. IEEE, 2009.

[6] N. Edel, D. Tuteja, E. Miller, and S. Brandt. MRAMFS: a compressing file system for non-volatile RAM. In *The IEEE Computer Society's 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 2004.(MASCOTS 2004). Proceedings*, pages 596–603, 2004.

[7] R. Freitas, W. Wilcke, and B. Kurdi. Storage class memory, technology and use. In *Tutorial, 6^{th} USENIX Conference on File and Storage Technologies*, San Jose, CA, USA, 2008.

[8] W. Gallagher, D. Abraham, S. Assefa, S. Brown, J. DeBrosse, M. Gaidis, E. Galligan, E. Gow, B. Hughes, J. Hummel, et al. Recent advances in MRAM technology. In *VLSI Technology, 2005.(VLSI-TSA-Tech). 2005 IEEE VLSI-TSA International Symposium on*, pages 72–73. IEEE, 2005.

[9] J. Jung, Y. Won, E. Kim, H. Shin, and B. Jeon. FRASH: Exploiting storage class memory in hybrid file system for hierarchical storage. *ACM Transactions on Storage (TOS)*, 6(1):1–25, 2010.

[10] H. Kim and Y. Won. MNFS: mobile multimedia file system for NAND flash based storage device. *Korea*, 442:600, 2009.

[11] S. Lai. Current status of the phase change memory and its future. In *Electron Devices Meeting, 2003. IEDM'03 Technical Digest. IEEE International*, pages 10–1. IEEE, 2004.

[12] E. Miller, S. Brandt, and D. Long. HeRMES: High-performance reliable MRAM-enabled storage. In *Proceedings of the 8th IEEE Workshop on Hot Topics in Operating Systems (HotOS-VIII)*, pages 83–87. Citeseer, 2001.

[13] A. Olson and D. Langlois. Solid State Drives (SSD) Data Reliability and Lifetime. 2008.

[14] A. One. YAFFS: Yet another flash filing system. *Electronic document available online at http://www. aleph1. co. uk/yaffs/index. html. Cambridge, UK*, 2002.

[15] S. Park, T. Lee, and K. Chung. A Flash file system to support fast mounting for NAND Flash memory based embedded systems. *Embedded Computer Systems: Architectures, Modeling, and Simulation*, pages 415–424, 2006.

[16] S. Park, J. Yu, and S. Ohm. Atomic write FTL for robust flash file system. In *Consumer Electronics, 2005.(ISCE 2005). Proceedings of the Ninth International Symposium on*, pages 155–160. IEEE, 2005.

[17] Y. Park, S. Lim, C. Lee, and K. Park. PFFS: a scalable flash memory file system for the hybrid architecture of phase-change RAM and NAND flash. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 1498–1503. ACM, 2008.

[18] T. Quan, D. Yeo, and Y. Won. Cmfs: Compressed metadata file system for hybrid storage. In *Proceeding of 2010 IEEE International Conference on Network Infrastructure and Digital Content (IC-NIDC2010)*, Beijing, China, September 2010.

[19] S. Raoux, G. W. Burr, M. J. Breitwisch, C. T. Rettner, Y. C. Chen, R. M. Shelby, M. Salinga, D. Krebs, S. H. Chen, H. L. Lung, and C. H. Lam". Phase-change random access memory-a scalable technology. *IBM Journal of Research and Development*, 2008.

[20] S. Tehrani, J. Slaughter, E. Chen, M. Durlam, J. Shi, and M. DeHerren. Progress and outlook for MRAM technology. *Magnetics, IEEE Transactions on*, 35(5):2814–2819, 2002.

[21] A. Wang, P. Reiher, G. Popek, and G. Kuenning. Conquest: Better performance through a disk/persistent-RAM hybrid file system. In *Proceedings of the 2002 USENIX Annual technical Conference*, pages 15–28, 2002.

[22] C. Wu, T. Kuo, and L. Chang. The Design of efficient initialization and crash recovery for log-based file systems over flash memory. *ACM Transactions on Storage (TOS)*, 2(4):467, 2006.

[23] K. Yim, J. Kim, and K. Koh. A fast start-up technique for flash memory based computing systems. In *Proceedings of the 2005 ACM symposium on Applied computing*, page 849. ACM, 2005.